

Bachelor Thesis

Enabling the creation and hosting of cooperative online escape events in M.O.R.S.E. without programming experience

Elwin Duinkerken
Gijs Groenewegen
Wessel Thomas
Bram Verboom
Timo Verlaan

in affiliation with Raccoon Serious Games

Bachelor Thesis

Enabling the creation and hosting of cooperative online escape events in M.O.R.S.E. without programming experience

by

Elwin Duinkerken
Gijs Groenewegen
Wessel Thomas
Bram Verboom
Timo Verlaan

to obtain the degree of **Bachelor of Science**
in Computer Science and Engineering
at the Delft University of Technology,
to be defended publicly on Wednesday July 1, 2020 at 12:00 AM.

Project duration:	April 20, 2020 – July 1, 2020	
Thesis committee:	T.A.R. Overklift Vaupel Klein, ir.	TU Delft, supervisor
	H. Wang, dr. ir.	TU Delft
	J.W. Manenschijn, BSc	Raccoon Serious Games, client

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

Preface

This is the report for the Bachelor End Project created by Elwin Duinkerken, Gijs Groenewegen, Wessel Thomas, Bram Verboom and Timo Verlaan as a part of the Bachelor Computer Science program at Delft University of Technology. Over the course of ten weeks, we have extended an existing system for creating and managing escape events in such a way that the production and hosting of external web pages is possible. This project was commissioned by Raccoon Serious Games.

Our deepest thanks to everyone at Raccoon Serious Games for not only all the support, advice and fun times they gave us during the project, but also for putting in the effort to really make us feel part of the team while we were there, with various activities, weekly (digital) drinks and transforming the office to make it possible for us to visit in the trying times caused by COVID-19. A special thank you to co-founder Jan-Willem Manenschijn, for taking the time to coordinate, evaluate and validate all the work we put in during these ten weeks, sculpting the product to fit the needs of Raccoon Serious Games.

Finally, we would like to thank our coach ir. Thomas Overklift Vaupel Klein for always finding time in his busy schedule to guide our process, provide feedback where it was needed and offer his assistance when required.

*Elwin Duinkerken
Gijs Groenewegen
Wessel Thomas
Bram Verboom
Timo Verlaan
Delft, June 2020*

Summary

The M.O.R.S.E. system is a tool for creating and managing large escape events, mainly used for local escape events. The tool is designed for only a limited range of puzzle types and styling options because most of the puzzles require physical items in order to solve a puzzle and only the answers have to be entered in M.O.R.S.E. Because of this design, it is really difficult to create online escape experiences, especially rich and immersive ones. It also requires a lot of programming outside of the M.O.R.S.E. system to do so. Raccoon Serious Games, the client, does not have many employees with programming experience and, therefore, it is not feasible for them to create the rich and immersive online escape experiences they want.

To be able to create such immersive experiences, we are extending M.O.R.S.E. with editable domains and web pages. Game designers can add domains and web pages to the existing event schedule and then puzzles can be created for web pages. Players can view one or multiple of these domains and for each domain, the active web page will be served.

Web pages can be created and stored in the domains, but the actual contents of the web pages still have to be made. Because making web pages is often a programming intensive task, a page builder has been created in M.O.R.S.E. This page builder allows the user to load and save web pages created in the M.O.R.S.E. system. It uses a drag-and-drop system to place building-block elements inside the web pages and allows for directly visible styling of those elements. Because of this, the user does not need programming knowledge of the underlying implementation of the web pages. It also facilitates the linking between M.O.R.S.E. features and the domains such as puzzles and triggers for buttons. Using the import and export functionality, users can easily copy previous web pages created with the page builder. This is not only limited to internal web pages but can also be used to import external code from outside the page builder. With user-friendly features such as the ability to undo and redo changes, the page builder tries to make creating web pages as easy as possible.

An important aspect of the escape games hosted by Raccoon Serious Games is team building. We extend upon this notion by adding roles and a leaderboard screen to M.O.R.S.E., both of which increase the need and opportunity for interaction between players. The addition of roles allows game designers to enforce cooperation in their escape events, by restricting the access to resources required for solving a puzzle to only a subset of the players in a team. This way they have to cooperate and combine their information and resources to solve all puzzles. The addition of leaderboards is also an extra incentive for a player in a team to work together efficiently because this will positively impact their score and, therefore, ranking on the leaderboard.

Contents

1	Introduction	1
2	Problem Research	3
2.1	Problem context	3
2.2	Analysis	3
2.2.1	Online escape experiences	3
2.2.2	M.O.R.S.E.	4
2.3	Definition of the sub-problem	5
3	Design	7
3.1	Domains	7
3.1.1	Schedule	7
3.1.2	Internal and external domains	8
3.1.3	Web pages	8
3.2	Player interaction	9
3.2.1	Roles	9
3.2.2	Leaderboard	11
3.3	Page builder	11
3.3.1	WYSIWYG editor	11
3.3.2	BuilderLibrary	12
3.3.3	Linking puzzles in the page	13
3.4	Hosting	14
3.5	Extensibility	15
3.6	Backwards compatibility	15
3.6.1	Roles	15
3.6.2	Domains	16
3.6.3	Page builder	16
3.7	Summary	16
4	Implementation	19
4.1	Current implementation	19
4.1.1	Architecture	19
4.1.2	Frameworks	20
4.1.3	Components	20
4.2	Domains	21
4.2.1	Web pages	21
4.2.2	Users	21
4.2.3	Additions to rulesets and schedules	22
4.2.4	Puzzles	23
4.3	Player interaction	23
4.3.1	Roles	23
4.3.2	Leaderboard stage	28
4.4	Page builder	28
4.4.1	Page builder tab	28
4.4.2	Connection to M.O.R.S.E.	30
4.4.3	Builder library	31
4.5	Summary	33

5	Testing	35
5.1	Verification	35
5.1.1	Unit tests	35
5.1.2	Manual tests	35
5.1.3	Backward compatibility	35
5.1.4	Scalability test	36
5.1.5	Static analysis	36
5.2	Validation	37
5.2.1	Weekly client demonstrations	37
5.2.2	Playtests	37
5.3	Summary	38
6	Process	39
6.1	General intentions	39
6.2	Workflow	40
6.2.1	Engineering methodology	40
6.2.2	Task division	40
6.3	Setbacks	40
6.4	Retrospective	41
7	Discussion	43
7.1	Does it meet the requirements?	43
7.1.1	Functional	43
7.1.2	Non-functional	43
7.2	Does it solve the problem?	44
7.2.1	According to us	44
7.2.2	According to the client	44
7.3	Ethical implications	45
7.3.1	Private user data	45
7.3.2	URL masking	45
8	Conclusion	47
9	Recommendations	49
9.1	Graphical User Interface (GUI)	49
9.2	Scalability	49
9.2.1	Roles	49
9.2.2	Domains	49
9.2.3	Page builder	50
9.3	Future improvements	50
A	Project plan	53
B	Requirements	57
C	Research	63
D	Admin manual	79
E	Scalability Test	99
F	Project description	101
G	Playtest Webshop	103
H	DigiHacked Playtest	107
I	Infosheet	109
	Bibliography	111

1

Introduction

Raccoon Serious Games is a company that develops and organises escape room inspired events for large groups, as well as smaller groups [1]. Their escape events are more generally called 'Serious Games'. Serious Games are not only designed to entertain but also to educate or create awareness of a certain topic [2]. The goal of these events is for players to learn something from these experiences and to stimulate team building. If such events can incorporate online activities, such as gathering information from websites or entering login credentials, this could be a valuable expansion of the possibilities of escape events.

Our Bachelor End Project (BEP) focuses on the existing Massive Online Reactive Serious Escape 2.0 (M.O.R.S.E.) system where escape game events can be created, updated and hosted. It is designed for the administrators that develop the events, the game hosts and also the players of the events. Puzzles, logistics and other essential components for escape events can be defined in the system. However, creating an online escape experience is not within the realm of possibilities. An example of such an experience is called 'DigiHacked'¹. The participants learn about the danger of scamming and hackers while playing the game, which can be done entirely online. The production of external web pages that are used in such an event and communicating with them is very troublesome in the current version of the M.O.R.S.E. system and it is not possible to track the progress of the players in puzzles on external web pages. It would, therefore, be ideal if M.O.R.S.E. could be upgraded in such a way that users would be able to create online escape experiences like DigiHacked.

In this final report, we will share the findings, achieved results and resulting conclusions of our project. To first give a good view of the problem, we define the issues at hand and examine the M.O.R.S.E. system along with the given example of an online escape experience called DigiHacked. This is all covered in section 2, to create a better understanding of what is currently missing in the current system and what this project will roughly include. After the required background in the previous chapter, we present the abstract layer of our solutions to the problems stated in section 3 and discuss the way these solutions are going to be woven into the existing system. After the abstract layer, we show and discuss the concrete layer of implementation in section 4 where more technical details are given as to how we engineered the system to induce the new functionalities. Of course, we have to show that the new functionalities do consistently what is intended, for which we look at section 5. This section gives the details of our testing processes and how we involved our client as well. After the previous three sections that explain what we did and how we did, we take stock in section 7. We review the process we made in comparisons with our expectations set at the start of the project and discuss the extent we met the demands of our client, Raccoon Serious Games. Ethical implications of our process are put under further scrutiny and the last part of the section is regarding the group dynamic and the approach that was taken towards development during the project are also be highlighted. To bring a close to the report, we give our thoughts and recommendations of the final version of the system, regarding its current capabilities, its limitations and what extra features could be added in future endeavours to further improve M.O.R.S.E. 2.0.

¹<https://digi hacked.raccoon.games/>

2

Problem Research

2.1. Problem context

M.O.R.S.E. is a tool for creating and hosting escape events. In these events, the participants are divided into teams where they receive a set of questions and puzzles that needs to be solved to complete the event. For certain puzzles, teams can receive physical items from the project staff that contain the necessary information to find a solution. Raccoon Serious Games developed one of these escape events outside of the M.O.R.S.E. system called DigiHacked. DigiHacked is an online escape experience that provides an immersive experience that shows us the dangers of the online world.

DigiHacked is a great example of an online escape experience that reveals what is lacking in the current M.O.R.S.E. system. We will discuss online escape experiences more in detail in section 2.2.1. While playing DigiHacked, you are guided through several external websites that serve as different front-ends, each of which is providing their respective puzzles. In contrast to events made in M.O.R.S.E. , where all puzzles and questions are shown only on the M.O.R.S.E. front-end, DigiHacked provides an additional layer of reality, due to the realistic websites. In DigiHacked, there is also a lot more emphasis on teamwork, since it is necessary to properly cooperate with each other in order to solve the puzzle. Nobody is able to solve the puzzle alone, whereas events created in M.O.R.S.E. are solvable if you are alone. With all this in mind, we find that the main question that we need to answer is; **How can we facilitate the creation of interactive puzzles on immersive web pages in M.O.R.S.E. that have a good emphasis on cooperation where programming experience is not required?**

2.2. Analysis

Since this is a very broad question, we needed to split it up into different smaller problems. To properly define these smaller problems, we first needed to dive further into online escape experiences and the M.O.R.S.E. system, so that it is clear what we are dealing with and to get a more detailed view of the tasks needed to be done and how we came to conclude what the sub-questions were that needed to be asked in order to solve the main question.

2.2.1. Online escape experiences

The final goal of the project is to be able to implement online escape experiences such as DigiHacked in the M.O.R.S.E. system without the need for any programming experience. To be able to create such experiences, the functionality of DigiHacked had to be investigated at the start of the project.

DigiHacked is an experience that aims to teach about phishing and similar malicious actions by tricking players into being the victim of one of these actions. Afterwards, these malicious actions are explained and ways of identifying them are discussed. For this experience, players form teams and receive only part of the information that is needed to solve the task at hand. Although DigiHacked is just one specific example of such an experience, it contains many key features that are also important for other online escape experiences . Instead of focusing on the specifics, we will explain the main flow of the game for a player. For a more de-

tailed explanation including the different types of attendees and their contributions to DigiHacked please have a closer look at section C.3 in Appendix C.

Player experience

Once a player has received their player code from the host, they can use this to log into the system. The logged-in player can then define their name. Each of the players is assigned to a team and the players in this team can specify their team name. The teams each will go through the same experience. Before the game has started the players can read a quick introduction to the game. After the host starts the game, the player sees the main area of the game.

For each group, there are certain player roles to be assigned. Each player gets assigned one or multiple roles within the game, dependant on how the number of players and the number of roles. These roles define what systems that player can access. For example, a security guard has access to security cameras in a room. This separation of roles makes communication between the different players crucial.

The main area of the game is a visual screen where each player can access a few different front-ends. These front-ends range from simple text views to front-ends where there is some user interaction built-in. An example would again be the security camera feed, where the player can rotate the camera or zoom in to focus on a certain clue. In addition to front-ends inside the main interface of the game, players must also interact with external sites. The actions of the players on these external sites are linked back to the team they belong to. Also, previous actions of someone in a team influences the game behaviour for all of them. For example, when one player in the team transfers a sum of money, all the players have to accept this transfer. In one of the front-ends, the player could get sent a link to their mobile phone by entering their phone number. The link was then sent as an SMS, to allow the player to use their phone's camera for two-factor authentication for a certain web app. Once a team completes the game they can see the leaderboard with the scores of all teams. In the case of DigiHacked, it is shown whether each team has transferred the money to the correct company or a group of hackers. Lastly, when the host has started the playback of a video to provide more background information, all players are redirected to the screen where that video is playing.

2.2.2. M.O.R.S.E.

M.O.R.S.E. is the system currently used to create and manage escape room events for large groups of people. M.O.R.S.E. excels at escape events at physical gatherings. There are multiple roles for such an event. There are the teams, which are solving the puzzles. All the teams are divided into clusters. For each cluster, there is an available foreman that tracks the progress and gives hints. Then there is the project desk staff. They help the foreman and hand out physical puzzle pieces to each team. Finally, there is also the host and the admin, who have control over the event while it is running. They can change the puzzle order and can decide what should happen with wrong and correct answers. They are in full control of M.O.R.S.E. while it is running. The difference between a host and an admin, is that the admin is the only person who can create and delete an event. All the staff allows for the experience of people solving puzzles in teams and entering their answers on the M.O.R.S.E. event website to progress through the event and complete more puzzles.

All the logic about the transitions and availability of puzzles during an event is handled by 'rulesets' in the M.O.R.S.E. system. The ruleset is a list of rules about the event, automatically executed by M.O.R.S.E. during the event. Every rule is divided into three parts: the triggers, the conditions and the actions. Firstly, the triggers define when the rule should be checked. An example of that would be: when the host pauses the event, a certain rule should be checked and possibly executed. Then the actions in M.O.R.S.E. make modifications to the state of the event. For example, teams can be moved to different screens and hints can be given to teams. The possible set of actions is predefined and can be set to apply to different targets in the event. For every combination of a trigger with one or more actions, extra conditions can be set in place to further specify how your event should behave.

The schedule available in M.O.R.S.E. gives an overview of the, often very large, events as well as handle the event flow. It shows the order of screens and groups of puzzles which can be made available for teams, clusters or even everyone in the event.

As with the previous section 2.2.1, more detailed information regarding all that M.O.R.S.E. can do, can be

found in appendix C.6 in C.4. If you are more interested in technical details regarding the implementation, you can skip forward to section 4.1, where we address how M.O.R.S.E. is currently structured and what frameworks are used.

2.3. Definition of the sub-problem

Now that we have more detailed information about the context of the problem, we are now ready to split the whole problems up into smaller problems.

The first problem we need to consider is ensuring players can access the different front-ends and that they can interact with them, just as we have seen in DigiHacked. The current M.O.R.S.E. system can track progress and process player input. With the use of multiple front-ends on external websites, tracking and processing are not as straight-forward as it is now. These front-ends should all have access to the same data so they too can remember which groups completed what puzzles and which stage of the event they should show to a certain group. They should also be able to accept input from the players and have the M.O.R.S.E. system check that input. To successfully deploy online escape experiences like DigiHacked in M.O.R.S.E. after this project, an answer is required for the first sub-question; **How do we link the external websites to M.O.R.S.E. and track the players' progress across these multiple websites?** The idea's of how we arranged the domains is covered in section 3.1 with the low-level explanation of the implementation given in section 4.2.

The second matter at hand is the adjustment to the player interaction that needs to be done. To be able to create a competitive environment between teams, while also having the opportunity to enforce teamwork between participants. For these stated aspects, the sub-question is asked; **How do we enforce teamwork and incorporate competitive elements in online escape experiences?** The concepts of how we wanted to include this into the existing system are discussed in section 3.2 whereas exact implementation and workflows are debated in 4.3.

The third problem is the production of the external websites that are used as front-ends for the participants. Developing the websites for online escape experiences like DigiHacked did require a fair amount of programming experience. DigiHacked has shown that Raccoon Serious Games is perfectly capable of creating such an event, yet a more efficient workflow would be preferable, due to the small number of employees with programming experience. Therefore, we had to find an answer to the third sub-question; **How do Raccoon employees develop a customised external website, without having to rely on programming experience?** Trying to answer this question, we need to keep various aspects in mind that need to be dealt with, such as the difficulty of usage and the easy development of prototypes to allow for a flexible workflow. The solution we opted for is a self-developed page builder. The concepts of this page builder are explained in section 3.3, with a more technical explanation of the implementation in section 4.4.

Based on the analysis we did on M.O.R.S.E. and online escape experiences along with our defined problems, we put together a list of requirements for the entire project that build towards a product that solves these problems. We made different list of requirements with priorities for each type of stakeholder within a M.O.R.S.E. event, along with a general distinction between functional and non-functional requirements. The entire list can be found in appendix C.10.

3

Design

This chapter outlines the design decisions that are made based on the requirements that were gathered during the research phase of the project but also taking into account how to find the best possible answer to the earlier specified (sub)questions in section 2. Ease of use, maintainability, extensibility, backwards compatibility and the amount of programming experience required are kept in mind throughout the design of the project. In addition, we made sure to keep the design of the functionality to be added consistent with the existing functionalities.

In the previous chapter, we stated several questions. The proposed solutions for the first, second and third sub-question are described in respectively section 3.1, 3.2 and 3.3. Then to complement the solution for the first sub-question, we will introduce our conceptual design for hosting all the different front-ends that are going to be present in the future. With the future in mind, we also explain in 3.5 how we are going to make sure that the extensibility of the system is in decent shape.

3.1. Domains

In this section a design is presented that will help answer the sub-question how external websites could be linked to M.O.R.S.E. and how to track their progress across them. One of the missing features that is needed to design the online escape experiences without programming experience is the ability to create different front-ends. The front-ends have both a structure inside M.O.R.S.E. as well as their visual elements. This section highlights the structure of these front-ends, the visual aspect is explained later in section 3.3. In M.O.R.S.E., there are already M.O.R.S.E. pages which can be used to create somewhat more custom layouts. However, for the use case of custom created front ends they are too limited. These M.O.R.S.E. pages are static and adding puzzles to them is impossible. In addition, these M.O.R.S.E. pages are always rendered with the M.O.R.S.E. layout surrounding them, see figure 3.1 for an example of such a page. This makes them less suitable to use for fully immersive experiences. To keep backwards compatibility with events that do use these M.O.R.S.E. pages, a new system of web pages is needed while keeping the old one intact.

To make a distinction between different front-ends as described in 2.2.1, domains are chosen. For the players, this is intuitive as they are familiar with the concept of domains or websites already, since websites on the internet are structured the same way. The game designers are familiar with domains in the same way, and an additional level is added for web pages. Domains are a collection of web pages, where each web page could represent a view of an application. This is in line with most domains where each domain represents a different application.

3.1.1. Schedule

In M.O.R.S.E. , there exists a schedule where all the different views which players can see are represented. Within these separate views, it is also specified where certain questions are asked to the players. Since these views are very similar to the domains and web pages, they can be extended using inheritance to hold new schedule types for domains and web pages. This design makes it easy for existing game designers to be able to create new domains and web pages because they are already familiar with the schedule editor. In figure

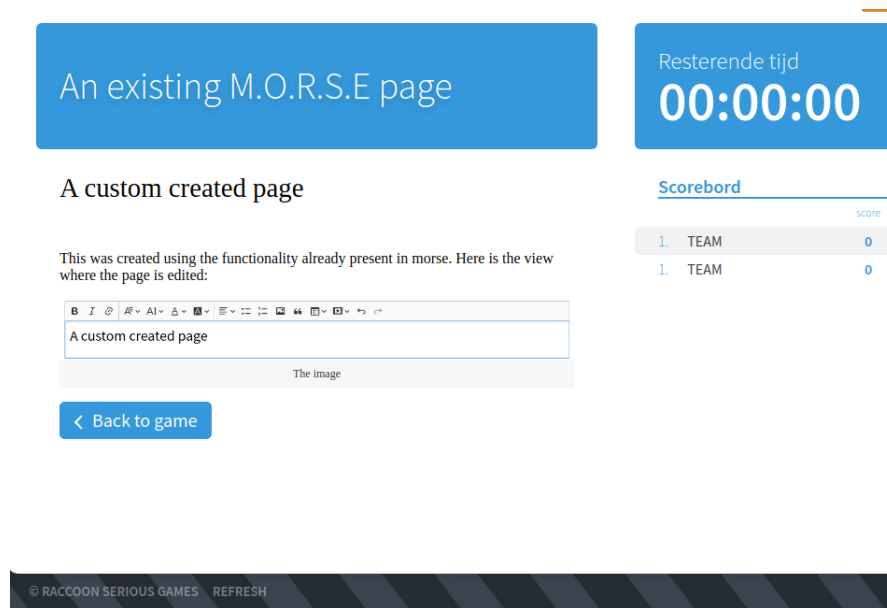


Figure 3.1: An example of an existing M.O.R.S.E. page

3.2, a sketch of how the domains and web pages would fit into the current schedule is drawn.

3.1.2. Internal and external domains

There are two domain types, internal and external. The term domains in this case, refers to the simulated domain by M.O.R.S.E. to give the impression the player is on a real domain. Internal domains can be viewed with the existing M.O.R.S.E. view surrounding it. The internal domains are easier to navigate to and from because the M.O.R.S.E. layout is still present. However, the layout makes it less immersive. To increase immersion, the external domains type exists. When the user visits such an external domains, none of the M.O.R.S.E. layout is present. This allows game-designers to create fully custom experiences.

3.1.3. Web pages

Web pages are part of the schedule as well, but they must always be inside a domain schedule. Since domains are a collection of web pages, it is possible to add multiple web pages to a single domain. This creates the link between the domains and web pages. Although the web pages is created here, the look of the page can only be changed in the page builder. In the existing M.O.R.S.E. schedule a player can only be in one schedule at the same time. Different domains can be viewed at the same time, for example with multiple tabs or windows. In addition, players can view multiple domains at the same time, since for certain puzzles you have to combine information from multiple domains to find the solution. Although multiple domains can be viewed at the same time, only the active web page of each domains is visible to the player. This allows for moving players from one web page to another with the rulesets just like the existing schedule-items, although with a different yet similar looking action.

Web pages are also a collection themselves, since multiple puzzles can be added to the pages. When creating a puzzle inside a web page, the visual elements are not added to the page yet. Since the program could not possibly know where the game designer would want to place the puzzle on the page, this is a deliberate choice. To keep the editing of the visual elements of web pages in one place, the elements of the puzzles can be added in the page builder.

This brings the design back to the sub-problem of how the players' progress across web pages can be tracked. The puzzles on the pages allow the system to track the progress through the system by tracking their progress across the puzzles on the web pages.

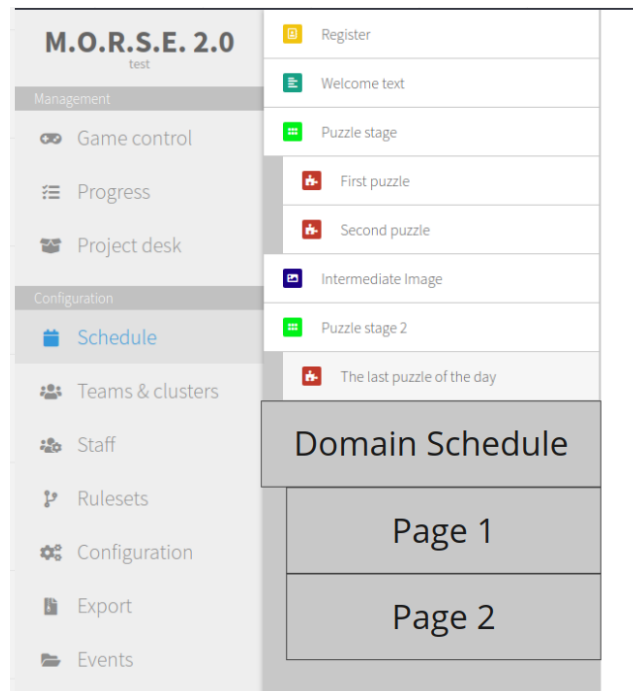


Figure 3.2: The design of domains and web pages within the schedule

3.2. Player interaction

In trying to answer our second sub-question ‘How do we enforce teamwork and incorporate competitive elements in online escape experiences’, as stated in section 2, we now discuss our design for such competitive and cooperative elements in M.O.R.S.E. By enabling a game designer to enforce teamwork and incorporate competitive elements, the resulting games could also prove to be a great team-building experience, which is an important goal of the escape events designed and hosted by Raccoon Serious Games. The leaderboards described at the end of this section provide exactly such a competitive experience that is required for a team-building exercise, because players can only get to the top of the leaderboard by working together. This is aided by the addition of roles, that we discuss below. Roles allow the game designer to restrict the access to specific resources for some players in a team, which means teams have to cooperate and combine their information in order to solve the challenges presented to them.

3.2.1. Roles

As described in section 2.2.2, one of the missing features in the current version of M.O.R.S.E. is the ability for players to be assigned roles which dictate their possible actions within the game. By incorporating such functionality, especially in combination with the new domains, immersive gameplay experiences can be created that force players in a team to cooperate. To this end, we first study the use case of the aforementioned roles in DigiHacked, in order to identify what the requirements are for this module. Then we specify the requirements for a more generic use case and identify the different perspectives in this part of the application: the player and the game designer.

Specific use case: DigiHacked To elaborate on why such roles are required, consider the specific case of DigiHacked, where players work together in teams to transfer money to another hypothetical company, for which they have to log in on several sites and collect the information and clues necessary to do so. DigiHacked has already been discussed in section 2.2.1, so in this paragraph, we only focus on its player role aspect. A key element in the cooperation between players is the fact that each participant only has a limited subset of all available information in the game. Only by working together and combining this information can the puzzles be solved and the game completed. Some of the roles in DigiHacked include the project manager, who, among other things, has access to a memo describing what the objective of the game is, and a security guard who has access to a virtual environment showing a security camera feed. These examples clearly illustrate the need to be able to restrict access to specific resources for players, based on their role within the game.

Generic use case Having identified the way player roles are used in DigiHacked, we can now establish a more generic set of requirements where the implementation in section 4.3 has to comply with. Since the set of requirements depends on whether the user is designing or playing the game, the design for this feature set is split up in two parts; the part of the player, and the part of the game designer. When looking at the player perspective, the requirements are fairly straightforward because all the player will notice about the role aspect of the system, is whether or not access to a requested resource is granted. In addition, the player should also be able to identify what their role is within the game. It is then up to the game designer to make sure that the corresponding permissions actually make sense. For example, it would hardly make sense to the player with the role of the project manager to be denied access to the mail account of the project manager. Below we discuss the requirements for both the player and the game designer.

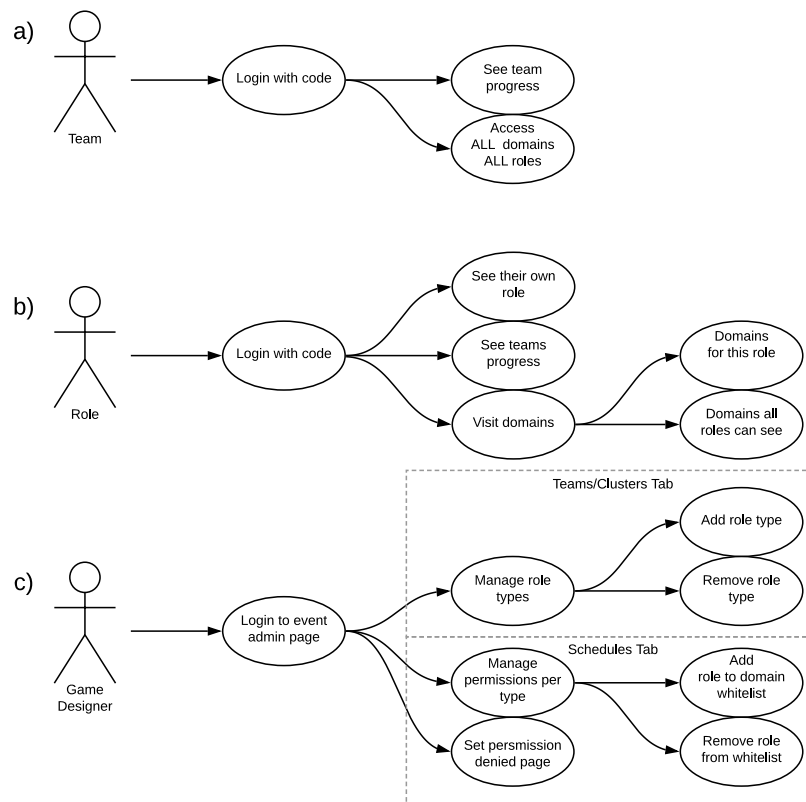


Figure 3.3: UML Use case diagram illustrating the design of all human actors required in the roles component of the system. a) the team that is already present in the current version of M.O.R.S.E. b) the role a player can login to and interact with during the game. c) the game designer responsible for setting up and configuring roles.

In order for the system to know if a player has access to a resource, the player should be uniquely identifiable. This can be done in either of two ways: every player can be provided with a unique login code, which is then coupled to a type of role (either automatically or by letting the player choose for themselves); or by letting all players in a team log in with the same credentials and then storing the role of the player in the session. A disadvantage of the latter option is, that if the player somehow closes the session and opens a new one (for example if an error occurs on their device) they have to log in again, and the system will no longer know what their role is in the game. An advantage, however, would be that all players log in with the same code, which is much easier from an organisational perspective. Despite this, the former method, illustrated in figure 3.3b, is adopted because it is much more compatible with the system already in place. However, since we kept the backward compatibility of the product in mind, we also left the option in to have all team members with the team code, meaning no one has a role and everyone has access to all domains, which is represented in figure 3.3a.

The requirements for the game designer are more complex because they are responsible for defining the roles and their permissions within the game. First of all, the game designer should be able to add and remove roles from the game. Rather than creating an entirely new web page in M.O.R.S.E., an obvious place to provide this functionality is the 'Teams / Clusters' tab that already exists in M.O.R.S.E., since the management

of roles is closely related to the management of players. In addition, game designers should also be able to specify for each role, what corresponding resources should be granted access to. For this too, could a new tab in M.O.R.S.E. be created, but it would make more sense to do this in the tab where resources are managed, which is in the 'Schedules' tab. In the 'Schedule', the game designer should be able to specify for each relevant schedule and schedule item what roles are allowed to access it. If a role is not allowed to access a schedule, it should also not be able to access the nested schedules and schedule items, because they will be impossible to reach in the first place. Although this functionality could technically be implemented for all types of schedules and schedule items, the client indicated this is only necessary for the domains and web pages described in section 3.1, which is also what we will adhere to. The entire design of the game designer perspective is shown in figure 3.3c.

3.2.2. Leaderboard

It is a huge addition to an event if everyone present gives it their all to win. No one wants to cooperate in a team where half the members can't be bothered to do anything. For a lot of people it helps a lot if there is a sense of competition, or that there is something to be won or lost. A frequently shown leaderboard that engages and excites all the participants to try even harder to win, can contribute to that competitive setting. However, there is already a projector where a leaderboard is shown, and there is also a leaderboard shown while you are answering question in M.O.R.S.E. So why do we need another one?

The current leaderboards are quite basic and not very easy to customise per player. The best you can do now is a custom message that can be shown on the projector, however, that message takes the place of the leaderboard and is a message to everyone, not a personal one for your team. So the only way to show customisation, is to remove the leaderboard on the projector.

To this end, we designed a solution to have a leaderboard possibly available at all times to the players, that could also contain customised messages without taking away all the already present elements. For this we want to recycle as much as possible from the leaderboard on the projector, since we do like the styling with which it was made.

3.3. Page builder

One of the defined subquestions in section 2 was how the game designer could be able to customise web pages without having to rely on programming experience. We created a new tab in the M.O.R.S.E. menu, where we implemented a page builder which is an editor that allows the user to design the style of the web pages from the schedule. The user should be able to create professional and realistic-looking web pages. In addition, it is important that the user can utilise the product with minimal to no programming knowledge and experience. This section describes the design of the page builder.

3.3.1. WYSIWYG editor

When creating a product for creating web pages that does not require programming experience, the WYSIWYG (What You See Is What You Get) principle is a possible solution. This means that the effects of the actions on the final result performed by the users are directly visible. Current editors that claim to be WYSIWYG often edit a copy of the data instead of the editing the data directly and can thus better be classified as 'What you see is what you *will* get' systems [3]. The kind of system that is directly editing the data is described by the TAXATA (Things are exactly as they appear) [3] principle.

For the page builder, the 'What you see is what you will get' version of the WYSIWYG principle is used. The web pages that the user is editing is a copy of the web page stored in the database on the server and is only updated when the user saves the web page. There are three main reasons for this. First, the user can experiment with design and changes on the web page and throw the changes away if the result of the experimenting is not to their liking. This would not be possible with the TAXATA principle because changes are saved on every action. Reverting to a working version you had before takes a lot of undo actions, which is not preferable because of the number of possible actions needed for reverting. This could be solved keeping track of saved versions of the work by the user, but this would require a lot of extra logic on the server. Second, updating every modification to the web page to the database on the server does massively increase the communication with the server. Since the M.O.R.S.E. system already has quite some latency, this is not

preferable, because increased communication with the server slows the system even more. The third and last reason is that with the current design of the domains, web pages saved in the database are automatically publicly visible for players in M.O.R.S.E. This is something that is not preferable when running an escape room experience and modifying the web pages, because players are able to see half working web page. One option is not modifying the web pages whilst running an event, however, this also eliminates or hardens the option to fix bugs in the web page during the event.

As said, the page builder edits a copy of the web page that is stored in the database. The web page is stored using HTML and CSS and is displayed to the user in a graphical representation, just as how the web page would look like when visited by the players of the online escape experiences. This follows the description of the WYSIWYG principle almost directly. However, there are two minor differences. One is that the web page is not viewed in the same dimensions as one would see it in a real page. This is because there is space around the edges for menu's that allow the modification and storage of the web pages in the page builder. This design for this is visualised in figure 3.4. The other difference is that the page builder adds some visuals to the web page to clarify the current selections by the user and to make some elements draggable. When clicking an element, it is surrounded by a green border and a description of what kind of element is selected.

Elements such as videos and audio, which have click functionality in them are difficult to select using the page builder. Therefore, a box with a drag icon has been placed in the top-left corner of the element to allow selection and dragging.

Those menu's allow for the user to visually drag and drop elements into the web page, as well as replace those elements, again conforming with the WYSIWYG principle. The modification of those elements is in line with the WYSIWYG principle, because changes made to the properties of the element are directly visible on the web page.

Because the user can directly edit the canvas on which the web page is drawn, they do not need the programming experience required to understand the underlying code. This same principle has been applied to the menus in the page builder as much possible.

3.3.2. BuilderLibrary

The existing M.O.R.S.E. website is built with Angular ¹. One way of implementing the page builder is to generate Angular components which can then be compiled and served to the users. The main benefit to this is that there is a perfect integration with the rest of the M.O.R.S.E. system, which already works with Angular. A problem with this approach is that the generated web page needs to be recompiled dynamically every time a web page is created or changed. One of the more bigger downsides, however, is that there is no easy way to import other web pages or templates that are not made with Angular. A major prerequisite for the web page, is that it follows the WYSIWYG design as discussed in the previous chapter 3.3.1. If the generated web page is so closely integrated into Angular, the other M.O.R.S.E. components might accidentally influence the look of the web page.

Because of these downsides, we decided to take a different approach for the page builder. For the final design, the page builder contains an iFrame ² which acts as a completely standalone web page. This means there is no accidental way the other components can influence the styling or behaviour, since all the CSS and the events are not propagated through the iFrame. The web page inside the iFrame only consists of normal HTML, CSS and Javascript without any frameworks. A benefit for this design is that it is trivial to import any existing outside web page or template as long as it only contains normal HTML, CSS and Javascript. The exact implementation can be found in chapter 4.4.

The page builder is designed in a way where all the logic is separate from the other Angular components. This separation makes it easier to control the flow of the program. All the changes that are made to the web page are executed via the BuilderLibrary. If this was not the case, it would be hard to add more complex features to the page builder, such as restoring the web page to a previous state. In our design, this is much easier to implement, since all the user actions are executed from the BuilderLibrary.

¹<https://angular.io/>

²<https://www.w3.org/TR/2011/WD-html5-20110525/the-iframe-element.html>

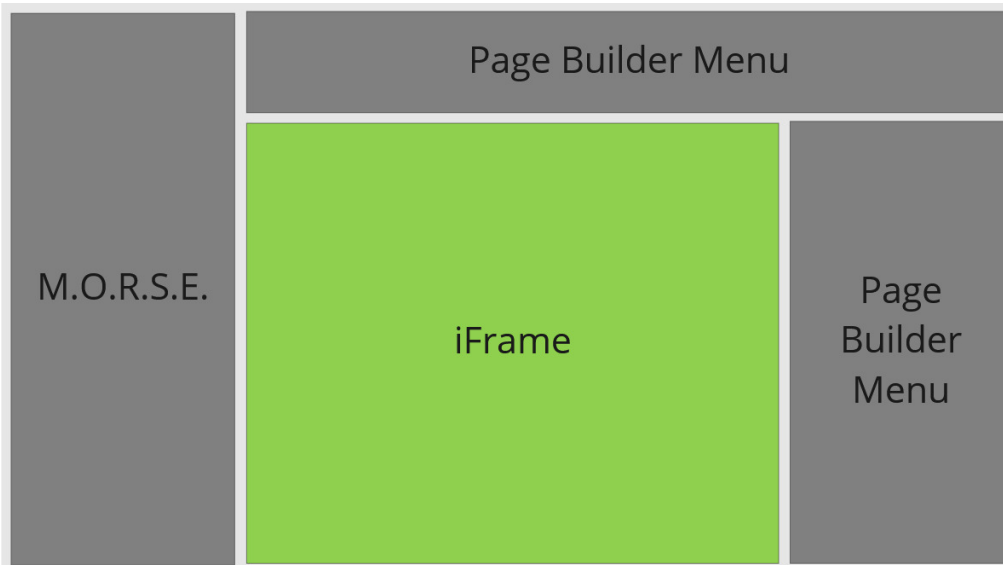


Figure 3.4: The design of the page builder UI.

The sidebar and the top menu of the page builder are made with Angular components, and never make any direct changes to the web page. If there is a change they want to make, a function is called in the BuilderLibrary which handles the request and makes sure it carries out the right behaviour. The BuilderLibrary can potentially call other modules internally. These internal modules are not accessible to the outside since they might accidentally be used inappropriately. A simple call to the BuilderLibrary might need to do multiple things to get the desired effect, such as remembering previous states of the web page, deselecting old elements or doing some cleanup of old memory.

The BuilderLibrary also contains some data that is available to the outside modules, such as the Angular components. All the shared data is accessible for the Angular components to see, but they should not change the values directly. The BuilderLibrary is responsible for setting the correct values. Some of the outside menu's might need to know when a certain event happens. This could be accomplished by letting the menu continuously check for if a certain value changes in the shared data. This is, however, not the design we think is most efficient and appropriate for this problem. Therefore, the BuilderLibrary can store certain event listeners, where outside modules can add listeners for a certain event. If the event occurs, the outside module is called by the BuilderLibrary with a specified callback function.

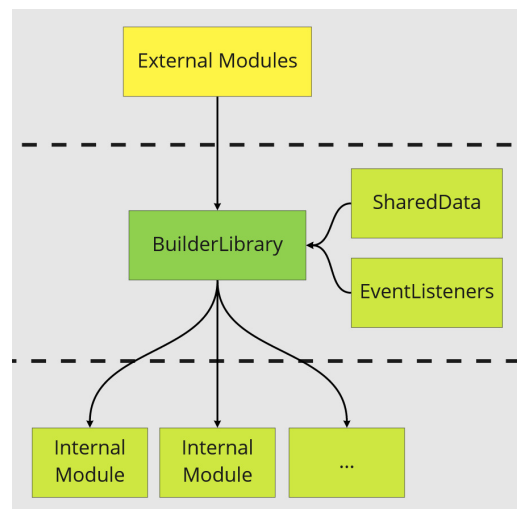


Figure 3.5: Overview of the BuilderLibrary.

3.3.3. Linking puzzles in the page

Maybe the most important requirement for the page builder, is the ability to have puzzles on the web page which can be solved by the players using the existing M.O.R.S.E. system. When a player wants to submit an answer to a puzzle on a web page, a message needs to be sent to the M.O.R.S.E. server. This means that a script needs to be executed on the client when for example a button on the web page is pressed. This script contains the code which sends a message to M.O.R.S.E. with the answer the player wants to submit. There are two possible moments where this script can be attached to the button. The first option is to attach the script to the button in the page builder already. This seems like a pretty straightforward solution because the page builder is the place where the actual button is created and the script is a part of the button. The

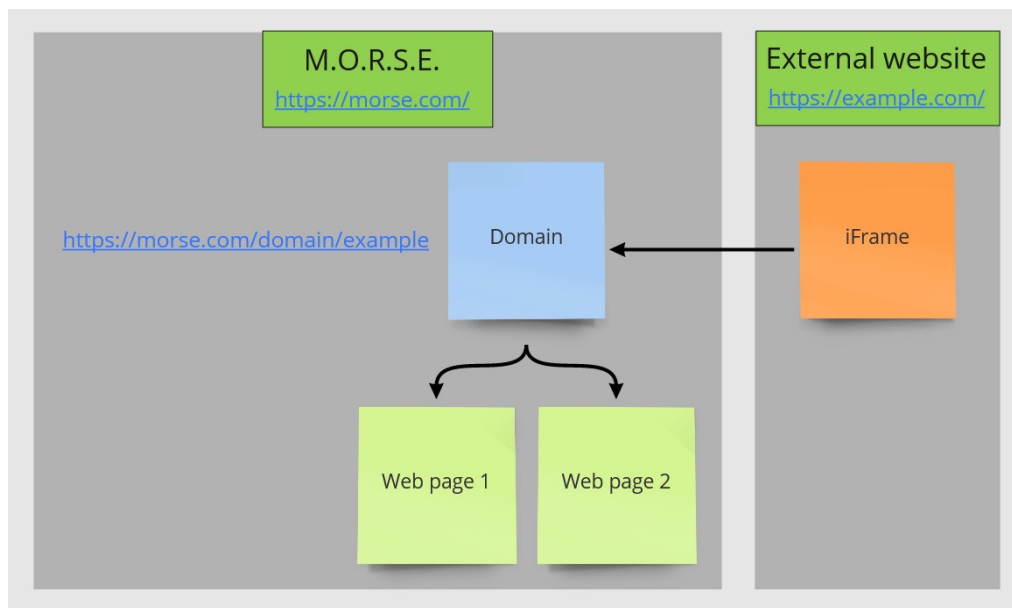


Figure 3.6: How an external website is connected to a domain on M.O.R.S.E.

second option is to inject the script at runtime when the web page is served to a player that is visiting the web page. We decided to go for the latter option, because the submission of puzzle requires some information which is not known yet in the page builder, for example the id of the player that want to submit the answer. When the script is injected the moment the web page is served to the player, this problem becomes trivial to solve, because all the necessary data and subscriptions to meteor are already present. With this approach, all the communication to M.O.R.S.E. on the external web pages can use the exact same functionalities as the existing internal ones.

The problem that remains with this approach is the fact that the right scripts needs to be attached to the right elements on the web page. There might be several buttons, some of which might not even be part of a puzzle at all or a different puzzle. The solution for this, is to mark the elements that are part of a puzzle with the id of that puzzle. This can be done by setting a custom attribute of the element. This way, all the right elements can be found on the web page by searching for elements that have this attribute. The exact implementation of how this is done can be found in section 4.4.2.

3.4. Hosting

To increase the level of immersion of the external web pages, they will have to be hosted online. There are two main parts of hosting a web page online. The first part is buying the domain for a website. This is something that we cannot do automatically for the employees of Raccoon games. The second is putting the website online and connecting it to M.O.R.S.E. This is the biggest and hardest part and something we can (mostly) do for employees of Raccoon games.

At the start of the project, different ideas for this part of the hosting and tracking had been researched and thought out, see appendix C.6 for the options that we considered. Ideas such as to use domain name shadowing on a DNS level or cross domain tracking for cookies were considered. Out of these ideas, the iFrame container has been chosen. To further explain, the idea is that an iFrame will be placed on the external website. This iFrame shows a web page hosted on the M.O.R.S.E. domain. Because the original web page is hosted on the M.O.R.S.E. domain, the user session and cookies remain intact and the already existing frameworks for communication with the server of M.O.R.S.E. can be re-used.

The file containing the iFrame only has to be set to the correct web page on the M.O.R.S.E. domain and then uploaded to the external website. This does require some programming experience because the 'src' attribute of the iFrame tag in the HTML file has to be modified before uploading it to the server.

Although this task does require a bit of programming experience, it also massively increases the immersion. The player of an event will have to navigate to actual external web pages and can see the real looking web page created in M.O.R.S.E. Thinking that they are on a real web page and interacting with it offers the immersion that Raccoon games is looking for.

3.5. Extensibility

To allow additions in the future to be as streamlined as possible extensible should be kept in mind throughout the design of the components. For page builder blocks, properties in the page builder and puzzles on web pages the extensibility is explained below.

In the page builder, there are blocks which can be drag-and-dropped into the web page. All these blocks are generated from JSON objects. This means that it is very easy for future developers to add new blocks to the page builder.

To allow better and more advanced modification of elements in the page builder, new properties can be added or existing properties can be modified. All the visual components in the side menus of the page builder that allow for property modification of elements in the web page are generated by a properties file in the builder library. Therefore, the user does not need to understand the Angular and Meteor based client code to simply add new properties. Of course, when adding a new and visually different property type in the page builder that needs other modification tools than the already existing ones, new client code has to be written.

If in the future new puzzle types would be added, the design of how the visual elements for puzzle types are created is important. To allow specifying these puzzle elements easily, a structure should be present that defines which elements belong to a puzzle type and what their attributes must be. Per puzzle type, a list should be defined of what HTML elements that puzzle type exists of. In addition to the elements themselves, their properties should be definable. For example the HTML tag and their type, 'input' and 'password' respectively for a simple password puzzle. This design allows to easily create new puzzles types in the future with custom HTML elements.

3.6. Backwards compatibility

The addition of the aforementioned features requires some alteration of the database and functionality of the existing system. One very important consideration in the design process of these changes is that it has to be fully backwards compatible, that is, old events that do not use this new functionality should still work in the new system. It should also be possible to add roles, domains and such to these events because the client already has a multitude of events in place in the old version of M.O.R.S.E., which they intend to migrate to the new version. In this section, we discuss how each of the new modules is designed to adhere to this requirement and in 5.1.3 we demonstrate having done so by manually testing the backward compatibility of our application. In general, it holds that features have been designed as an extension or addition to the system, rather than replacing or overwriting an old feature. As a consequence, all original functionality will still be in place and left mostly untouched.

3.6.1. Roles

As described in 3.2, the addition of roles to teams requires a couple of changes to the database and the way the current login system works. The key idea behind the design that has to make this work is that roles are simply a special type of team. Rather than implementing an entirely separate system that handles management and logging in for roles, we use the system that is already in place and extend it, such that it supports both regular teams, and roles within teams. By making all these extensions optional fields in the database and not restricting any of the program logic to roles specifically, we ensure that the regular teams still works as well. As a consequence, the original functionality remains intact, because the regular teams can still be used alongside the newly added roles.

3.6.2. Domains

In a similar fashion to how the roles are designed, the new domains are implemented alongside the pages that M.O.R.S.E. already supports. This way old events that use the M.O.R.S.E. pages can keep on using these pages and, where desirable, gradually migrate to use the new domains instead. As described in 3.1, the addition of domains to M.O.R.S.E. means new types of schedules are added to the system and although schedules are strongly intertwined with M.O.R.S.E., new schedules can be added fairly easily by inheriting from the other classes that already exist. This way the old schedules are left unaltered and it ensures that they will still function as they did before.

3.6.3. Page builder

Section 3.3 describes the design of the the page builder. In order to ensure this is fully backwards compatible with older versions of the system, this entire new module is designed to be a virtually stand-alone application in a separate tab in M.O.R.S.E., with a very limited set of connections to M.O.R.S.E. In doing so, we guarantee not only that it can function and be updated independently of the M.O.R.S.E. system, but also that, besides during the game design process, M.O.R.S.E. can function completely independently from it. This means events created without the page builder will work fine alongside the page builder since they do not have to interact with each other in the first place.

3.7. Summary

We now summarise our conceptual ideas for the project. An overview of the design alongside the existing functionality is shown in figure 3.7. For linking external websites to M.O.R.S.E. and tracking the players' progress, we introduced the concept of domains and web pages to be added as schedule items in 3.1. Then, to create more interaction with players on these sites, we adopt the idea of roles, as used in DigiHacked, to allow the game designer to incorporate teamwork in events, see 3.2. The domains then can be developed by the game designer in our page builder, proposed in 3.3, without the need for programming experience. Of course, all domains have to be hosted some way or another and we chose the approach of using an iFrame container. Since it must be easy to extend the M.O.R.S.E. system, we plan to keep all possible page builder properties and blocks separate from the more complicated Meteor and Angular code as much as possible, as described in 3.5. At the end of this project, all the already existing functionality should still be working. In 3.6, we explain our strategy for making sure everything stays intact. By keeping the page builder component separate from the rest of the system and building the functionality of the roles around the current code base, rather than trying to weave it in between, we ensure our product stays backward compatible.

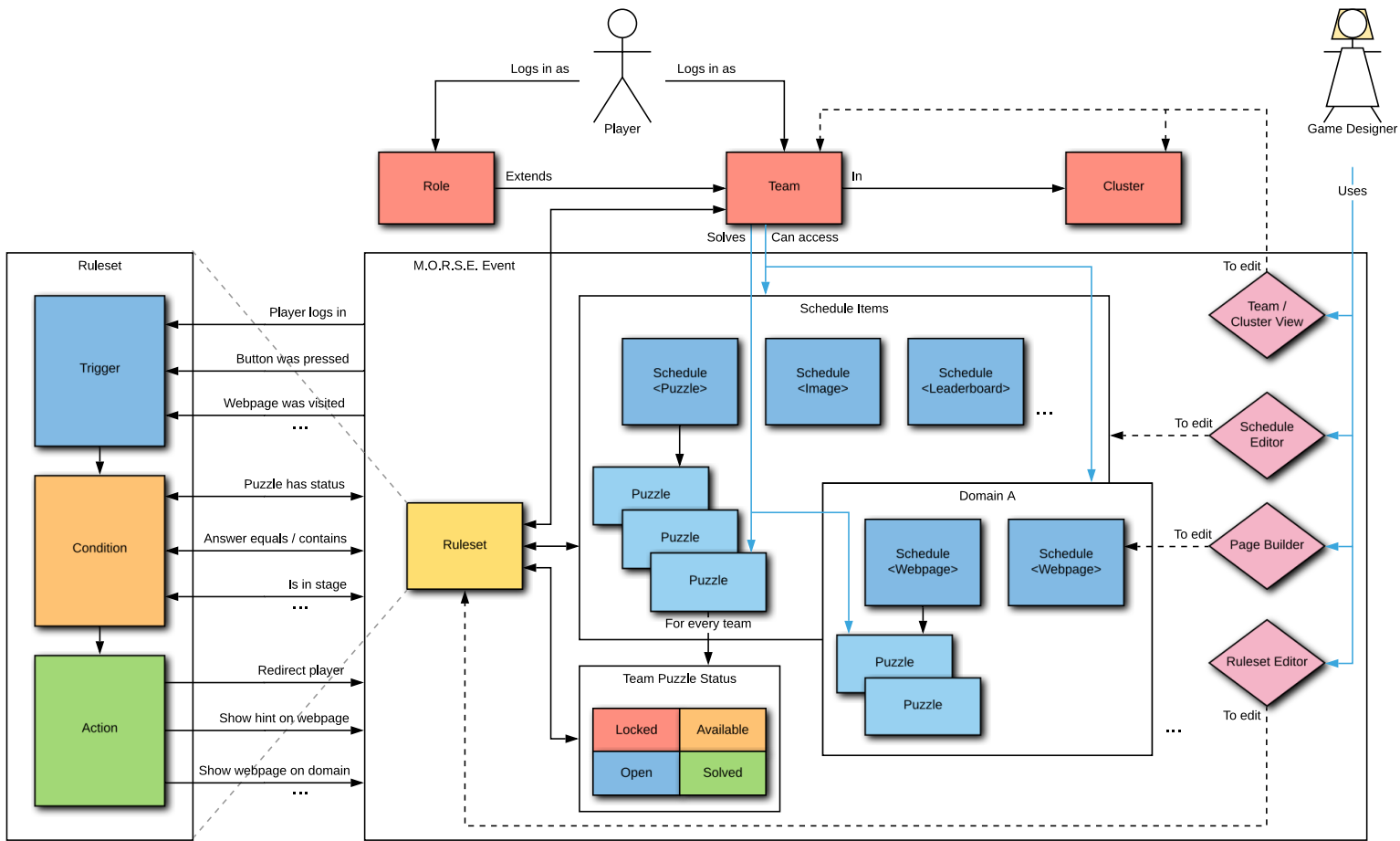


Figure 3.7: An overview of the updated design of M.O.R.S.E. with domains, web pages and roles

4

Implementation

In the previous chapter, we gave our conceptual plans and ideas for how we wanted to solve the questions stated at the start of the report. For this chapter, we give a more in-depth explanation of how we incorporated those concepts into the M.O.R.S.E. system.

In 4.1, we explain the code structure of the current implementation along with the frameworks that were already in place. Then 4.2 will give the implementation details of the domains as a solution for the first sub-question, namely accessing and interacting with different front-ends. Section 4.3 does the same for the second sub-question, which was about the cooperation within teams and adding competitive elements, where among others roles for participating players was offered as an answer. The technical details for the solution for the third and final sub-question about how one should be able to construct and customise the different front-ends are given in 4.4, which is about the proposed page builder.

4.1. Current implementation

To provide some context for the following sections, this section is dedicated to sketching an overview of the current implementation of M.O.R.S.E. Since section 2.2.2 already discusses the general structure and components of M.O.R.S.E., we only discuss aspects of M.O.R.S.E. specifically related to its technical implementation. While many of our choices regarding design and implementation are largely based on their compatibility with the system that is already in place, the purpose of this section is only to provide context for the subsequent ones, we limit the scope to implementation details relevant to the features we have added, rather than providing a full in-depth analysis of the current implementation. For a more thorough discussion of the parts left undiscussed in this report, we refer to the corresponding bachelor thesis for the initial design of M.O.R.S.E. 2.0 [4]. In this section, we discuss the general architecture of the system we have extended, as well as the frameworks used and a brief analysis of the structure and implementation of some of the components that are most relevant to the scope of this project.

4.1.1. Architecture

First of all, we discuss the architecture of the system that is already in place. M.O.R.S.E. uses a client/server model, where the clients are single-page web apps that run a web browser. Multiple different web apps exist for different kinds of end-users. For example, there is a separate player app and an admin app. Regardless of their respective end-users, these web apps communicate with the server about changes in the data and about actions executed on that data. The client keeps a local copy of a part of the database that is relevant for them, which is then updated whenever a change occurs in the original database. It is important to note though, that the client is not allowed to alter the database directly. All changes to the data have to be made by sending a request to the server, which then updates the database and sends a response, possibly accompanied by the changes made to the database. All code of the program has been divided into three large sections. A part for the server, one for the client and a shared part called the imports, that is used by both. The latter ensures that the client and server are both working with the same data types and functions to act upon this data.

4.1.2. Frameworks

M.O.R.S.E. is an application built using TypeScript¹ and uses several frameworks for the management and rendering of data. We briefly describe the frameworks used, since we ended up using them as well.

Meteor The server uses the Meteor.js² framework, which is responsible for the interaction with the database and providing a layer for communication between the server and the client. Meteor provides so-called server methods that can be called by the client, which are hooks for different pieces of server code that the client needs to interact with the database. Because not all data should be publically available, Meteor offers a subscription-based model for distributing the data to the client. The client can subscribe to one or more publications of the Meteor server, and these subscriptions directly update the clients' local copy of the database. This way, the clients' access to the data can be fully controlled by the publications that offer the data to the clients.

MongoDB, Minimongo and Astronomy.js MongoDB³ is used as a database, which has a pretty big impact on the overall design of the data structures because Mongo uses a document-based model, rather than a relational data model. As a consequence, related objects are often stored as nested documents, rather than separately. In order to convert these documents back into objects that can be used in TypeScript, Astronomy.js⁴ is used, which provides a document-to-object mapping for MongoDB. This means documents can be interacted with as if they are regular Typescript objects. It was already mentioned that the client stores a local copy of a small part of the database. This is done using the Minimongo⁵ library.

Angular Since the client serves as a view for the data supplied by the servers publications, this view has to be updated reactively. That is, whenever a relevant part of the database is updated, the view of the client should be updated as well. To this end, Angular.js⁶ is used. Angular is a front-end framework that uses components which can be nested into each other, which greatly enhances the modularity of the application while providing methods to easily make the client fully reactive.

4.1.3. Components

Many of the features we have added are extensions of components that already exist in M.O.R.S.E. In order to provide a clear context for the following sections that discuss these new features, we will briefly describe the most relevant components in the system. For a more in-depth description of these components, we again refer the original M.O.R.S.E. bachelor thesis [4]. Sections that build forth upon an existing component will also discuss the relevant aspects and attributes of these components, so the description below is meant solely to provide the reader with some perspective regarding the place and purpose of these components within M.O.R.S.E.

Users All players and admins in M.O.R.S.E. have a corresponding User object, where their credentials, state and progress are stored. All information regarding gameplay is stored in a nested object called a UserProfile.

Puzzles Puzzles are arguably the most important components in M.O.R.S.E., because they facilitate the interaction with the player. A puzzle object stores information about what the question is and what the possible answers are. Different kinds of puzzles exist, such as open questions and multiple-choice questions. We have also added two new puzzle types: form puzzles, which are open questions with several inputs, and page visit puzzles, which (as their name implies) require the player to visit a page in order to solve the puzzle. Both of these new puzzle types are discussed in section 4.2.4.

Rulesets Rulesets are used to define automated events within M.O.R.S.E. that can be triggered by, for example, a player solving a puzzle. A rule in the ruleset consist out of three different elements: triggers, conditions and actions. The triggers of a rule determine when the rule should execute, whereas the conditions

¹<https://www.typescriptlang.org/>

²<https://www.meteor.com/>

³<https://www.mongodb.com/>

⁴<https://atmospherejs.com/jagi/astronomy>

⁵<https://www.npmjs.com/package/minimongo>

⁶<https://angularjs.org/>

represent extra constraints that have to hold in order for the attached actions to be executed. The conditions are optional. We have added both triggers and actions to M.O.R.S.E., which have all been documented in the following sections. Most of them have to do with navigation between, and interaction with web pages.

Schedules All gameplay in M.O.R.S.E. is part of a schedule item. These schedule items dictate the general flow throughout the game. Rulesets can be used to move a player from one domain to another. The schedule item where a player is in determines which of the aforementioned gameplay elements they gain access to. We have added several schedule items to M.O.R.S.E. including domains and web pages, which are discussed in the next section as well as section 4.3.2.

4.2. Domains

Inline with the design of the domains proposed in section 3.1.1, inheritance is used for the new schedule types. The domain schedule items are inherited from the schedule group item. This existing schedule type could be used to group together other schedule items. A restriction was added such that only web pages could be added to the domain schedule instead of all schedule types. Just like schedule groups are the base for domain schedules, the puzzle schedules were used as a base for the web page schedule. Because the puzzle schedules are also a collection of puzzles, most code could then be used by the web page schedule as well. However, the existing type-checking was done using a type field, which would not work nicely with the inheritance we added. Therefore we updated some of the type checking to use the TypeScript keyword *instanceof*. A sample conversion can be found below to illustrate this further:

```
// Old snippet:  
schedule.type === 'ScheduleGroup'  
// Updated snippet:  
schedule instanceof ScheduleGroup
```

By inheriting from the existing schedule item types, the schedules could be stored alongside the existing schedules in the database. Since domain schedules are more complex than the schedule groups, extra fields were added for the domain name and whether the domains was external or not. As discussed in section 3.1 external refers to whether the existing M.O.R.S.E. layout will surround that domain. For the implementation, this field determines which domain component is rendered. The internal domain component extends the external one to minimize code duplication since the functionality is practically equal, but only the layout is different. Additionally, a field for the role access was added to allow for access control. More on this topic can be found in section 4.3.

4.2.1. Web pages

The web page schedules are a bit more complex than their puzzle schedule counterpart. Just as the puzzle schedules, the web page schedules store the contained puzzle. However, a new link was added to the visual representations of the pages. This page collection is a separate collection in the database. Although currently, the mapping of web page schedules to the pages is one to one, the decision to create a separate collection was deliberate. First, this separates the visuals from the logic behind it. Rarely both the visual elements of a page are required as well as the schedule logic behind it. For example, when editing the schedule, the content of the page is irrelevant. Vice-versa this holds for editing the content of the web page using the page builder. In addition to compartmentalising between the content and the logic, it could allow future extensions of the system to link multiple schedules to the same web page.

4.2.2. Users

To be able to store which teams or users are on what web pages, the user's profile was used. This was done to keep it consistent with the existing storage of what the current schedule item for each team is. As explained earlier in section 3.1, the system should allow users to access multiple domains at the same time. Therefore, the user's profile stores the current web page for each domain for that team. To move a team on a domain from one web page to another, the corresponding domain entry in the user's profile has to be updated to reflect the new web page. Changing this can be done using a new action, which is explained in more detail in section 4.2.3. This implementation also saves the progress on a certain domain. Therefore users can close their browser without losing any of their progress. When a new team is created, they are automatically assigned the first web page of every domain schedule. This reduces the number of rulesets needed and improves

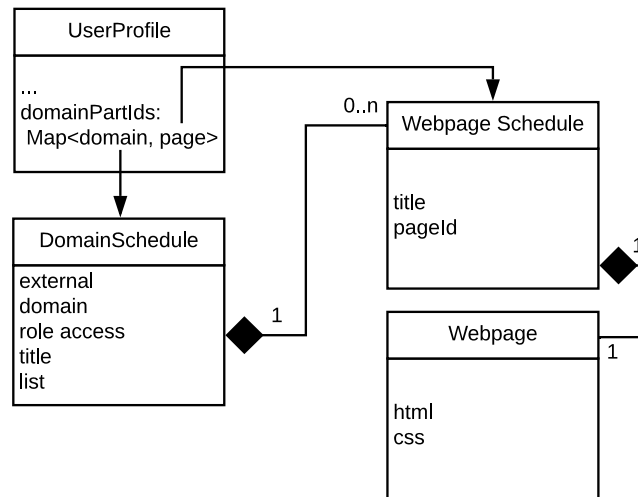


Figure 4.1: An entity relation diagram showing how domains, web pages and user profiles relate to each other

usability for the game designer. For an overview of the relations of users, domains and schedule, see figure 4.1

4.2.3. Additions to rulesets and schedules

Now that we have arranged for the hosting of domains and web pages, we also want to make them interactive and track the interaction the players have with the web page. This is vital to provide an answer to our first sub-question asked in chapter 2. The following paragraph explains what has been added in order to provide different kinds of interactions with web pages.

Page visit trigger It can be useful to know when a certain team has visited a certain web page, but this would be difficult to find out with using just the M.O.R.S.E. system. Especially if you want to perform other actions when someone visits a certain web page, additional solutions would have to be programmed. So, for this specific problem we created the PageVisit trigger. As the name says, this is a trigger that goes off when a certain web page is visited.

Button trigger Buttons can be used for various purposes on domains. Often you want certain actions to be executed when a button is pressed, for which you require a certain function to be executed. To greatly increase the possibilities of the M.O.R.S.E. system, we added a button trigger, which skips the phase of having to write a function that has to get executed, and causes certain actions defined within the M.O.R.S.E. system to be done the moment a certain button is pressed. Not only does this allow for more interaction with the current web page, but to also easier interaction with the M.O.R.S.E. system.

Redirect action As discussed in section 4.2, domains can contain several web pages. We ideally want to be able to switch between them. For this cause, we implemented a special action that can be called by a trigger, the redirect action. This prevents that the game designer has to arrange all these redirects by hand. By specifying a domain and a web page, the action sets the shown page of a domain to the specified page. If you have the domain opened, the content of the web page will switch towards the specified page. If not, then the next time you open the domain, then you will see that web page (unless it has been changed again by another trigger in the ruleset).

Page hint action When a team is stuck on a certain puzzle, the old M.O.R.S.E. system could send a (possibly predefined) hint to help that team on their way. This is still possible, but with the new adjustments, it is very well possible for a team to get stuck on a puzzle not placed in M.O.R.S.E. but on a web page. If the administrators then send a hint, the players might not see it, since it would only show in M.O.R.S.E. and not on the web page they probably have opened at the time. Therefore, a ruleset action was required that would show a hint to a team while on a domain. This is now present on the system, in such a way that you get a similar popup as others, but a page hint does not disappear on its own. It waits until the player clicks it away,

thereby confirming that they have seen and hopefully read the hint. If they accidentally click it away without reading, they can approach the staff present at the event who then can easily re-execute the action so the players get shown the same hint again.

4.2.4. Puzzles

The immersive aspect of an online escape experience is mainly caused by puzzles placed on web pages in such a way that the player is solving the puzzles, without them having the feeling that the puzzle is part of M.O.R.S.E. but part of the site. This contributes to the puzzle seeming like a real problem.

Since domains and web pages weren't part of the M.O.R.S.E. system before either, implicitly puzzles on such web pages weren't present either. In the page builder explained in 4.4, you can insert various input elements who can be linked to puzzles created in the schedule editor from the M.O.R.S.E. system. If the player gives some input to these elements, the system submits them as answers using the regular workflow of questions within the M.O.R.S.E. system. We made sure the existing puzzle types could be used, but we also added some new puzzle types of our own.

Form puzzle Even though we can now put puzzles on web pages, there was still an important action often found on web pages that could still not be added intuitively, namely logging in. A log-in screen could technically be constructed using two separate text input puzzles, but then you would have to submit these puzzles separately, while as with real login systems, it gets submitted as a whole. You want two answers getting verified at the same time.

To not only allow for login puzzles, but also for puzzles with more than 2 input fields at once, we have implemented the form puzzle. This puzzle allows exactly what is described for a login system if you were to create a form with only 2 input fields. For more complex puzzles, you create a form with more input fields, allowing the game designer to increase the difficulty of his online escape experience as he wishes. To allow for even more flexibility with form puzzles, the input type for each of the input fields can be changed. The usual text, password and number fields can be added for some basic forms. These are however just a small subset of the available inputs in HTML⁷. In M.O.R.S.E. we also support more uncommon types such as colour, date, time, and week.

Page visit puzzle As stated in the problem description, it was hard to keep track of the progress of all the teams regarding what sites they have found and visited. Since you are not doing something active, but something passive like visiting the page, it is very hard for the project staff to see what your progress is within the event regarding the domains. It is quite cumbersome to program something that keeps you updated whenever a domain is visited.

Mainly for creating such awareness of progress without having to rely on programming experience, we added the page visit puzzle. This puzzle can only be solved by visiting a specified URL which would be the 'answer' of this puzzle. It is not necessary a puzzle that a player conscientiously solves, even though you can use it as such. The main addition is that the administrator and game hosts can see what web page has been visited by which team. With this, the earlier stated problem is solved, since not only does the project staff know what knowledge is available to each team, but also for how long this knowledge has been available. This is very useful information for deciding if a certain team needs a hint.

4.3. Player interaction

In section 3.2, we explained the conceptual design and motivation for elements for extra player interaction. We first explain how we wove the roles functionality around the existing system. Then we detail how we have implemented an additional, more personally customisable leaderboard.

4.3.1. Roles

In this section, we describe both the data structure and component interaction of the roles module in the application. In 3.2, we described how the required functionality of the roles can roughly be divided into two

⁷<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>

parts: the management of roles (the game designers perspective) and the actual usage of those roles (the players perspective). We will adhere to the same structure in this section since these parts can almost be perceived as independent. Up until this point, the term 'role' has been used to both refer to a type of role in general, as well as a specific instance of such a role type. Since this section contains a more detailed description of these roles, we will strictly distinguish between role types and role instances, from here on out. In the first half of this section, we provide an overview of the data and interactions in the management part of the roles, where we have look at how role types are created, deleted and assigned permissions. In the second half, we discuss how players use instances of these roles to login to M.O.R.S.E. to access and interact with resources such as web pages and puzzles.

Role management: data

The game designer is responsible for creating role types in an event and assigning them the proper permissions, such that each player can only access a subset of the information required to solve a puzzle. Most functionalities added to support this are located in the admin module of the client. More specifically, in this admin view, we only add visual elements to the 'Teams / Clusters' tab and the 'Schedules' tab of M.O.R.S.E. The addition of roles to M.O.R.S.E. required changes to be made to three of the major data structures in M.O.R.S.E.: the Event, which stores general information about the configuration of the event; the User, which stores all account-related data of a player; and the ScheduleDomain, which was added as a part of this project to facilitate extra front-ends to M.O.R.S.E. Please note that all of those changes are additions of extra optional fields, rather than actual changes to existing fields. Therefore, these data structure will still behave correctly when used in the contexts they were originally intended for. The data structures are all implemented as Astronomy.js classes, each stored in their own MongoDB collection. Each of the aforementioned classes is discussed below.

- The Event structure only required a small change, because the only thing that needs to be added for the desired functionality is a list of all available role types within a system. These types are stored as GameRoleType objects in an array inside the attribute availableGameRoleTypes in an Event. Such a GameRoleType only contains a string representing the name of the role. One could argue that the entire object might then be just as well replaced with a single string, but we chose not to do this because an object provides much for easier extendability in the future.
- The data structure of User is probably the most significant decision to be made in the implementation of roles because it directly affects how users will interact with them. Several alternative structures were considered before deciding to use the User class that already exists and extend upon it, rather than creating an entirely new class for the new functionality. At first, we were quite hesitant to incorporate the roles into the User class, because this meant we would have to be very careful not to break any of the already existing functionality. A huge drawback of implementing an entirely new class, however, would be that there would be a lot of code duplication since most of the methods for managing user accounts and logging in to them from as a player were already in place. Because of this, we ended up implementing the roles as a specific type of User referred to as a 'GameRoleInstance', while changing as little as possible about the way Users behave. By adding fields to the User object that store their role type, as well as one for storing the team they belong to, we did not have to change any of the existing attributes of the User class. The fields for role type and parent team are respectively named 'gameRoleType' and 'parentTeamId' in the User class. Both fields are strings, and optional, such that they can be left empty if a User is not a role instance. Each User object also has a nested UserProfile object and, therefore, a role instance will also require one. The second half of this section describes how such UserProfiles are created for- and synchronized between role instances and teams.
- The ScheduleDomain class was added to M.O.R.S.E. as part of this project to facilitate creating managing groups of web pages in the schedule editor of M.O.R.S.E. This class was only assigned one extra field, called the rolesWhitelist, storing a whitelist of role types for that domain. A user should only be granted access to this domain if their role type is included in this whitelist. It would have been possible to add such whitelists to other schedule items in M.O.R.S.E. as well, but this would have been much harder since there is an important difference between how the progress is stored in regular schedules and in domain schedules. A user is at all times considered to be inside a domain schedule, whereas they can only be in one 'regular' schedule at a time. As a consequence, if a role is denied access to a schedule item that their team is in, they can't do anything until their team progresses to the next stage.

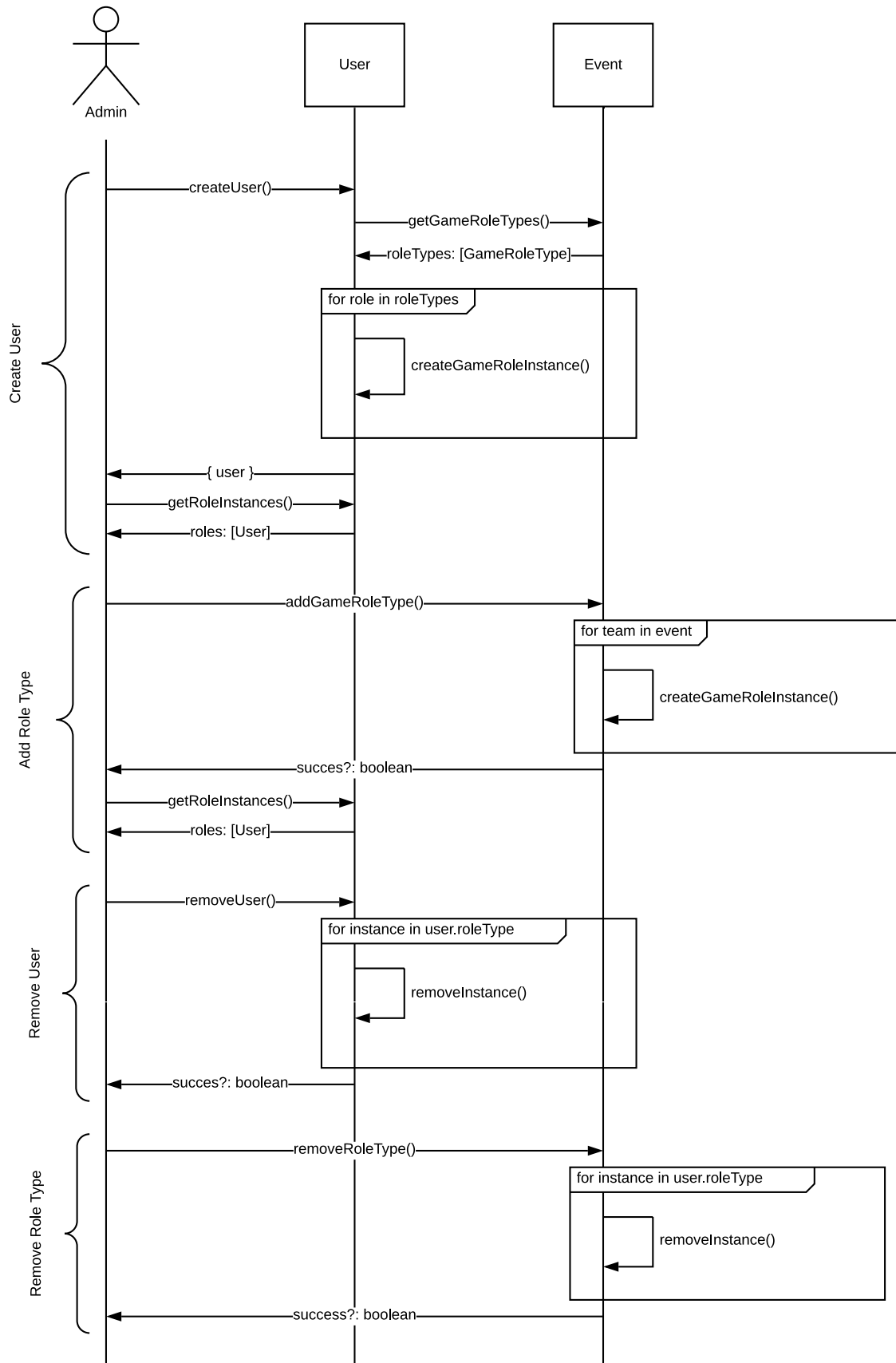


Figure 4.2: An UML sequence diagram showing how game designers can create role types and instances, and how this affects other components in the system.

Role management: functionality

Having extended our classes in such a way that they can store all information required for the new functionality, we are able to implement the interaction between these classes. From the game manager's perspective, such interaction should include the management of role types, instances of those roles and their permissions. We discuss each separately below.

In order to be able to add role types to an event, we have added a form on the 'Teams / Clusters' tab of M.O.R.S.E. where the game designer can add a role type by typing a name and pressing return, and remove roles by clicking the 'x' button on the corresponding tile or pressing the backspace key. Role types are displayed as a list of rounded badges, which have been implemented using the Material Chip components of the Angular-material library. Since this only updates the list of types in the front-end and loses track of all types as soon as the page is refreshed, it has to be connected to the database as well. One of the design principles of M.O.R.S.E. is that database entries can not be changed from the client. This means that, in order to add the types to the Event in the database, Meteor methods for getting, adding and removing role types are required.

Having implemented the management of role types, specific instances of the role can still not be used, because they require entries to be made in the User collection as well. Using the createUser method that was already in place, new accounts are created for each role type whenever a team is created. And similarly, role instances will also be created for every team if a new type of role is added. If a role type or user is removed, their onRemove event handler in the Astronomy.js class is called, in which the corresponding instances are also removed. This process of creating and removing role types and instances is shown in figure 4.2.

After having defined the types of roles that exist within an event, they can be added to whitelists in the domain schedules. For this, the same type of material chips are used as for the management of role types, but this functionality is located in the 'Schedule' page of M.O.R.S.E. When these chips are added or removed, they are also updated accordingly using the 'Schedule.update' Meteor method that was already in place for the original schedules. How these whitelists are used to grant or deny a player access to a web page is described in the remainder of this section.

Role accounts: data

An important aspect of the gameplay experience of players is the fact that they can progress throughout the game and reach new puzzles and challenges. This requires the progress to be stored in the backend as well. In M.O.R.S.E. the progress of a player is stored in their UserProfile object, and their answers to puzzles and whether they have been solved are stored in an object called a TeamPuzzleStatus. Below we discuss how both are shared or synchronized to allow players to work and progress through the game together. The data structure of these TeamPuzzleStatuses and UserProfiles is also shown in figure 4.3.

- A TeamPuzzleStatus is an independent object in the database. Consequently, these objects can easily be shared between teams and their role instances, because they can use references to the same objects. In doing so, a team and its corresponding role instances use the same object to store their progress for each puzzle and, therefore, automatically progress together as well.
- Earlier in this section, we already discussed the changes made to the User class. We also mentioned that each account has its own nested UserProfile attached. Since these are directly part of the User class, rather than being referred to, it is not possible to let role instances and teams share these profiles. Rather than extracting these UserProfiles from the User class and introducing a new level of indirection between the two, we chose to synchronize the profiles of role instances and users. Whenever a profile of a role instance is updated, it sends the changes to the corresponding team's profile as

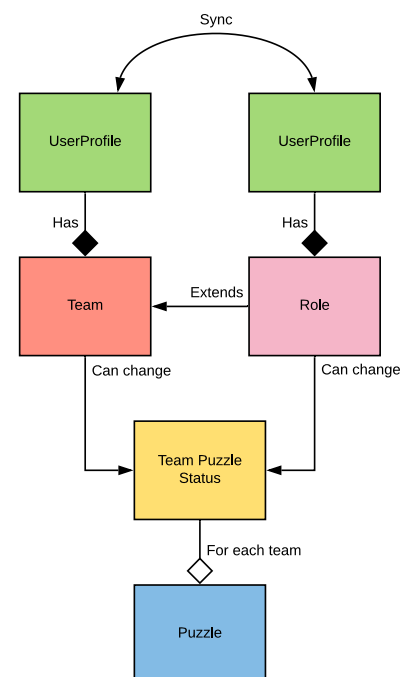


Figure 4.3: A UML diagram showing how the User and Role class relate to the TeamPuzzleStatus and UserProfile

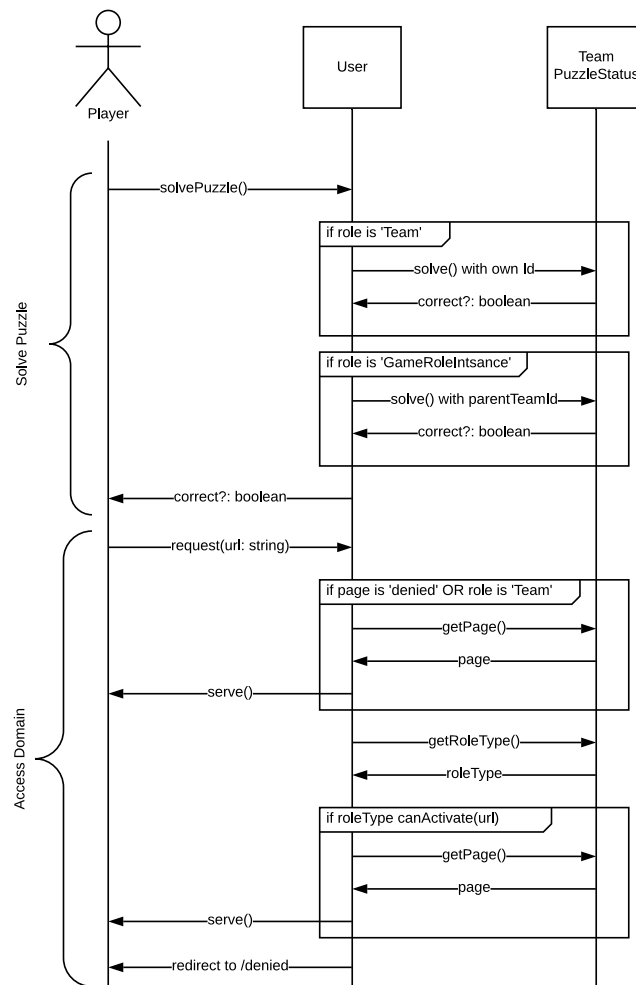


Figure 4.4: A UML sequence diagram describing how users can submit puzzles regardless of whether they are a team or a role instance.

well. Similarly, if a team's profile is updated, the changes are also forwarded to all corresponding roles.

Role accounts: interaction

Finally, we discuss the interaction between a role instance and its corresponding profile and puzzle statuses and how these have been implemented. When a player visits a domain, the Angular router ensures that they are redirected to the correct web page. In order to ensure that role instances can only access the domains their roles are whitelisted for, a router guard has been added to the '/domain' route in the Angular router. This router guard will decide whether a user should have access to that route every time the user tries to access the site. If a team tries to access the domain though, the router guard will always return true, to ensure teams always have access to all domains.

When a player attempts to submit an answer to a puzzle, an extra check is executed, which identifies whether the player is a team or a role instance. If the player is logged in as a team, the puzzle will be submitted as normal, but if the player is a role instance the puzzle is submitted using the instances parentTeamId. This process is shown in figure 4.4.

One of the main aspects of M.O.R.S.E. is its functionality regarding rulesets and schedules. These should also work in combination with role instances. Luckily, these all use the UserProfile to interact with the player and since we have already synchronized those within a team, rulesets and schedules will automatically work too.

4.3.2. Leaderboard stage

Here we introduce the leaderboard stage, which can be added in the schedule within M.O.R.S.E. Along with the showcase of the timer and leaderboard, the game designer can also choose a custom message to be displayed on this stage. In contrary to the projector, this message is displayed simultaneously with the other elements. This is possible since this stage is only displayed on the screen of the user, meaning the text is allowed to be a much smaller size, while still being readable. On the projector, it is projected on the wall of the location of a certain escape event. Then the text needs to be big enough that everyone in the room can read it, therefore not allowing enough room to keep displaying the leaderboard and timer.

The game designer can create multiple leaderboard stages, who all have their own custom text. This allows the game designer to redirect different users to different stages, meaning they get shown different messages (think of a leaderboard with the text 'You did it!' and another one with the text 'Better luck next time!').

4.4. Page builder

The goal of the page builder is to be able to design professional-looking web pages that can be used by M.O.R.S.E. This should be possible without needing any programming experience. The page builder does not create the web pages, this happens in the schedule. After the web pages are created, they can be opened in the page builder. Each web page starts out empty without any elements, even though the page schedule item might contain puzzles. Once a web page is opened in the page builder, new elements can be drag-and-dropped into the web page and the style of each element can be changed. The puzzles components can either be drag-and-dropped or linked to existing elements on the web page.

The structure of the page builder is separated into two parts. The first part is the visual representation of the page builder for the user. This contains the menus and the canvas where the user has to design the web pages. This visual part is integrated into the current M.O.R.S.E. system with Angular. This part will be further denoted as the page builder tab. The second part is the logic behind the page builder, from now on denoted as the builder library. This builder library modifies the canvas based on the interaction of the user with the page builder. The division between the two is made to clearly separate code regarding logic and page builder visuals. Because of this, the code is a lot more maintainable and it is easier to modify menu's of the page builder without it breaking canvas logic and the other way around. Both parts reside in the client-side of the M.O.R.S.E. system. The reason for this is that the builder library is directly modifying the HTML document for visual changes during web page creation and modification by the user. The visual Angular code is client-side because the rest of the visual Angular code is also in the client-side. Therefore, letting this be generated server-side makes no sense.

4.4.1. Page builder tab

As mentioned, the page builder tab is made using Angular in the client-side of M.O.R.S.E. Since this is a feature that should only be accessible by the admins of an event, a new tab in the admin page of M.O.R.S.E. has been created. Because of this, the page builder tab still has all of the M.O.R.S.E. admin functionalities provided by Angular and Meteor⁸, as well as the M.O.R.S.E. admin menu, to quickly switch from the page builder tab to other tabs for a fast event creation workflow. The styling of the page builder has been done by using Angular Material⁹.

The structure and implementation of the page builder tab have been divided into multiple smaller components to keep a good overview and have separation of the code. The structure of this is visible in figure 4.5.

The top component, the page builder tab, has been divided into three components, namely the canvas, the side menu and the top menu. The canvas contains an iFrame in which the user can create and customise the web pages from the scheduler. Because all the logic is done in the builder library, this is a rather small component only containing the necessary code for instantiating an Angular component and some styling.

The top menu contains the buttons that call builder library actions regarding the whole web page inside the canvas. This means actions such as loading and storing a web page, clearing the page on the canvas and

⁸<https://www.meteor.com/>

⁹<https://material.angular.io/>

exporting the web page.

The side menu contains all the options for modification of parts the web page in the canvas. It allows for dragging new blocks into the page, connecting selected elements in the web page to puzzles and triggers in the M.O.R.S.E. system, modifying the styling, text and attributes of selected elements in the web page and allows for importing and exporting custom HTML, CSS and Javascript. Those features are divided into four different tabs of the side menu, namely: blocks tab, puzzle tab, properties tab and the customization tab.

The blocks tab has all the draggable web page elements available for the user. Those elements are split up

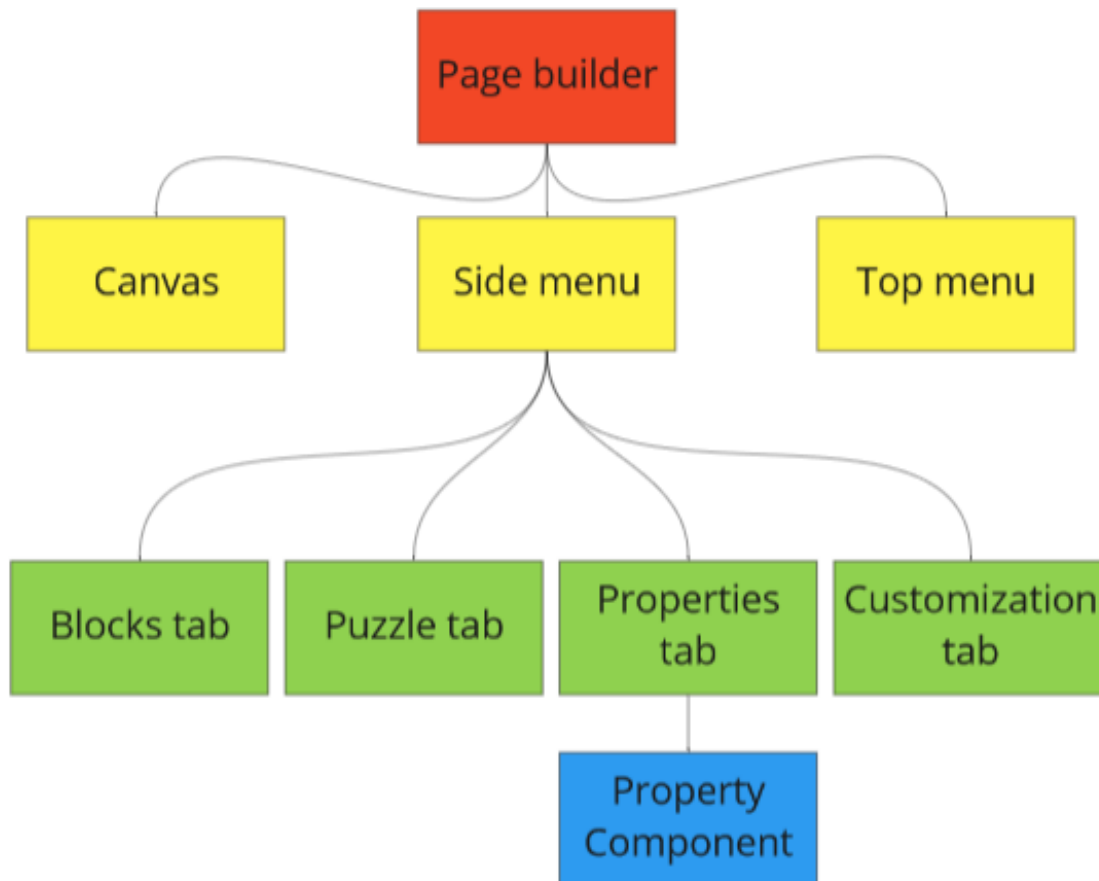


Figure 4.5: Structure of the page builder tab.

in groups to keep a better overview of the available blocks. The current groups are Layout, Basic, Inputs, Pre Built and Advanced. However, as mentioned before, the names, order and the available elements itself can easily be changed when needed. This tab also contains a custom HTML block. This is a block in which it is possible to write HTML code and drag that custom written HTML into the web page directly as an element. Because of this, the user is not limited to only the elements made available by the page builder. This does not even require real programming experience if the user were to just copy a snippet of HTML code from the internet.

The puzzle tab contains the UI for connecting elements from the web page to the puzzles and triggers in M.O.R.S.E. It is possible to connect an existing web page element or to drag and drop a new, generated, element that contains all the required inputs of for the puzzle or trigger. This generated element is then automatically linked correctly to M.O.R.S.E. In 4.4.2 we will explain how this works exactly.

The properties tab contains UI for styling the selected element on the web page. It also allows for text edit-

ing and modification of attributes of the element. The user can modify the style of the element by changing values in text inputs, sliders and colour pickers. Because of this, no programming knowledge is required to change the styling of elements. Every modification to those inputs is directly visible on the web page itself. Therefore, the users directly know what they are editing, making the utilisation of the page builder a lot more understandable. Every input that changes a certain part of the style of a selected web page element is a property component containing the type of input and its values. This is required for proper generation of the UI with the JSON object that contains all the available properties.

The customization tab contains the UI for importing HTML and CSS. This can be done by either importing a file or pasting the code into a text area. This tab is very useful for loading templates or loading existing web pages when converting a website to the M.O.R.S.E. system. This feature can also be used to transfer web pages from one event in M.O.R.S.E. to another event.

4.4.2. Connection to M.O.R.S.E.

The connection of the web pages and the M.O.R.S.E. system consist of linking callbacks to HTML elements in the web pages. This section will detail on how this is done for puzzles and triggers used on said web pages.

The web pages in M.O.R.S.E. are created in the schedule and stored in the database. Each web page has the possibility to contain puzzles which should be present on the page. This does not happen automatically, because there is no way for the program to know exactly where the puzzle elements should be in the page. The page builder can be used to open the pages and show them in the canvas after which they can be edited. The puzzles that belong to the page can either be inserted to the page, or linked afterwards to an existing element on the page. Although the puzzles have to be inserted into the page with the page builder, they are not immediately connected to the M.O.R.S.E. system yet. The actual connection of the puzzle on the page is made the moment a player actually visits the page.

Each puzzle consists of multiple components that need to be linked to an element on the page. Concretely, each answer component needs to be linked to an input field on the page, and each question needs to have a submit button which can be clicked on to submit the answer. This is accomplished by setting the 'puzzleId' attribute of the HTMLElements to the ID of the corresponding puzzle. Setting this 'puzzleId' attribute in itself does not do anything yet. It only stores the attribute value in the HTML page. When the page is fetched from the database to show to a player, it searches for all the element on the page which have the 'puzzleId' attribute. Then, an event handler is attached to the submit button which will be executed when the button is clicked by the player. The script submits the answer to M.O.R.S.E. by getting the values of the input-fields which contain the same 'puzzleId' attribute. In section 3.3.3 we describe the pros and cons of the aforementioned methods.

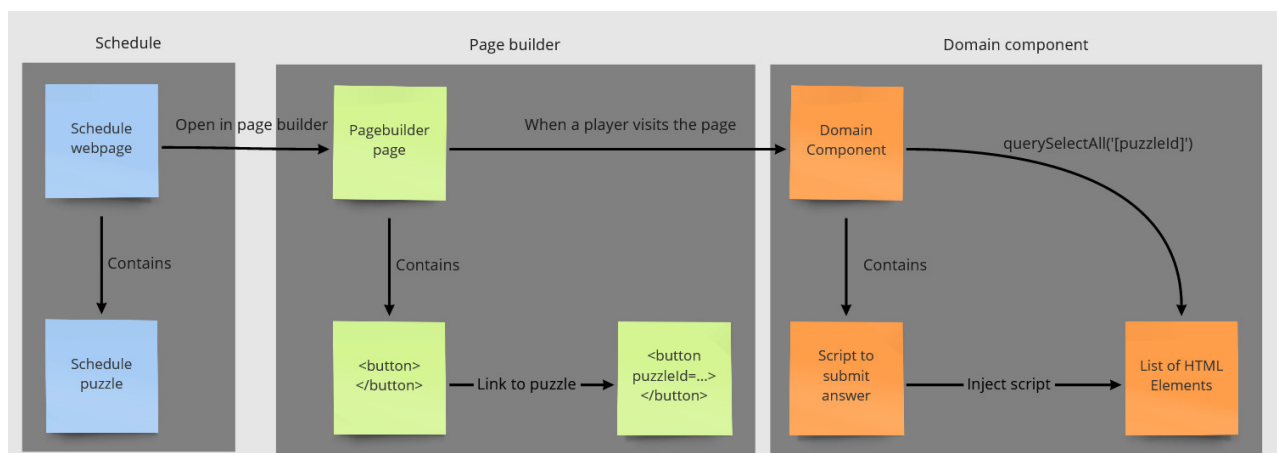


Figure 4.6: How a puzzle is connected to M.O.R.S.E.

Triggers are connected to the page in a very similar way to the puzzle. In the page builder, when an HTML element is linked to a trigger, it only sets the 'triggerButton' attribute value to the Id of the trigger. Again, setting this 'triggerButton' attribute in itself does not do anything yet. It only stores the attribute value in the

HTML page. A difference compared to puzzles, is that triggers can be linked to multiple HTML elements on the page. This means a player can possibly click on multiple elements to execute the same trigger. The main reason for this design choice is because redirect triggers are used to navigate the pages on a domain, and in a lot of real-world pages there are multiple buttons to go to another page.

4.4.3. Builder library

The builder library is where all the logic and the state of the page builder is handled. As mentioned, this is also done in the client-side. Because of this, every user has their own session, which will be reset when switching to another web page or when closing the browser. The different parts of the builder library are discussed in this section.

Data folder

The data folder in the builder library is a folder containing all the templates and the blocks and properties for which UI has to be generated. Because of this folder, adding new templates is a matter of adding a new file and including it into the list of templates. Adding new blocks and properties is as simple as copying one of the already defined blocks and modifying it to your liking.

As mentioned before, the blocks are split up in groups of similar blocks. Every block group has a name to display in the UI and an array that contains all the blocks. Every block itself also has a name to display in the UI and a variable containing the HTML code that should be put in the web page when dragging and dropping the block.

The properties are more complex than the blocks. There are multiple property types. Each property instance has different variables based on their type, such as min and max values of sliders and the options for a dropdown. Each property type has got its own class such that those variables are set correctly when the class is constructed and to keep code variables separated and exclusive for the according type. These classes all extend the property class to enforce that every property type has all the basic variables needed. All the properties that the page builder should show in the UI are exported as an array containing instances of the aforementioned property classes. These properties classes are then used in the page builder to generate the Angular code needed to display the styling options in the page builder.

Shared data

The shared data refers to an object in the builder library that contains all the variables that are visible to the outside, in this case, the angular components. It keeps track of the save state, the selected element, the selected page and event listeners needed by page builder components. The shared data uses the save state for popups if the user tries to leave the page while it still has unsaved changes. The selected element is used for rendering the properties display in the properties tab. The selected page is required for saving and loading pages, as well as showing the user which page they are currently editing. The event listeners are used such that the builder library can notify subscribed components of changes. It is currently used for displaying snackbars¹⁰, which are angular material popups, to notify the user of the effects of their actions in the page builder. Another usage is to notify the page builder that a new element is selected and that new information has to be displayed based on that newly selected element.

¹⁰<https://material.angular.io/components/snack-bar/overview>

Page manager

The page manager is responsible for all the changes that happen to the page. The reason for this is that it can easily remember all the changes that happened throughout the session. The changes to the page are encapsulated in an Action object. The Action class is an abstract class which contains an 'execute()' and an 'undo()' method. Each action inherits this Action class and must implement the two methods. For all the actions, it is crucial that the 'undo()' method does exactly the opposite of the 'execute()' method.

The actions are the only way to edit the page. This allows the page manager to undo and redo all the previous actions. The memory of the actions is cleared every time the user switches to another page, and is not restored when you go back to the old page again. This is because there is only a single instance of the action memory stored locally.

Drag and drop

The goal of the page builder is to let the user be able to create pages without requiring any programming knowledge. An intuitive way to accomplish this is by using drag-and-drop for all the element on the page. There are several important things to consider for making dragging-and-dropping feel intuitive. The first is to have some indication of where the block will go. This is done by displaying a marker on the exact location a block would go if it were to be dropped. The position of the marker is continually updated whilst dragging an element somewhere over the page. The second consideration is to make sure the blocks are inserted at a logical position. This is not trivial at all, since there is no way to be completely sure where the user wants the block to be inserted. The last thing we keep in mind, is to allow the possibility of more complex page structures such as nested elements.

The following pseudo-code describes the simplified algorithm which determines the marker position.

```

if the target does NOT have children
  => then compare the distances from the cursor to the borders of the target
    if the cursor is below a certain threshold
      => then insert the block BEFORE the target
    if the cursor is above a certain threshold
      => then insert the block AFTER the target
    otherwise
      => then APPEND the block to the target itself
if the target does have children
  => then find the closest child of the target
    if the cursor is above the closest child
      => then insert the block BEFORE the child
    otherwise
      => then insert the block AFTER the child

```

There are some additional notes about the pseudo code above. The threshold is dependent on the target label. This is the case since it makes it easier to drop blocks in an empty container element for example, whilst making it more difficult to accidentally drop a block inside a text element. For elements that can not have children, the threshold is always in the middle to prevent the user from dropping anything inside it. There is also an additional rule (not present in the simplified algorithm above) which checks if the cursor is really close to the edge. In this case, the block is inserted before or after the target instead of inside. This handles the case where the target element and its child element have no space between them and it would not be

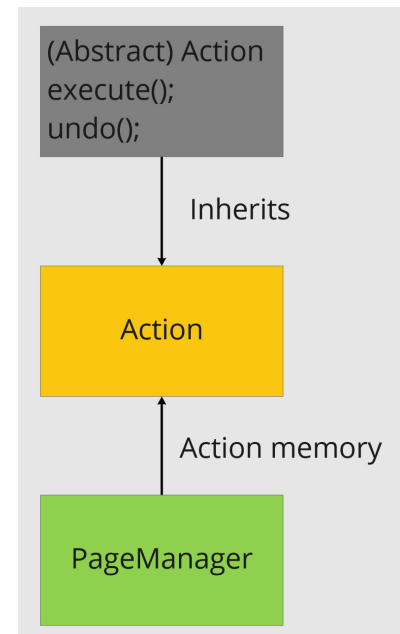


Figure 4.7: Overview of the PageManager.

possible to insert anything in between.

It is also important to note, that all the comparisons of the positions only look at the value on the y-axis. The reason for this is that all the element on the page are stored as HTML elements. The structure of HTML is hierarchical and doesn't have a single straightforward solution for encoding their position on the x-axis. There are several ways to position element left or right from other elements (floats, flexboxes, grids, etc.), and they are often not compatible with each other which makes it even harder for it to work with imported HTML code. Although implementing the ability to place elements next to each other would be possible, it would take a significant amount of time to do this and would come at the cost of working on other parts of the project. Even then, making it work with imported HTML code would be even more difficult, since the method it uses can be completely different from our solution and might not be compatible. It is still perfectly possible to position the elements in the page builder by changing the proper properties of the element, but it is not possible to do this with the drag-and-drop functionality.

InputHandler

Most of the user input is originated from the side menu, where the user can click on buttons to change the properties of the selected element on the page or start dragging new blocks into the page. However, some of the input needs to be handled by the page itself, such as handling the drop of a new block or selecting an existing element. The InputHandler module is responsible for handling these input events. The InputHandler also checks for hotkeys that are being pressed.

inputevent	action(s)
ondragover	update the position of the marker based on the position of the cursor
ondrop	insert the block in the position of the marker
onclick	select the element
onkeyup	check which key is released and act accordingly
onkeydown	check which key is pressed and act accordingly

4.5. Summary

After the design description in chapter 3, we now covered the implementation details of our product. We discussed the existing frameworks of M.O.R.S.E., such as Meteor and Angular for the front-end display and MongoDB as the database. Components of the code base were covered and explained next, being the users, puzzles, rulesets and schedules.

With the current implementation covered, we moved on to the implementation of our designed proposals. The first sub-question was how the the hosting and interacting with the websites would happen. To this end, iFrames were used for hosting sites and the domains and pages were added as schedule items using inheritance, so interactive features could be easily built. The user profile was adjusted so tracking is possible of which page each player is on. To better integrate with M.O.R.S.E., we added the page visit trigger, the button trigger, the redirect action and the page hint action, so the sites can be navigated and actions can be triggered, increasing the immersiveness of the event. Form puzzles and page visit puzzles were also introduced, to even further increase the possibilities and the immersiveness.

The second sub-question was how we could increase player interaction. We introduced roles into M.O.R.S.E. and a new leaderboard. The roles required data changes in the Event, the User, the ScheduleDomain and the TeamPuzzleStatus structure. Then we explained how the game designer could add different roles and how they can be whitelisted for certain websites. Then the player perspective was given, how players with different roles can still submit puzzles and how rulesets and schedules are not affected. Next to the roles, a new leaderboard was explained, which could be personalised for the user with a custom text.

The third and final sub-question presented the problem of producing websites without relying programming experience. For this we proposed a page builder, which we further detailed. First, the visual additions into M.O.R.S.E. were explained, where one could find the page builder and what it looks like. Then, we explained how the page builder interacts with the schedule items and the domain component, along with how a puzzle on a web page is connected to M.O.R.S.E. Finally, we explain how all the required data by the page builder is

divided in the builder library and how changes are processed by the page manager. All aspects contributing to how the solution of the stated questions is implemented.

5

Testing

An essential part of project development is testing. Not only making sure the freshly implemented features do as they are expected to, but also checking if all functionality that is already in place is still operational. They provide defect detection and reliability estimation [5]. So since a lot of code was already in place of the existing M.O.R.S.E. system at the start of the project, we acknowledged that the testing phase required a significant amount of attention and was not to be neglected. The types of testing are separated into two different aspects, namely verification testing and validation testing.

5.1. Verification

The types of tests within this verification section, are the tests that only check that the written code works as it is supposed to. All these tests require only our attendance and it is up to us to uphold a high standard of performed tests to ensure no bugs appear in the project.

5.1.1. Unit tests

Since this project is an extension of a project done by a previous group participating in the Bachelor End Project, we started our sprints with a large set of unit tests that spanned the entire code-base of M.O.R.S.E. with high coverage. At the start of the project, we set the bar high for our code development and stated that every branch that was finished, should not decrease the overall line coverage of the project. We were aware that line coverage does not mean enough on its own, but it is certainly a good start to make sure everything works as supposed at the lowest level of the application, to prevent endless debugging at the later stages of the project when more high-level functionality is in place.

5.1.2. Manual tests

The second level of making sure the product behaves the way it is supposed to do is manual tests. Of course, the developer performs many manual tests while developing new functionality, but the team members reviewing the code need to also perform manual tests on the new functionality, with the intent to let the code malfunction. These tests are a lot more high-level in comparison with unit tests, meaning that these tests not only test the written functionality but also shows if it smoothly integrates with the existing code in such a way that the product still works as it did before. Since more components are included, you can encounter faults in the code that never would have come to mind with unit testing.

Not only are manual tests useful, but in this context, they are also a necessity, due to a large amount of front-end code that was present and written additionally. Testing this code automatically is very hard, especially since no automated front-end testing was in place and all the setup still had to be done. Therefore, the manual tests were stressed to be performed frequently and extensively to make up for this absence of front-end testing.

5.1.3. Backward compatibility

While the goal of our project was to expand and upgrade the M.O.R.S.E. system, it could not be at the expense of existing possibilities. Everything possible before we started our development, should still be possible at the

end. So at the end of each sprint, when all changes of that week were reviewed and included in the product, the whole team ran the product and performed a series of various tasks to see if the product did not break by the addition of new features. This routine of tasks changed a bit each week based on what changes were made in the last sprint. If a lot of changes were made on our page builder and the domains, then we would not spend a lot of time to see if the projector with the timer and leaderboard was still operating as intended. Since the entire team had to perform a check each week, we could confidently close each sprint and start a new one without having to worry if we accidentally disabled large parts of the functionality that slipped by unnoticed.

During these weekly tests, we used the events from the databases we had locally, which did not include any existing events in the old M.O.R.S.E. At the end of the project, we also performed a backwards compatibility test using a copy of the database that was used with the old M.O.R.S.E. system. In this test, we verified that the existing events did not break. Unfortunately, this was not the case the first time we tried this, but after a quick patch, the old events worked as expected. However, due to the way access to domains is implemented, adding domains to an existing event without resetting the event does not work as expected. The old users were unable to access the domains in the system. We will look into fixing this, but with a single press on the 'reset event' button, this issue is resolved for that event, although all progress in the event is lost.

Because we kept the backwards compatibility in mind when designing the new functionality, minimal changes are needed to integrate it with the existing events. Before handing in the project to the client, we will resolve these issues to ensure a smooth transition from the old M.O.R.S.E. system to the updated system.

5.1.4. Scalability test

Something that had to be kept in mind, is that online escape experience can have a significant amount of participants. Therefore, we had to make sure that if that was the case, that M.O.R.S.E. would not be incredibly slow and that it would take forever to perform simple actions. In communication with the client, we set a baseline of 200 participants to check if the event still operated smoothly with that many players. We performed such a test once when only two weeks were remaining. It showed that there was a limit of the number of participants the server could handle and the stress you could put on it, but this limit was at a level where the baseline of 200 participants was still executable. Hence, we were not too unhappy with the result. For our detailed approach and findings, please read appendix E.

5.1.5. Static analysis

During the development of the project, we used static analysis to check and improve upon the code quality. We will discuss the two different static analyses we used.

ESLint ESLint¹ is a tool that statically analyzes code. We mainly used it to keep the formatting of the code consistent. In addition, it checks for long functions and unused variables or imports. Long functions are a sign that a function does more than one thing and, therefore, it should be refactored into separate functions. ESLint was also integrated into the GitHub pipelines to block merge request which generated ESLint errors. This setup ensured that the ESLint configuration was enforced. However, in a handful of cases, we found the errors unjustified or there was no better way of doing it, so we disabled the error for that specific case. At the start of the project, there was already an ESLint configuration present which we used as well. We quickly discovered that the configuration is not as strict as it could have been. When changing it to the default configuration, a lot more warnings were generated. Since all of these warnings were caused by the existing code, we decided to leave the old configuration as it was before. Changing this would mean fixing a lot of the existing code. This was not the goal of the project and would also cause merge conflicts with the other BEP group, team Kirby, which was also working in parallel on the same code base. We do think the system would benefit from a stricter ESLint configuration. For example, a lot of warnings come from missing parameter type checking for server methods, which could cause a lot of server errors if a malicious user would be able to call these methods with other than intended types.

SIG results The Software Improvement Group (SIG)² results show that we scored high on Volume, Unit complexity, Unit interfacing and Module coupling. However, high scores do not mean that SIG did not flag any files in which improvements can be made on those topics. For example, SIG flagged the 'BuilderLibrary.ts'

¹<https://eslint.org/>

²<https://www.softwareimprovementgroup.com/>

file since it was too high in coupling. Under normal circumstances, this might be a bad thing, but in this case, it was done on purpose. To separate the logic and the UI of the page builder, we decided to make a separate library for all the logic, which can be imported by the UI for all the logic calls. Because this library contains a lot of functions that need to store save states, selected elements, history of actions and more, it is possible to create bugs if the incorrect functions are called directly and not using the intended methods. Therefore we decided to create one file that figures as a connection point with the builder library. With this connection point, we can ensure that all the correct logic is called because one only has access to the correct functions. It is also easier in usage because one does not have to look through all the files in the builder library to find the available functions. It is now all accessible in a single file. Therefore we decided that this file was flagged incorrectly by SIG and we decided not to modify it to reduce the coupling.

The Duplication score was average with a score of 3.0 out of 5.5. When looking at the cases of duplicated code, it was almost exclusively pieces of code that were written by the previous developers of M.O.R.S.E. Other cases were files in which we added methods and functions in the same manner as previous developers were flagged as duplication as well. We decided not to modify those files, because that would mean refactoring enormous files written by previous developers and that is not the focus of this project. New features specifically requested by the client were of higher priority. If one wants to improve the code quality of the M.O.R.S.E. system itself, we suggest refactoring and removing duplicate code of such files. Only a few files that flagged on duplicate code do not fall under the previously described cases. The file in which we had a lot of code duplication was the properties file of the page builder. This file defines the settings for the CSS properties that can be changed using the page builder. For almost every property the default value and type had to be duplicated. The optional variables, based on the property type, like the format, min and max values and drop-down values had to be set in addition to a regex for reformatting the property's value. To reduce the duplication in this file, classes with default values have been made for the different types of properties. This reduced the amount of duplicate code, resulting in a final size of less than a quarter of the original size. Because of the removal of duplicate code, it is now easier to tweak property-type specific values. On top of that, the classes make the code more readable.

Unit size and component balance show very low scores. For unit size, this is mainly due to the large files that were already present in the system. However, we added quite a substantial amount of code to some of these files without splitting these components up into separate components. For example, the schedule editor component was already a massive component from the start. We added the new domain and web page schedules to it, as well as the form puzzles. We think this component should be split into multiple other components; at least a component for editing the schedules' properties and a component for editing the puzzles.

5.2. Validation

The tests in this section were performed, also to see if bugs would occur, but mainly to get feedback regarding the use of the product. It is important that we build what the client wants, not what we *think* the client wants. So in all these tests, we ask the opinion of the client and if there are adjustment that needed to be done even if certain functionality was already 'finished'.

5.2.1. Weekly client demonstrations

This is not necessarily a type of test, but since it too is a possibility for bugs to be encountered and to receive validation feedback from the client, we have included in this section as well. Both us and the client were keen on weekly demonstrations since it provided very visual progress updates for the client and we had the chance to receive additional validation feedback.

5.2.2. Playtests

After a few weeks, we got quite familiar with the product and how to operate it, along with the new features we implemented ourselves. This meant that our judgement was no longer reliable to determine if new features were user-friendly or intuitive. Fresh eyes were needed to understand the thought process of a user to whom our product is unknown. Multiple Raccoon Serious Games employees agreed to participate in play-tests with varying goals and setup. Mainly, in the beginning, participants were left to their own devices and could just explore the page builder menu, followed by simple instructions to carry out to see if they were able to easily find what they would need for these tasks. If there was confusion regarding what was supposed to be done then

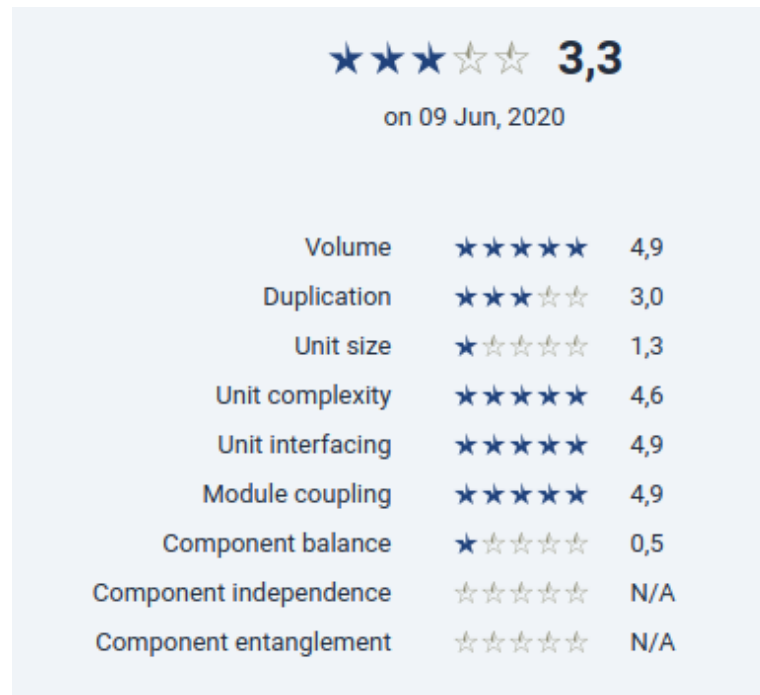


Figure 5.1: SIG results overview

we would provide tips and explanation to clarify certain matters. Later on, as more functionality and previous feedback were processed in the product, these play-tests were performed with more difficult tasks, while also reducing the amount of help given by us. All the acquired feedback was added to the sprint planning, with a high priority on average. After all, the Raccoon Serious Games employees will be the ones to use M.O.R.S.E. in the future so they have a big say into what should be the final result of our product. To have a detailed look at how the play-tests were performed and what feedback was given, please have a look at appendix G and H.

5.3. Summary

After the design proposals in chapter 3 and the implementation explanations in chapter 4, we now focused on making sure verifying and validating the product.

The verifying was done with unit tests for low level verification and manual tests for higher level verification. Then a backwards compatibility test was performed to make sure all existing functionality before the project started still was operational, and a scalability test was performed to see if events could still be hosted with a lot of people participating. To verify that our code did not only work but was also properly formatted, static analysis was performed on the code base by utilising ESLint for style errors and submitted our code base twice to SIG, where attributes like volume, unit complexity and coupling are measured.

To validate our product, we held weekly demonstrations for our client together with play-tests with other employees of Raccoon Serious Games that gave their ideas of what could be improved.

6

Process

We would like to reflect on the process of further developing the M.O.R.S.E. system. At the start of the project, we constructed a project plan that showed our intentions for the workflow and planning for the project. To view the entirety of this project plan, please have a further look at Appendix A. We will detail the general intentions and the workflow, but also setbacks we encountered.

6.1. General intentions

All team members were given a set of roles at the start, which represented certain responsibilities. This did not influence the workload someone was given each week relating the product, but it did on deciding who was going to do the miscellaneous tasks that appeared along the way. If for example employees of Raccoon Serious Games needed to be contacted, the external communicator was the one responsible for this task. This saved a lot of time discussing who was going to do these tasks every time they were required. The majority of the miscellaneous tasks were covered by the role assignment and that saved a lot of time.

To have an overview of our deadlines, a timeline was set in place with all official deadlines, but also checkpoints for ourselves regarding the process of our requirements, which can be seen in detail in Appendix B. The main checkpoint is our deadline for all the must-have requirements, which represent all functionality that is needed to meet the goal of the project, namely creating online escape experiences similar to DigiHacked within M.O.R.S.E. without programming experience. We roughly made that deadline, with some must-have requirements receiving their final touches in the week after the checkpoint, but in compensation to that were several should-have requirements that had already been finished or were in progress at that time.

We made a planning for the entire project regarding all the meetings that had to be done on a regular basis. These meetings included those with our client, TU Delft coach and team Kirby who was co-adjusting M.O.R.S.E. It also provided deadlines for the documents to be made for our SCRUM sprint planning, on which we will reflect in the next subsection. Almost all of these meetings continued as planned, except for several meetings that were scheduled on holidays. Those were all moved to one or two days later, depending on the schedule of all the attendees. There were also meetings with Raccoon Serious Games employees, which were scheduled based on the availability of the employee in question. A meeting planning had its benefits since the schedule was set for all the weeks of the project, which saved a lot of time that would have been spent actively planning each week with every party we wanted to have a meeting with. A pitfall was, however, that the non-regular meetings could more easily be forgotten if the meeting planning was being relied on too much. This almost happened to us once, resulting in us being a bit later to the meeting than usual. To prevent us from falling in such a pitfall again, we started each sprint with a moment to reiterate all meetings of that week, especially the irregular ones.

As stated earlier, we had weekly meeting with the other group who was also doing the Bachelor End Project at Raccoon Serious Games, team Kirby. Since both teams were operating on the same code base, some precautions had to be set in place in order to prevent conflicting adjustments. It would be a shame if changes from one group made the project from the other group malfunction or crash. Along with the weekly meeting,

where we discussed our weekly progress and gave an indication of which code we were adjusting, we also had a smaller weekly meeting where 1 or 2 members from each team would sit together and actually look at each others code adjustments and merge them into a co-existing project if there were conflicting changes. This way we were assured that later in the project we would not run into unexpected issues.

6.2. Workflow

An approach was made for our workflow, so our progress was easy to track for our client. This also enabled effective and efficient communication. Since this project took place during the peak of the quarantine period caused by COVID-19, we had to deal with a lot of issues which we never had to deal with before. If we were not confined by the pandemic measures, we would have been able to meet as 5 at the office, which allowed for flexible meetings if not all team members are required to be present. Also if help was required by someone on the team, all other members would be present to help instantly. So we needed to find a means of communicating that could simulate these possibilities as well. After deciding what platform to use, we'd encounter other various issues like the unstable internet connection of certain team members, but also the varying quality in microphone sound, which could both be caused by poor microphone quality but perhaps also by the increased use of the platform due to everyone around the world being forced to use communication platforms due to the pandemic constraints.

6.2.1. Engineering methodology

Due to it being already familiar to all of us, we decided to use SCRUM [6] as our software engineering methodology. Not only did we not have to figure out the details of this methodology, but it also aligned very well with the wish of our client, which was being able to see a weekly demonstration of our progress. Along with the weekly demonstrations we provided our client with our SCRUM documentation, to reflect on how each week went and what the causes were for possible delays, but also to give us an idea of how we should spend our time the next week.

To illustrate, a quarter of our tasks planned for the first week of implementation were not finished. This was because we overestimated how much we could get done in this first week of implementation, which was mostly due to the fact that we had to figure out a lot regarding the code that was already in place and the frameworks we were going to use. At the end of each sprint we made a retrospective where we discussed the reasons for certain tasks not being finished. In most cases these delays were due to the fact that we found other issues during the sprint which were given higher priority. It also occurred a couple of times that not all branches had been merged at the end of a sprint, because reviewing them and fixing bugs we found on Friday afternoon turned out to be much more time consuming than anticipated.

Extra rules were also set in place for our workflow, to make sure that possible mistakes were always reversible or easily solvable. All tasks that were finished needed to fulfil certain standards before it could be approved by the rest of the team. Proper documentation and testing should be in place, along with that no errors should be given by the style checker ESLint. These rules are very useful and prevent many style errors, resulting in a maintainable code-base.

6.2.2. Task division

In an ideal world, all team members are just as proficient with all aspects of the product. However, for a relatively short time, it would be a better to have all members work on parts of code they are familiar with. This was not always possible but making such a division of tasks prevents team members having to ask others what the idea of their code for further development was and how it is connected to other components. In general, we had a subset of the team dedicated to the page builder component of M.O.R.S.E. and another working on incorporating player interaction and website hosting. To not completely make team members unaware of how the functionality, created by the others, worked, we made sure to have everyone participate in reviewing the functionality they were not familiar with.

6.3. Setbacks

Every team is bound to encounter setbacks during a project. Ours was no different but luckily the setbacks that did occur were only minor and happened early in the project. This meant that we could communicate this

with the client and redefine the expectations of the product.

The trend with our setbacks always was always the CI of GitHub¹. In the first week of implementing the requirements, there was a lot of time necessary to get it working the way we wanted it to. Without it, we couldn't guarantee proper testing and styling of the code. It took two days for 2 people to finally get it working, which was almost half the time of one sprint for almost half the team. Needless to say, not all planned tasks were finished that sprint. Then a few weeks later, we discovered we had a limit of GitHub actions² that we could use, before we had to pay for extra. This momentarily stopped the production twice, since an alternative had to be found since we were heavily reliant on those actions. After finally running out of Github actions, we switched to CircleCI³ for the remainder of the project. However, this was not done easily. Like the previous CI, this one required a lot of manual labour to get it working, again taking two days of time from two team members.

Another smaller setback was the issues that some laptops had with hot-reloading the server. This is not an issue for the client, since running the production version worked flawless. However, if some of us made some changes to the code we wanted to try out, then once in a while the operating laptop would simply shut down. This did not cause as much of a delay as the CI-related issues, but it did cause for a fair amount of frustration.

6.4. Retrospective

Overall, we are really content with how we cooperated during the project. We feel like we succeeded in being transparent to the client and successfully adapting our plans to new wishes the client presents. This resulted in a significant amount of features being ready at the end of the project. There are always things to improve though, which also was the case with us. Since we were a group of friends already before the project started, you start the project a little more relaxed and also more talkative. So if we had a meeting with all 5 of us, it was often the case that someone had something to tell (not always project-related), that would throw everyone off-track with the matter at hand. So in the later weeks, we were getting better at dealing with this, with everyone saying when we would be distracted from the issue at hand. The topic of distraction was then to be dealt with at a later stage of the day, so not all 5 of us needed to be present. Even though we got better at dealing with this, we still were overall not as efficient with meetings as we would have liked to be. It's certainly something we would have liked to have done differently. The communication was not very straight to the point and unclear at times, which also lead to various minor misunderstandings. However, all things considered, we were quite content with that being the biggest struggle between our teamwork, since it isn't that big of a struggle at all.

¹<https://github.com/>

²<https://github.com/features/actions>

³<https://circleci.com/>

7

Discussion

7.1. Does it meet the requirements?

To validate our added functionality to M.O.R.S.E., we must check whether our changes meet the requirements we have created during the research phase. During the development, we have altered some of the requirements to match our updating view of the system as a whole. These changes were done in consultation with the client. An updated version of the requirements can be found in appendix B. In addition, we have marked the requirements that have been met with a check-mark in front of them.

7.1.1. Functional

To call the development of the product a success we must check that we at least have implemented all the must have requirements. As can be seen in the appendix all must have requirements have been implemented. We will point out two must have requirements that we decided to scratch. One requirement we had was that the game designer could change the styling of internal pages of M.O.R.S.E., however this proved to be very hard because those pages can only be edited at compile time, not at runtime. Changing it to be able to do it during runtime would be outside of the scope of this project. In addition, most of the internal pages could be recreated using our added domains and pages, therefore, we decided that this feature should not have even been a must have in the first place. The other requirement we scratched was the requirement that a web page could also be seen in a separate preview window. However, in the page builder you already see the preview of the page, so therefore this feature would not provide much added value. Furthermore, a lot of should have requirements have been implemented, in fact, all should have requirements for both the page builder and the player have been implemented. In the development process we have decided to skip the external host completely, this type of host would be able to see all the player codes and share them with the players. This role was not deemed of very high priority as it is also possible to view the player codes as an admin of the event. However it is a worthwhile feature for a future improvement, as it could greatly decrease the work for Raccoon Serious Games employees by allowing third party users to access these codes and sharing them with their group.

7.1.2. Non-functional

In addition to the functional requirements which specify what the system should be able to do, we have non-functional requirements that describe other aspects of the system. We have met every non-functional requirement except for one. We have not made all functionality that is present in DigiHacked so it can be created in our system. In addition we have not made a formal test document for the front-end which could be used to verify that the front-end functions as expected. Scalability tests have been performed to check whether the added functionality performs well under large loads. In general the system performed very similar to M.O.R.S.E. as it was used without the domains and pages, so the scalability of the system remained as it was. For more information about the scalability, see appendix E. Merging with the other group that is also working on M.O.R.S.E. has already happened multiple times, and the final merge also went smoothly.

7.2. Does it solve the problem?

Lets first restate the problem that we were trying to solve: It is very hard if not impossible to design and create online escape experiences with a wide variety of different visuals without any programming experience. We set out to solve this by extending the M.O.R.S.E. system with domains, pages and a custom page builder. Using this it is theoretically possible to create such events without any programming experience.

7.2.1. According to us

We first needed to see if we were convinced whether we provided a proper solution. However, since our team does have programming experience we should not be the ones to be the participants for playtests to judge whether this problem was solved. Therefore, we have done some larger playtest with Raccoon Serious Games employees who should be able to use the system effectively. In these tests, we noticed that they were struggling quite a bit with the existing M.O.R.S.E. functionalities such as the rulesets, this also made our added functionality harder than anticipated. During development, we assumed that the end-user would already be familiar with these rulesets, but that was not the case during these playtests. So we did not have all the conditions perfectly right for an accurate playtest, but we still received solid feedback either way. For more specific information about the playtest we have conducted, please take a look at the playtest set-up and received feedback in appendix G and H. In the end, we let someone else, someone without programming experience, but with a bit of experience of M.O.R.S.E. , recreate the existing DigiHacked. This test went a lot smoother since the little extra knowledge of M.O.R.S.E. saved a lot of extra explanation. The volunteer managed to recreate a substantial part of DigiHacked all within roughly three hours, with the main feedback being unclear functionalities that could be clarified either by adjustments in the code or in the admin manual, which we also included in appendix D.

When looking at the sub-questions stated earlier in section 2.3, there was one subjective question and two objective questions. The objective questions about the hosting of websites and tracking the progress of players, and about creating more player interaction in the product were both questions that can both be verified when using the product. We made sure to validate it with the client when creating the requirements list in appendix B, so as long as we completed the must-haves that were listed as the solution towards the objective sub-questions, then those problems behind those questions could be considered as solved. The subjective question, however, is not as easy. What might be easy to do for one, might be difficult or confusing for another, even though they both might not have programming experience. Therefore, the plan we created as a solution with the client could naturally be verified, but to label it as a proper solution, we needed it to be validated by arranging the mentioned playtests. We concluded from these playtests, that although programming is not required, having some basic knowledge of CSS and HTML is nice to have. Therefore, we think we have solved the subjective problem at hand, namely the production of web-pages without requiring programming experience, but we do allow a higher potential for people who do possess and want to utilise their knowledge of programming.

So when only looking at the questions, we provided a solid solution to all the stated problems, and in that aspect we are quite content. However, we also had requirements to look at, one of which being that the product should be able to exactly reproduce DigiHacked. This, sadly, is not possible. Our product can make a very good attempt that will very closely resemble DigiHacked, but there are a few functionalities that can not be added yet but are vital for completing the event.

The missing functionalities are the external host, storing team-specific variables on the server, being able to show those variables in the web pages in M.O.R.S.E. In addition to that, we need more options for roles, such as role-specific triggers, conditions and actions as well as the ability to hide or show elements on web pages based on the role of the player.

So when looking at the requirements, we did not deliver the product that was expected. However, DigiHacked might be slightly out of reach, but there are many possibilities for future online escape experiences that can very well be developed within our product.

7.2.2. According to the client

Near the end of the project, we asked our client to answer a few of our questions regarding our product. This was for us to not only reflect on our product, but also on us as a team and our communication and cooperation with the client.

What do you think about the final result?

The client was of the opinion that the product works good and easy. It looks good with the usability being on a good level. There was proper research towards user friendliness, which resulted in a lot of quality-of-life features present in the product, which is pleasant. The solution for the external domains and the linking of the puzzles with M.O.R.S.E. was smart.

What are you currently missing?

As we too mentioned, the client pointed out that sadly DigiHacked could not be reconstructed entirely. This means that this system can not yet be used for DigiHacked and will only be usable when a new online escape experience gets developed. The client would have liked more and better comparison testing, meaning trying to recreate DigiHacked, not only as a playtest, but for ourselves as well. Creating a list of available features of DigiHacked and a list of features that still are lacking. If there were slight differences, those had to be marked. Afterwards we argued about the importance of those differences.

How well have we processed your ideas about the product?

The client found that we listened pretty well to his ideas and wishes. The amount of testing, inquiring and experimenting was nice to see and raised trust.

7.3. Ethical implications

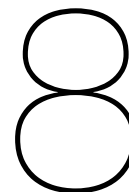
When developing software that is to be used by a large number of users that do not necessarily have experience with the program, it is also important to consider the ethical side of it. This is less of a concern if in the end the program is only used by a small set of specialised people, but in the case of M.O.R.S.E. hundreds, and probably thousands of users will participate in escape events created and hosted using M.O.R.S.E. Handling their credentials and possibly private information in an ethically profound manner is, therefore, a responsibility that should not be taken lightly. In this section, we discuss several design choices and their ethical implications.

7.3.1. Private user data

To provide a personal experience for every single player that participates in the event, data about the user has to be collected to at least some extent. This is because we have to track their progress in the game and want to identify them if they log into a web page that is part of M.O.R.S.E. In doing so, the player implicitly trusts us to handle this information carefully. Because of this, all features we added (and all features in the original version of M.O.R.S.E. as well) ensure that all information is distributed on a need-to-know basis only. That is, only admins can see login codes and progress of players that participate in the event, whereas players do not receive any information about fellow players, except for their scores on the leaderboard and the teamnames they chose for themselves.

7.3.2. URL masking

The functionality of the domains and web pages we added rely partially on the fact that the user visits a domain name and is secretly redirected to a page within M.O.R.S.E. without them knowing. In many situations, this type of construction could be perceived as deceitful, because the user is unknowingly and unwillingly shown a site from a different origin than they think, which is a trick used often in phishing and similar scams performed by individuals with malicious intents. We argue though, that in this case doing so is morally permissible because it is part of the experience that the user agreed to take part in and it is not meant for harmful practices.



Conclusion

We stated the problem at hand, which was formulated as the main question; **How can we facilitate the creation of interactive puzzles on immersive web pages in M.O.R.S.E. that have a good emphasis on cooperation where programming experience is not required?**

To get a better view of what we were dealing with, we performed an in-depth analysis in chapter 2. This analysis was about the existing M.O.R.S.E. system and an example of an online escape experience called DigiHacked. After this analysis, we were able to define a prioritised list of requirements and split the problem up in several sub-questions, namely:

- **How do we link the external websites to M.O.R.S.E. and track the players' progress across these multiple websites?**
- **How do we enforce teamwork and incorporate competitive elements in online escape experiences?**
- **How do Raccoon employees develop a customised external website, without having to rely on programming experience?**

We gave the design of how we should solve these problems in chapter 3, with the implementation of these solutions being explained in chapter 4. Then in chapter 5, we detail how we not only verified what we developed but also validated that we developed the correct additions. For chapter 6, we gave insight into our workflow and how our weekly process took shape. We also revealed the setbacks we encountered, along with the things we would have liked to have done different. After this we gave our thoughts about how the product ended up, the process we made, to what extent we succeeded in solving the problems at hand and possible ethical implications that needed to be considered. This was all done in chapter 7.

The external websites that are hosted with the use of iFrames that are linked to M.O.R.S.E. using new schedule items representing a domain and a page within such a domain. With adding new triggers, actions and puzzles it is possible to interact with these websites. The tracking of progress across websites is done with an external field in the database for the user, along with a page visit puzzle that shows the event staff when a group has visited a certain site. Those are all the problems incorporated in the first sub-question, so that can be marked as 'solved'.

The second sub-question raises the problem of how one can increase the amount of teamwork required and how a competitive setting can be created. For this we proposed the added functionality of roles, along with more interactive leaderboards. The roles cause that all members of a team are reliant on teamwork to find an answer. The leaderboards give a more personal way of motivating all participants to do their best. So both teamwork can be enforced and competitiveness is increased, meaning also the second sub-question has a proper solution.

The solution to the third and final sub-question was the page builder. Here a game designer could design

their websites and customise existing ones, without having to rely on programming experience. In comparison to the other sub-questions, this question could not be objectively checked whether we solved it or not. Therefore, we arranged play-tests for employees with no programming knowledge of Raccoon Serious Games to give their opinion regarding the ease of the page builder. With mostly positive feedback and the approval of the client, along with the page builder having enough features, the third sub-question too has its solution.

With all this in mind, we look at the main question again. The creation of puzzles is certainly possible. The page builder allows for the production of those websites and also making them immersive. The roles and leaderboards arrange the emphasis on cooperation. Since all this is either available in M.O.R.S.E. or with the use of the page builder, programming experience is not required. However, if you do possess knowledge of HTML and CSS then you are also capable to apply this in the page builder, allowing more possibilities for the creation of the websites. Knowing all this, the main question behind the problem has a solid solution, allowing us to transfer our system to Raccoon Serious Games in good conscience.

9

Recommendations

To close out the report, we discuss our recommendations for Raccoon Serious Games regarding the M.O.R.S.E. system. First, we give our thoughts about the Graphical User Interface of M.O.R.S.E. Next, recommendations about scalability, which is an important part of M.O.R.S.E., are introduced. Afterwards, new features for future improvements are proposed.

9.1. Graphical User Interface (GUI)

As experienced by us and the participants of the play-tests who have never used M.O.R.S.E. before, the GUI of M.O.R.S.E. has a very steep learning curve. Participants described it as an 'intimidating product', with which we could relate. In the beginning, we were struggling with smoothly operating the system as well. However, we do think it is a system that allows a lot as the user learns more about it. During the same period as our project, team Kirby has been working with the M.O.R.S.E. system as well, trying to flatten the learning curve of M.O.R.S.E. by adjusting the GUI.

9.2. Scalability

The scalability of a system like M.O.R.S.E. is very important. M.O.R.S.E. is designed to and should be able to host events with hundreds of participants. Because of this, our product is developed with this in mind as much as possible. However, there are still some cases in which the scalability of the product could be improved.

9.2.1. Roles

In the current M.O.R.S.E. system, adding users is very slow. With the roles feature, some extra users are needed. For example, if there are 5 roles, it means that 5 users will be represented with 1 team and 5 role users. This will effectively increase the number of user accounts in M.O.R.S.E. by 1. What happens is that a percentage of users, based on how many roles are used, is added to the users needed before. If one wants to improve the speed of adding new roles, we recommend improving the creation of new users in the system. Since the rest of the M.O.R.S.E. system's speed is not significantly affected by the number of users, roles do not change the scalability in any further sense.

9.2.2. Domains

The domains are planned to be used by a large amount of players. Because of this, the feature should also be able to handle that amount of players well. Scalability test executed, Appendix E, show that the M.O.R.S.E. server can handle 200 players without too much delay on a Quad Container ¹ on Galaxy, the hosting service of Meteor hosting ². This means the domains are decently scalable, however for a larger group of players (or a smaller Container), the current speed will not be sufficient. Pages will not load or react because the server does not have any computing power left.

There are two options for optimising the domains. First, creating a new lightweight Angular structure around

¹<https://galaxy-guide.meteor.com/containers>

²<https://www.meteor.com/hosting>

the web pages. The existing structure, which is the same as the rest of the M.O.R.S.E. app, slows down the client web page loading as well as the server. With a lightweight structure around the web page, the server should need less computing power for simply serving the page, freeing up computing power for more players.

Additionally, optimising how the server and client communicate with each other is also a solution. Currently, a lot of database calls are made frequently. Reducing bottlenecks like that will result in a much better working app, making domains more scalable.

9.2.3. Page builder

For the page builder, two main things can be done regarding scalability. The first one has to do with the number of web pages in the event. Currently, domains are stored in the database. If one were to make lots of large web pages, it would fill the database. Combining this with a lot of users participating in an event would result in a lot of big payload database requests. This would result in long loading times for users. Storing the web page as a file on the server and serving it from there reduces the latency introduced by the database calls required to load the web pages.

9.3. Future improvements

For future improvements, we recommend a few new features. Some of those features are not implemented features from the Moscow requirement list, whilst others are completely new.

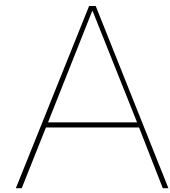
The features from the Moscow requirements list are chatbot, external host, animations, a full-screen preview of web pages and CSS class support in the page builder. The chatbot is a feature where the user can receive hints on external pages from a chatbot. This chatbot should then, of course, be configurable and manageable by the admins and hosts of the event. The external host, the same as in Digihacked, is a user who has access to all the user codes and can distribute them among all the other players. This way no admins are required to distribute the player codes or start the event. Animations, a full-screen preview of web pages and CSS class support are all features for the page builder.

Apart from the already known possible features, we came up with a few features of our own as well and discussed in the list below.

- **Assigning roles when logging in with an event code.** Currently, the admins of an event have to send all the team codes to the players of the event. It would be much less work for the admins if every player could log in with an event code and that team and role codes would be assigned automatically.
- **Role specific actions and triggers.** This would be a feature in which the game designer has more control of role-specific gameplay in the events. Think about only moving a certain role to a page, allowing only a certain role to solve a puzzle.
- **Custom hint styling.** This means that the hints currently displayed on web pages can be styled for each web page or domain. This way the event becomes more immersive and the admins have more control over how they want their web pages to look and feel.
- **New version of the leader boards for players.** Currently, there is a ranking with the 5 best performing teams regarding points gathered during the event. However, this creates an obvious winner and loser situation. This is not what the final screens of the events for which this product is created are supposed to do. It would be much nicer if the players would be shown in groups with similar results. This was suggested by the client as well.
- **Improved domains.** Currently, the player cannot use the back button when navigating through a M.O.R.S.E. created web page and cannot visit a specific page on a domain by link. If every page was also given a unique link, navigation to that would be possible outside of the iFrames encapsulating the web pages and make the back button work. Also, normal visiting by link would be possible. Another advantage of that is that instead of working with rulesets, which can be quite cumbersome for users without programming experience, you can just put a link on a button in the page builder directly. However, currently loading a web page takes a long time and shows a loading screen. This would trigger every time a player navigates on the web page. A solution for this would be either lowering the loading time significantly or smart caching such that the page would not need as many things to load when navigating.

- **Lazily loading of Angular components.** Currently, a big issue with the domains is that the loading time of a web page is quite long. This could be solved by lazily loading Angular components. This would reduce the number of components that need to be loaded before the page can be visited, subsequently lowering the loading time of the web page.
- **Loading screen customisation.** For every external web page, the loading screen is the same. This does lower the immersive and professional feeling these web pages are supposed to achieve. Being able to customise that loading screen for each web page strongly increases the immersive and professional feeling compared to the current loading screen.
- **Improve the pagebuilder.** Although the current pagebuilder contains most of the planned features, there are still a lot of possible improvements that could still be implemented. The most important feature that is currently missing is good support for the horizontal placement of elements. Based on the feedback gathered during the playtests, more time was spent on the importing of custom HTML. Although it is now very easy to do this now, it is not as easy to use the features of the page builder on the imported page. This has to do with the fact that the page builder uses bootstrap for most elements, while the imported pages don't use this and have their own styling. An improvement to this can be to make the use of bootstrap an optional feature. Another improvement could be the ability to copy the style of an existing element to another element.
- **Improve the generated pages.** At the moment, the page that is generated from the page builder is simply a single HTML file. The custom CSS is inserted as an inline style tag in the header, and all the custom style changes to elements on the page are stored inline. An improvement would be to store the style in a separate CSS file.
- **Role specific information** When a user logs in with a role, they are taken to the same schedule item as all other roles in that team. Therefore it is rather unintuitive to provide information to only one role. A feature could be added that shows different schedule items for each role. Another option to solve this is to redirect a role upon login to a specific domain. This way, each role could be redirected to a different domain. On each of these domains, the information specific to that role can then be added.
- **Improved web page control** Game designers in M.O.R.S.E. can create web pages and link puzzles and ruleset triggers to it. This is technically enough to make online escape experiences, but it does not allow for finer control required by very advanced online escape experiences such as DigiHacked. With the following features, those online escape experiences can also be made with our product. Game designers should be able to create variables in which data can be stored with player interaction on web pages and that can afterwards be displayed on, possibly other, web pages. Game designers should also be able to hide or show specific elements on a web page depending on the role of the player that is currently viewing the web page. With those two features and previously described features, such as the external host, the creation of DigiHacked with this product should be possible.

In addition to features, we also have recommendations for future improvements for additional security and error handling. Currently, the server methods that are called by the clients do not have any type of checking for the given parameters. If these methods were to be called by possibly malicious users, the server throws an error for every such call. We suggest adding type checking for these server methods to improve resilience against possibly malicious users.



Project plan

A.1. Introduction

The project plan forms the basis of our bachelor project. In this project, we will be working on a problem proposed by the client. This plan defines the project that will be worked on as well as how the team is organised. An overview of the timeline of the project as a whole serves as a guideline to be able to verify if we are on track to finish the project within the specified time period.

A.2. Project description

Raccoon Serious Games is a company that develops and organises escape room inspired events for large as well as smaller groups. These are not only designed to entertain but also to educate or create awareness for a certain topic. The goal of these events is for players to learn something from these experiences and to stimulate team building. Raccoon Serious Games hosts two groups participating in the Bachelor End Project; No ReMorse and Team Kirby. These two groups will be making adjustments to the same initial codebase in separate repositories, where it is possible for the projects of both groups to have some overlap. Therefore both groups need to be careful and inform each other of their plans and adjustments made, which will be done using weekly meetings.

Our project focuses on the existing M.O.R.S.E. (Massive Online Reactive Serious Escape) 2.0 system where escape game events can be created, updated and hosted. It is designed for admins that develop the events, the game hosts and also the players of the events. In the system, puzzles, rulesets and a few other essential components for escape events can be defined. Although the system works efficiently for hosting on-location events, the functionality for providing mostly digital experiences is limited. In these digital experiences, of which DigiHacked is one, users have to interact with several (external) websites to complete the events and solve puzzles. Once the users advance during a puzzle on an external website, their progress will be communicated with the server to update and unlock other puzzles or to complete the event. M.O.R.S.E. 2.0 currently has very limited to no support for building these rich digital experiences without the need for programming. The aim of the project is to expand the system to facilitate creating these digital experiences almost exclusively in the M.O.R.S.E. 2.0 system without needing any programming experience. A few challenges have to be overcome such as the creation of user interfaces in the system, making them reactive and where possible allow cross-domain tracking. Cross-domain tracking is required to allow M.O.R.S.E. 2.0 to link external sites together and to distinguish between the different teams interacting with them. This way they don't have to enter their team code on every site, while their progress can still be tracked.

A.3. Project management

Client Raccoon Serious Games, Jan-Willem Manenschijn

TU Coach Thomas Overklift

Project team

Role	Role description	Member
Team leader	The role of the team leader is to create transparency in the team, uphold deadlines, arrange clear communication and to keep a clear overview of the project.	Wessel
Lead process documentation	Responsibility for documents describing the process of how the project is developed.	
Lead programmer	Responsible for the rulings regarding the functionality of the program and the structure of the codebase	Gijs
Lead tooling	Making sure of proper use of version control and operational tooling	
Lead testing	Responsible for the quantity and quality of written tests.	Elwin
Lead code documentation	Responsible for documents describing how the code works.	
Lead code quality	Responsible for enforcing a proper coding style.	Bram
External communicator	Responsible for general communication with external parties and setting up meetings.	
Scrum master	Has to make sure Scrum principles are correctly applied during the project.	Timo
Secretary	Has to make sure notes are taken at all meetings.	

A.3.1. Software Engineering Methodology

We have decided to use Scrum while working on this project. This decision was based on a few reasons. The most obvious one is that all team members are familiar with the workflow of Scrum, since it is a required process in other project-based courses during the Computer Science bachelor. Since it is a familiar process, we can smoothly adopt it in this project. Another reason is that our client has indicated he would like to frequently see a demonstration of updates, which can be perfectly done with Scrum due to the weekly sprints, where we have an improved version ready at the end of every sprint. The final aspect that spoke in favour of using Scrum, was the fact that we are working on the same software as another group at the same time. Therefore, flexibility is required to quickly adjust your own plans, if the new developments of the other group collide with your own.

A.3.2. Meetings

What	Who	When
Possible progress update	No ReMorse	Every Monday 09:00 - 09:30
Sprint planning	No ReMorse	Every Monday 09:30 - 10:30
Prepare client meeting	No ReMorse	Every Monday 10:30 - 11:00
Prepare Kirby meeting	No ReMorse	Every Monday 11:00 - 11:30
Update exchange	Team Kirby	Every Monday 12:00 - 13:00
Client meeting	Client	Every Monday 14:00 - 15:00
Standup meeting (only 1 member of No ReMorse required)	Raccoon Serious Games	Every Tuesday - Friday 09:00 - 09:15
Standup meeting	No ReMorse	Every Tuesday - Friday 09:30 - 09:45
Weekly master update	No ReMorse	Every Friday 15:00 - 16:00
Retrospective meeting	No ReMorse	Every Friday 16:00 - 17:00
Update exchange	TU Coach	On initiative by No ReMorse

A.3.3. General Rules

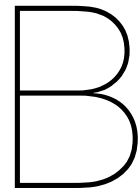
- In general, the work times are from 09:00 to 17:00, with a 1-hour lunch break.
- One should be on time for meetings.
- Work time exceptions should be communicated at least one day in advance.

A.3.4. Workflow

- The master branch should always be operational.
- A dev-branch will be used as a fail safe master branch.
 - Weekly merge to master on Friday (requiring approval of all 5 developers).
- Regular merge requests require approval from at least 2 other developers.
- All code must follow the style guide (ESLint).
- All code should be properly tested.
 - The test coverage is aimed to be above 90%.
 - UI code is excluded.
 - Other files can be excluded as well in consultation with the team, the client and the TU coach.
 - Scalability tests.
- All code must be properly documented.
 - All methods must be explained in JSDoc.
 - All code should be written in an understandable way without having to rely on in-line comments.

A.4. Project timeline

Week	Plan	Deliverables
1	Meetings with client and TU coach Create project documents Start research	
2	Make product backlog Finalize research report	Research report
3	Architecture design	
4		
5		
6	All must-haves implemented	First SIG submission
7		
8	Deadline for implementing major features	Second SIG submission
9		Final report
10	Finalize final deliverables	Infosheet Final presentation



Requirements

B.1. Functional requirements

For all stakeholders the same requirement will stand, which is that they all should keep the functionality they currently have access to in the current M.O.R.S.E. system.

B.2. Player

Must

As a player I must be able to ...

- visit an external web page that is part of the event.
- see a puzzle on an external website.
- have their progress saved in the existing M.O.R.S.E database when a player interacts with a puzzle on an external website.
- not do the same puzzle a teammate has already completed, since team progress is saved.
- see the leaderboard in M.O.R.S.E. after they are redirected to the projector page, once they have finished the event.
- see the websites properly formatted on a computer, so that unintended horizontal-scrolling is not required.
- receive a popup where they should enter their player code when the player visits an external website that does not know their player code.
- identify what their role is.
- access resources that are available to all players.
- only access the role-specific resources for their role.

Should

As a player I should be able to ...

- access external websites that are part of the game without needing to enter their player code.
- see the websites properly formatted on all modern web browsers, including browsers on tablets and smartphones.
- No unintended horizontal-scrolling.
- Zooming in or out is not required in order to be able to view and read all content on the page.
- No overlap between web elements.
- UI elements are still operable.
- see the detailed leaderboard, after they finished the event, with specific details for themselves.
- see their team highlighted on the leaderboard.
- see a message for “win”, “lose”, “in progress” on the leaderboard.
- visit an external website that does not know their player code, after having to wait for a redirect to the M.O.R.S.E. system that retrieves their player code.

Could

As a player I could be able to ...

- visit an external website without entering their player code without the redirect.
- see what input their teammates give on input places viewable to the player and their teammates.
- see a chatbot where hints can be received.
- ask questions in the chatbot.

Would

As a player I would be able to ...

- use external websites on a different device without entering their team code at least once on that device.

Admin/Game designer**Must**

As a game designer I must be able to ...

- change the style of internal pages in M.O.R.S.E. visible to the player.
- specify that a puzzle must be placed on an external website.
- use actions in rulesets which can be executed on external websites.
- use triggers from external pages in rulesets.
- create a trigger that activates on a button click on an external web page.
- create a trigger that activates when a player visits an external website.
- create a trigger that activates when a player submits a form on an external website.
- specify what web pages should be served by M.O.R.S.E. based on the domain through which the player accesses the system.
- add a stage for a leaderboard display at the end of an event.
- specify where a page created in the UI builder can be visited at a URL located within a dedicated section of M.O.R.S.E.
- add an external page visit puzzle.
- specify the page that needs to be visited in order to automatically solve the page visit puzzle.
- define roles within teams.
- define the available pages which can be viewed by each player based on the role.

Should

As a game designer I should be able to ...

- add custom messages by the leaderboard for winning players.
- add custom messages by the leaderboard for losing players.
- add custom messages by the leaderboard for when the game has not finished.
- add custom messages by the leaderboard for when the game has finished.
- add a form puzzle.
- add a condition on the answer to a form puzzle.
- add multiple inputs to the form puzzle.
- specify the input types for each form input.
- specify the correct answers for each of the form inputs.
- pause the game, such that functionality of the external websites is also paused.

Could

As a game designer I could be able to ...

- specify the style of a page in M.O.R.S.E. that is visible to the player based on triggers that happened during the event.
- create a chat bot on an external website to give predefined hints.
- change the style of the leaderboard.
- add a trigger for when an audio element has finished playing.
- add a trigger for when a video element has finished playing.
- use an action to change a page on an external website.

Would

As a game designer I would be able to ...

Foreman

Must

As a foreman I must be able to ...

- see given and correct answers for each puzzle on external websites.
- ~~access the external websites a team can see.~~

Should

As a foreman I should be able to ...

- view instructions for stages on external websites.
- send predefined hints for puzzles on external websites.

Could

As a foreman I could be able to ...

- answer questions asked by players via the chatbot.

External Host

This role does not exist in M.O.R.S.E. so this has to be added. The role itself is not a necessity and has therefore no must haves. The game could also be played without an external host, if the game designer or (internal) game host in M.O.R.S.E. gives out player codes.

Should

As an external host I should be able to ...

- see player codes
- start the event
- stop the event
- specify a custom message

Could

As an external host I could be able to ...

- share playing codes in an automated way.

Page builder

Although this will most likely be the same person as the game designer, we have defined the functionality of the page builder separately to keep a more clear overview. The responsibility of the page builder is to create the pages for the external web pages.

Must

As an external host I must be able to ...

- select a template from a finite collection to create a web page
- edit text on a web page
- select a puzzle in web page from the existing puzzles
- ~~click on an eye button that shows you a preview of the web page while building that shows you the web page in another screen as a player would see it.~~
- publish the web page to M.O.R.S.E.
- add custom HTML to a web page
- add custom CSS to a web page.
- change the styling of elements on a web page including:
 - background color, gradient or image
 - foreground color
 - font and font size
 - border size, color and rounded corners
 - padding
 - margin

Should

As an external host I should be able to ...

- drag and drop elements on a web page.
- remove elements from a web page.
- copy a web page from one event to a web page from another event.
- add elements to a web page including:

- headers
- paragraphs
- links
- empty spaces
- images
- videos
- audios
- forms
- input fields
- input areas
- buttons
- checkboxes
- radio buttons
- lists
- tables
- dividers
- footers
- undo the last edit.
- redo the last undo.

Could

As an external host I could be able to ...

- save a web page as a custom-made template.
- load custom-made templates.
- add elements though drag-and-drop.
- create a new puzzle in a specified stage in a template.
- create custom classes with styles.
- add the custom classes to elements.
- add bootstrap items including:
 - alerts
 - badges
 - breadcrumbs
 - cards
 - carousels
 - collapses
 - dropdown menus
 - jumbotrons
 - modals
 - navs
 - navbars
 - pagination
 - popovers
 - progress bars
 - scrollspies
 - tooltips
- add google maps.
- add countdown.
- add typed text.
- add colour picker.
- rotate elements.
 - have a preview for other devices:
 - desktop
 - tablet
 - mobile
- use the following hotkeys:
 - Ctrl + Z → undo

- Ctrl + Y → redo
- Ctrl + S → save
- Ctrl + C → copy
- Ctrl + V → paste
- Ctrl + X → cut
- Del → delete the selected element

Would

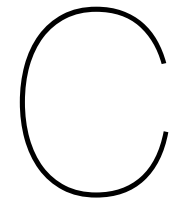
As an external host I would be able to ...

- add background audio.
- animate elements.

B.3. Non-functional requirements

- One of the important non-functional requirements, is one where some of the functional requirements are based on, is that all functionality in DigiHacked can be created in the final product. These functionalities are described in section C.3. This should be doable without having programming experience.
- As explained in the section on framework C.7, the new functionality should be added using the Meteor framework.
- Front-end tests are performed manually.
- Back-end tests are written using the Mocha ¹ framework, in combination with Meteor testing functionality.
- An escape event can be organised for a few people, but also for enormous groups with more than 100 persons participating. The product must be scalable to allow for these large groups to participate in the escape event without additional issues.
- None of the newly added features should compromise security.
- To protect their privacy, no data of the players is stored persistently, except the given teamcode, a username they enter at the start and their progress in the game.
- During an event, occurring errors must be dealt with properly, displaying a strong robustness in the system.
- The added source code must have a consistent style and must be readable. To prevent bugs, inefficiencies and possible duplicate code.
- Any conflicts in code with team Kirby should be resolved in the final product.
- MongoDB should be used for server side storage.
- The programming languages used are JavaScript, TypeScript.
- The above requirements are to be implemented as much as possible in eight weeks, starting May 4th.

¹<https://mochajs.org/>



Research

Raccoon Serious Games is a company that develops and organises escape room inspired events for large groups, as well as smaller groups [1]. These escape events are called 'Serious Games' in general. Serious Games are not only designed to entertain but also to educate or create awareness of a certain topic [2]. The goal of these events is for players to learn something from these experiences and to stimulate team building. If such events can incorporate online activities, such as gathering information from websites or entering login credentials, this could be a valuable expansion of the possibilities of escape events.

Our project focuses on the existing M.O.R.S.E. (Massive Online Reactive Serious Escape) 2.0 [4] system where escape game events can be created, updated and hosted. It is designed for the admins that develop the events, the game hosts and also the players of the events. Puzzles, rulesets and other essential components for escape events can be defined in the system. Communication with external webpages, however, can be very troublesome in this system and it is not possible to track the progress of the players in puzzles on external pages. In this research report we further analyse this problem and consider several solutions.

C.1. Overview

In this research report we will summarise the findings of our research. First, the problem definition and analysis is explained in section C.2. After that we describe the online escape experiences that should be made possible by our solution in section C.3. Here we also study the specific example DigiHacked, one of the game events Raccoon Serious Games want to be able to host in M.O.R.S.E. 2.0. In the section after this, section C.4, the M.O.R.S.E. system is summarised by giving an overview of the current functionality of the system and the features that are missing. One of the main missing features is the fact that the design of new webpages requires programming experience. To this end, we look at web page builders and summarise their functionalities in section C.5. In order to keep track of the player's progress on the external websites, we study and compare different techniques for cross-domain tracking in C.6. Based on the research done in the aforementioned sections, we draw up a set of requirements using the MoSCoW method in section C.10.

C.2. Problem definition and analysis

M.O.R.S.E. is a tool for creating and hosting escape events. In these events, the participants are divided into teams where they receive a set of questions and puzzles that need to be solved to complete the event. The puzzles can be solved by the given physical items. Raccoon Serious Games developed one of these escape events outside of the M.O.R.S.E. system called DigiHacked¹. DigiHacked is an online escape experience that provides an immersive experience that shows us the dangers of the online world and how people can be vulnerable to hackers.

DigiHacked is a great example of an online escape experience that reveals what is lacking in the current M.O.R.S.E. system. We will discuss online escape experiences more in detail in section C.3. While playing DigiHacked, you are guided through several external websites that serve as different front-ends, each of which provides their puzzles. In contrast to events made in M.O.R.S.E. , where all puzzles and questions are

¹<https://dighacked.raccoon.games/>

shown only on the M.O.R.S.E. front-end, DigiHacked provides an additional layer of reality, due to the realistic appearing websites. The main problem that we need to solve is therefore creating experiences in M.O.R.S.E. that are immersive and can be done entirely online. To solve this problem, we consider two important aspects.

The first aspect is the production of the external websites that are used as front-ends for the participants. Developing these websites for online escape experiences like DigiHacked requires programming experience. DigiHacked has shown that Raccoon Serious Games is perfectly capable of creating such an event, yet a more efficient workflow would be preferable, due to the small number of employees with programming experience. Therefore we have to find a solution that allows Raccoon employees to develop an external website, keeping various aspects in mind that need to be dealt with, such as the difficulty of usage and the easy development of prototypes to allow for a flexible workflow. In section C.5, we dive deeper into this problem.

The second aspect we need to consider is ensuring players can access the produced websites and that they can interact with them. The current M.O.R.S.E. system can track progress and process player input. With the use of multiple front-ends on external websites, tracking and processing are not as straight-forward as it is now. These front-ends should all have access to the same data so they too can remember which groups completed what puzzles and which stage of the event they should show to a certain group. They should also be able to accept input from the players and have the M.O.R.S.E. system check that input. To successfully deploy online escape experiences like DigiHacked in M.O.R.S.E. after this project, we first need to construct a solution for linking the external websites to M.O.R.S.E. and track the players' progress across these multiple websites. This second sub-problem will be further discussed in section C.6 and C.8.

C.3. Online Escape Experiences

The final goal of the project is to be able to implement online escape experiences such as DigiHacked in the M.O.R.S.E. system without the need of any programming experience. In order to be able to create such experiences, the functionality of DigiHacked has to be investigated.

C.3.1. Overview

DigiHacked is an experience that aims to teach about phishing and similar malicious actions by tricking players into them in a controlled environment. Afterwards, these malicious actions are explained and ways of identifying them are discussed. For this experience, players form teams and receive only part of the information that is needed to solve the task at hand. Although DigiHacked is just one specific example of an experience, it contains many key features that are also important for other online escape experiences. Instead of focusing on the specifics, we will explain the main features for each of the different roles, admins, hosts and players that participate in the events. This focus on the main features is done to get a clear view of the functionality that is required to be able to create new and different events in the final system.

C.3.2. Admin

The admin is an employee from Raccoon Serious Games. He or she can initiate an event whenever an external party has booked such a game. During the creation of this event, the admin can define the number of teams and players. On top of that the admin can specify the time slot for which the event is available to be played. The admin can then share this event with the host of the external party. Currently the event is shared through a link which the host can view. From this point onward, the host is able to host the event without the need for the admin.

C.3.3. Host

Once the host has received the link he or she can view the event. The host is presented with a list of teams and for each of these teams the player codes. These player codes can be shared to the players through their own communication channels and each player should receive their own player code. The host is also able to enter a custom message that the players will see at the end of the game. When players join the game using the provided codes, the host sees this mentioned after the specific player code. After all players have joined, the host can start the game. The host receives no additional information that would give them an advantage in playing, so the host can also participate using one of the player codes. The host is also able to stop the game. Stopping the game is possible before all teams have finished. The host can finally start a playback for all players of the video explaining some more of the background of the game that was played.

C.3.4. Player

Once a player has received their player code from the host, they can use this to log into the system. The logged in player can then define their name. Each of the players is assigned to a team and the players in this team can specify their team name. The teams each will go through the same experience. Before the game has started the players can read a quick introduction to the game. After the host starts the game, the player sees the main area of the game.

Player roles

Each player gets assigned a role within the game. These roles define what systems that player can access. For example, a security guard has access to the security cameras in a room. This separation of the roles makes communication between the different players crucial.

Front-ends

The main area of the game is a visual screen where each player can access a few different front ends. These front-ends range from simple text views to front-ends where there is some user interaction built in. An example would again be the security camera feed, where the player can rotate the camera or zoom in to focus on a certain clue. In addition to front-ends inside the main interface of the game, players must also interact with external sites. The actions of the players on these external sites are linked back to the team they belong to. In addition, previous actions of someone in a team influences the game behaviour for all of them. For example, when one player in the team transfers a sum of money, all the players have to accept this transfer. In one of the front-ends the player could get sent a link to their mobile phone by entering their phone number. The link was then sent as an SMS, to allow the player to use their phone's camera for two factor authentication for a certain web app. Once a team completes the game they can see the leader board with the scores of all teams. In the case of DigiHacked, it is shown whether each team has transferred the money to the correct company or to a group of hackers. Lastly, when the host has started the playback of a video to provide more background information, all players are redirected to the screen where that video is playing.

Conclusion

The mentioned features are indirectly part of the requirements of the functionality to be added to M.O.R.S.E. Some of the features might already be part of M.O.R.S.E. However, it is not possible to recreate all of the required functionality in M.O.R.S.E. The design and creation of online escape experiences in M.O.R.S.E. without the need for any programming experience is the main focus of the project.

C.4. M.O.R.S.E.

M.O.R.S.E. is the system currently used to create and manage escape room events for large groups of people. M.O.R.S.E. excels at escape events at physical gatherings. There are multiple roles for such an event. There are the teams, which are solving the puzzles. All the teams are divided into clusters. For each cluster, there is an available foreman that tracks the progress and gives hints. Then there is the project desk staff. They help the foreman and hand out physical puzzle pieces to each team. Finally, there is also the host and the admin, who have control over the event while it is running. They can change the puzzle order and can decide what should happen with wrong and correct answers. They are in full control of M.O.R.S.E. while it is running. The difference between a host and an admin, is that the admin is the only person who can create and delete an event. All the staff allows for the experience of people solving puzzles in teams and entering their answers on the M.O.R.S.E. event website to progress through the event and complete more puzzles.

The goal of the project is that experiences like DigiHacked will be possible by using M.O.R.S.E. . First we discuss the currently already available features that M.O.R.S.E. offers and that are useful when creating online escape experiences . Afterwards, the main missing features are discussed.

C.4.1. Already available and usable features

M.O.R.S.E. is a system that is currently being used to host on location events. The main idea is that the similar events can be hosted, but using an online environment. This means that some if not most of the currently available features from M.O.R.S.E. can still be used for those online experiences.

Roles

For online experiences, the admin role can be reused, because it has similar behaviour as an admin in normal physical events usually created and hosted using M.O.R.S.E. . For the players and teams in DigiHacked,

the teams and clusters in the current version of M.O.R.S.E. could possibly be used. A team in M.O.R.S.E. would then translate to a player in DigiHacked and a cluster in M.O.R.S.E. would then translate to a team in DigiHacked.

Ruleset

The current ruleset in M.O.R.S.E. handles all the logic about the transitions and availability of puzzles during an event. The ruleset is a list of rules about the event, automatically executed by M.O.R.S.E. during the event. Every rule is divided into three parts: the triggers, the conditions and the actions.

Triggers define when the rule should be checked. An example of that would be: when the host pauses the event, a certain rule should be checked and possibly executed. In the new online environment in M.O.R.S.E. it should still be possible to do everything one could do on the website in the physical events. Therefore the currently available triggers can still be used in the online environment.

When a rule is triggered, it will check if it meets all the conditions for the rule to actually apply to the event. These conditions are needed in the online environment as well for all the logic.

Actions in M.O.R.S.E. make modifications to the state of the event. For example, teams can be moved to different screens and hints can be given to teams. The possible set of actions is predefined and can be set to apply to different targets in the event. In an online event, all the the logic, puzzle transitions e.t.c. has to be done using those actions.

Schedule

The schedule available in M.O.R.S.E. gives an overview of the, often very large, events as well as handle the event flow. It shows the order of screens and groups of puzzles which can be made available for teams, clusters or even everyone in the event. In an online environment, where you cannot physically check the state and order of the event, such an overview is very important.

C.4.2. Missing features

The current version of M.O.R.S.E. does not have enough functionality to make rich online experiences such as DigiHacked. The missing features that could add that functionality are divided into the roles, ruleset features, schedule features and a web page builder tool.

Host role

In online experiences, the other role that is required for online experiences is a host. One might think, because there is already a host role in M.O.R.S.E. , it can be reused and is exactly the same. This is however not the case. A host for online experiences can be a person participating in the event as well, and thus a special kind of player. The online host has the power to start the event and to leave a final message when the event has been finished. This means that a host in the online experience is a role that should still be added into M.O.R.S.E. .

Player roles

Another feature of DigiHacked that is currently not supported in M.O.R.S.E. is the different roles a player can have within a team. M.O.R.S.E. should add the possibility of distributing the available information and tools over the team members.

Ruleset

The triggers that already exist in M.O.R.S.E. are currently designed for the physical events and do not have much support for the triggers of extra websites and puzzles. Therefore extra triggers, conditions and actions should be added to support those extra websites and puzzles.

Schedule

The schedule in M.O.R.S.E. , as noted before, allows for a great overview of bigger physical events. Moving this event online and extra websites containing extra puzzles will reduce that overview. Therefore, the extra websites and puzzles should be added to the schedule to keep the overview currently offered by M.O.R.S.E.

Web page creation tool

A part of the problem is moving the M.O.R.S.E. experience online with external connected web pages. However the other part of the problem is the fact that those external web pages are currently hard to make without programming experience. Therefore M.O.R.S.E. could include a web page creation tool that is compatible with M.O.R.S.E. . More information regarding web page and user interface design building in section C.5. Another solution would be creating an import tool that can import and modify web pages created using other web page builder tools.

C.5. User interface building

The puzzles for the escape events can be created with the M.O.R.S.E. system. Based on the type of questions used in the puzzle, a web page is generated. At the moment, there are only a two kinds of puzzles that can be made with M.O.R.S.E. , open and multiple choice questions. The main problem with this system is that the generated web pages are not really customisable. Although a few basic attributes can be changed, for example the background colour, the rest can not be changed. The goal of the project is to make it possible to have fully customisable web pages where the logic can be used with the M.O.R.S.E. system. This should be possible in a way that doesn't require any programming knowledge.

C.5.1. Web page

A web page consist of three parts: the content, the style and the behaviour. These parts are stored in three corresponding files: a HTML, a CSS and a javascript file. HTML stands for Hyper Text Markup Language and contains all the data and the structure of the web page. CSS stands for Cascading Style Sheet and it describes how the elements are displayed on the screen². The javascript file contains code which is executed when the page is first loaded or when certain events happen, such as a button press.

User Interface elements

A web page consists of a collection of User Interface (UI) elements. The elements can be things like text, images, tables, lists, buttons and other input forms. Each element has some assigned attributes which defines how the element is displayed in an internet browser. Some examples of attributes are the background colour and size. All the elements are defined in the HTML file. The attributes are defined in the style of each elements and is generally stored in a CSS file.

Layout

The layout of a web page decides where each element is displayed on the page. Elements can be grouped inside a division. The division can then be treated as a single element. The web pages might be viewed on different devices. This means that it is important for the web page to be responsive. A responsive web page automatically resizes and moves the content to make it fit on the target screen. This can be accomplished by using media queries. Media queries can tailor the presentations to a specific range of output devices without changing the content itself [7]. Another way to make the web page responsive is by using flexboxes. If an element in the flexbox doesn't fit horizontally anymore, it moves down vertically to the row below. There also exists several web design frameworks such as Bootstrap [8] which handle the responsive part of the web design. This is described in more detail in the next subsection.

Design

The files that form the web page can be written by hand or can be generated by another program. When the page is written by hand, the developer needs to have knowledge of all the syntax that is needed to write the files. This is not an option for this project, because the websites should be created without requiring any programming knowledge. The other option is to generate all the files of the web page by another program.

C.5.2. Existing tools

In this section we will analyse existing tools that can be used for building a web page. Because each page must have a connection with the M.O.R.S.E. system, there are two kinds of approaches we considered. The first is to use an existing off-the-shelf page builder which we would somehow need to link to the M.O.R.S.E. system. The second approach is to create a page builder from scratch, possibly with the help of existing libraries or frameworks when appropriate. We will first discuss all these tools before we give our conclusion about what we think the best solution is for this project.

²<https://www.w3schools.com/>

Existing page builders

There are already a lot of existing off-the-shelf page builders which can be used to create a web page without requiring programming knowledge. Some examples of this are Elementor³, Leadpages⁴ and Wix⁵. A web page that is generated from an outside program is in no way connected to M.O.R.S.E. This means that there has to be some way to link the generated web page to the functionality of M.O.R.S.E. This would be in the form of a new feature in M.O.R.S.E. which loads in a specified web page after which it can be used in the system. One important consideration for this to work is that the files of the web page must be fully accessible.

Key features Although there are many different page builders, they all tend to share some key features and generally have the same workflow. The UI elements can be dragged and dropped from a sidebar into the page. The page is a real time preview of what the generated page will look like. The elements on the page can be selected after which the style can be changed. This is done by changing the attributes. Most page builders already have some pre-build templates which can be chosen at the start. Some page builders can change the style of several elements at the same time, instead of having to manually change each element.

GrapesJS GrapesJS [9] is an open-source web builder framework. It can be used to build websites without requiring coding knowledge. With plugins, which GrapesJS fully supports, almost the whole framework can be modified. It is available for commercial use and can easily be ran locally as well as on a server. Using GrapesJS and self written plugins, one can make a web builder that can be integrated in M.O.R.S.E. Although GrapesJS does have a lot of features, they are not all that easy to use. Changing the layout of the page is not straightforward and might sometimes break the responsiveness when done incorrectly. Because it is such a big program already, if there are any bugs, which we already came across, they are difficult to find and fix.

Building from scratch

Instead of using an external program to create the web pages, we can also make a new page builder from scratch. The main benefit of doing this is that there is a lot of freedom in our design choices. It is comparatively easy to have a connection to M.O.R.S.E. , because it can be build specifically around the system. The client also indicated their preference for being able to create fast prototypes. Direct manipulation of the website in the builder allows for fast prototyping [10]. This direct manipulation is already included with the no programming experience offered by our page builer. There are some tools that can be used for making the page builder.

Bootstrap Bootstrap is an open source toolkit for developing web pages [8]. It has an extensive list of pre-built components which can be used to easily create a good looking page. It automatically handles the responsive part of the web page. This means that the page looks good on all kinds of screens, such as tablets and mobile phones. It is also compatible with all modern browsers.

There are many other frameworks which provide similar functionality, such as Foundation⁶, Semantic UI⁷ and Bulma⁸. The Bootstrap framework is built around fast prototyping and focuses on the functionality. This means that the pages made with Bootstrap might look somewhat similar at first, but the advantage is that it doesn't require heavy customisation for it to work. Bootstrap is also the most popular framework and is very well documented with a lot of extra resources available online.

GridStack Gridstack [11] is an open source Javascript/Typescript library which can be used to create a responsive layout. It can create items which are resizeable and can be dragged and dropped anywhere on a grid. This makes it extremely easy to make a layout for a web page and place all the elements on the desired location. Each layout component would be an item inside the main grid and also be a grid itself, whilst the elements inside the layout would be an item in the layout grid. For this to work, it is essential that there is good support for nested grids. Although Gridstack does allow nested grids, it is not intended to be used in the way we would like to. The elements in a grid can only be transferred to another grid which has the same depth. Therefore, it would require some modification before we can use it. On top of this, gridstack doesn't

³<https://elementor.com/>

⁴<https://www.leadpages.net/>

⁵<https://www.wix.com/>

⁶<https://get.foundation/>

⁷<https://semantic-ui.com/>

⁸<https://bulma.io/>

Criteria	Import from an external builder	Integrate an existing	Building from scratch
Integration in M.O.R.S.E.	–	+/–	+
Extendability	–	+/–	+
Easy prototyping	–	–	+
Development cost	+	+	–
Usability	+	+/–	+/–

Table C.1: Comparison of three ways of creating functionality for building web pages.

A + indicates that a certain criteria is met or is easy to meet, whereas a – indicates that a criteria cannot be met or is very hard to meet. A +/- falls in between them when it is neither hard nor easy to meet

generate pure HTML/CSS but needs to have the gridstack library on the client as well. This would make the page builder completely dependent on gridstack.

jQueryUI jQueryUI [12] is a Javascript library built on top of jQuery which contains interactions, widgets, effect and other utilities. JQueryUI can make any DOM (Document Object Model⁹) element draggable, drop-pable, resizable, selectable and sortable. The widgets are pre-built components similar to some of the bootstrap components. Although jQueryUI provides some seemingly useful interactions for any DOM element, it doesn't work in a responsive way.

C.5.3. Comparison

In table 2.1 is an overview of the advantages and disadvantages of the different approaches. Based on the advantages and disadvantages we came to the conclusion that building the page builder from scratch using Bootstrap is the best approach to take. This approach allows for a good integration with the rest of the M.O.R.S.E. system and is the most flexible solution if there is something that needs to change.

The page builder should have the same basic features as other existing page builders, such as being able to drag and drop UI elements from a sidebar into the page. There should also be the option to choose a pre-built template to allow for fast prototyping.

C.6. Cross domain tracking

As described in section C.4 M.O.R.S.E. only includes a single website where users fill in answers to puzzles in order to receive the next puzzle. In order to integrate external websites into M.O.R.S.E. properly, the means of communication between the site and M.O.R.S.E. is a very important aspect to consider. If puzzles are available on external websites, solving them should still allow the user to advance to the next puzzle and some actions on web pages should only be possible after having completed some other puzzle before. To this end, we want to be able to track the necessary interactions of a user on the external sites. However, this can be very cumbersome, because browsers will prohibit websites from interacting with the data of websites opened in another tab. It is important to note though, that we will only track users' team codes and no private information will be tracked or stored.

In this section, we compare several possible solutions. Since tracking is a very broad term, we will look at two aspects specifically for each proposed solution. The first thing we want to track is the user's identity. This is not necessarily in the form of actual personal credentials, but should at least include knowing a players team-code within M.O.R.S.E.

The second aspect we consider is the ability to track a user's progress. Examples of such progress would be having visited a site, solved a puzzle, or entering a wrong answer. Being able to track this progress is necessary if we want external puzzles to interact with M.O.R.S.E. during the event.

Cross domain tracking is used a lot in online advertisement [13] and many companies such as Google¹⁰ offer sophisticated tools to track the behaviour of users between websites [14]. After having taken a more thorough look at these tools however, we do not think they fit our needs, because they focus more on large-scale analysis of traffic, rather than quickly responding to individual users' actions. Luckily there are many other options available that fit the requirements of our project better, the most important of which are discussed below, followed by a comparison where we determine which is most suitable for our purposes.

⁹<https://www.w3.org/TR/WD-DOM/introduction.html>

¹⁰<https://about.google/>

C.6.1. No tracking

Although this may sound counter-intuitive at first, the desired result can actually be achieved without tracking external activity of the user at all. Regardless of whether a puzzle is hosted on an internal or external website, the user can still be asked to enter their answer to the puzzles in M.O.R.S.E. to continue with the event. This way M.O.R.S.E. is still able to keep track of correct and incorrect answers given, while no extra tracking of the user is required. Although this is by far the simplest solution regarding both complexity and implementation, we do not think this is a viable option, because the user will likely experience this going back-and-forth between M.O.R.S.E. and external websites to be less immersive, or even cumbersome. Alternatively, a submission form can be shown on the external page, but then the user would be required to enter their team code at least once on every external website. Although this might be a slight improvement, this can still be experienced as cumbersome by the user.

C.6.2. URL tracking

Instead of having a user manually enter their code on every website, it is possible to embed this in the URL (Uniform Resource Locator) or send it along with all communication underneath the surface. This way the user is no longer bothered with entering their team code after clicking a link. A downside of this method, however, is that a user still needs to enter their code when manually browsing to a website by entering the URL in their browser. Between the available options of embedding the code in the URL, or in the request data, the latter is probably preferable, because this way of tracking is not visible to the user and therefore contributes to the immersive experience of the escape event.

C.6.3. Session cookies

Rather than sending the team code along with each HTTP (Hypertext Transfer Protocol) request, it is also possible to store the code in a session cookie [15] in the browser. When using cookies, stateless session cookies are preferred, because they do not keep a server-side state for each session [16]. This reduces the server load, which is needed because M.O.R.S.E. needs to be able to handle a lot of users concurrently. The advantage of using cookies over sending the code along with every request is that after having visited the page once, the user can also browse to the website manually and it will still work until they restart their browser. A downside of using cookies to store this information is that cookies are domain-specific. As a consequence, external domains still do not have access to the player code, so this technique does only work in combination with one of the techniques mentioned before.

C.6.4. Redirection

An improvement to the aforementioned URL tracking can be made in a way very similar to how most Single Sign-on [17] implementations resolve the identity of a user. If the identity of the user (or in this case their team code) cannot be derived, we redirect the user to a page within M.O.R.S.E. as the first step of retrieving this code. On this login page, the user is either logged in already, in which case the team code is known and we can immediately redirect back to the external site, with the team code hidden in the response, or the user has not logged in before, in which case he still has to. The advantage of this method is that the user only has to log on once per browser session, whereas the methods mentioned before could not guarantee this. In addition, this method will also work together with both cookies and URL tracking.

URL masking A downside of redirecting like this is that users will briefly see a changed URL, which might confuse them or affect the immersive feeling in a negative way. Using URL masking we can solve this problem. URL masking allows sites to show the content of a site hosted on another domain, while still showing the same URL. This way user will still see the same URL, but behind the screens they are still being redirected. Another advantage of URL masking is that it provides a layer of abstraction over the layout of the actual site, such that the user is no longer bother with details about the actual implementation of the infrastructure the site runs on.

Internal redirection Instead of redirecting the entire page to a specific login page in M.O.R.S.E. and then back, this can also be done internally. One way in which this could be achieved is using an iframe that redirects to the login page in M.O.R.S.E. and then returns with the player code, to store it in the browser. A downside of this approach is that communication between inside and outside an iframe can be very complicated. There does exist an open source library for cross domain cookie sharing [18]. However this is a fairly unpopular

library and the last code modifications were 3 years ago and therefore not a valid option, because the code for this project should be maintainable as well as robust, which this library cannot offer. The general method used by this library, creating an invisible iframe which shares cookies from the main domain, however, is very interesting and could be implemented by ourselves.

C.6.5. Comparison

Having discussed several techniques for keeping track of the user's identity, we now compare them and decide which is most appropriate for our purposes. No tracking at all, or tracking solely via URL parameters does not provide enough immersion for the purpose of this project, therefore they are not viable solutions on themselves. They can, however, still be used as fallback methods for the one we end up using. This way, when the more sophisticated means of identification fails, the user can still be prompted to enter their code once. In addition, there is also no way around the fact that the code has to be entered by the user manually at least once, when starting the application. This will still be done using manual authentication.

Out of the remaining solutions, session cookies and URL masking seem especially promising, because they ensure the user is not bothered with any kind of need for interference after they have logged in once, and they will not notice what is happening behind the screens. We think an optimal solution would consist out of a combination of these techniques, in a way that harnesses the strengths of both. URL masking allows for all sites to be located on the same domain, while appearing to be on different domains. This way the user will not experience any difference, but communication between different sites will become much more natural, since they are, in fact, on the same domain.

C.7. Frameworks

In the project we need to extend the M.O.R.S.E. system. The current system is built with Meteor and Angular. Extending the system with other libraries and frameworks will introduce an extra level of communication between M.O.R.S.E. and the external libraries and frameworks. Since Meteor and Angular do fit our needs, we will keep using that. It will save us the extra layer of communication and since the company is already used to the old frameworks and libraries, they will have an easier time making changes to the code in the future, making it more maintainable. As we covered in C.5, we chose for building a page builder from the ground up. To support this process, Bootstrap is used to build our own UI.

C.8. Server-side solution

In this section, we discuss five different techniques that can be used to serve external web pages to the user's browser. Important aspects that are considered are their support for communicating with the existing server and frameworks, the extent to which they provide access to the data stored in the M.O.R.S.E. database, and whether the user will feel as if he/she is visiting a real website. After discussing all five techniques, we compare them and pick the one best suitable for this application.

C.8.1. Standalone server with custom communication

There is the conceptually simple solution, which is creating a standalone server where we can implement our own communication customised to our specific needs for this project. If carried out successfully, it would be a solid option to consider, since it can do exactly what we want it to. To complete such a feature would require significantly more time than the other options we are about to discuss. And for every external website that would be created for the use of an online escape experience there would also need to be a new server, again requiring customised communication to properly function. This is an addition to the already huge workload for this option. Another downside would be that a domain for the website needs to be acquired at the start of the project. This means that prototyping and changes to the external web pages are not possible, which is in direct conflict with the wishes of the client.

C.8.2. In M.O.R.S.E.

Another option for hosting the external pages would be to let M.O.R.S.E. serve them. External pages would then appear for the users as coming from the existing M.O.R.S.E. websites domain name as a sub-page. This makes it easy to prototype because serving the websites is relatively easy inside the same domain. Another benefit for this is that when adding interactive elements such as puzzles, the existing connection to the back-

end can be used. However, a major disadvantage is the fact that pages appear to be from the M.O.R.S.E. domain. This is unrealistic and therefore reduces the immersive experience for the players.

C.8.3. CNAME URL masking

In order to improve the desired immersive experience we propose to mask the URL of the pages in M.O.R.S.E. A solution could be to do this on the DNS level through CNAME forwarding [19]. However the DNS forwarding has the limitation to only redirect to the top-level domains instead of also subdomains, therefore this option would not be viable.

C.8.4. .htaccess URL masking

Another option to improve the immersive experience while serving pages from M.O.R.S.E. is to mask the URLs by editing .htaccess files on servers that run Apache¹¹. This way, it appears to the user, as if they are viewing an external page, while actually viewing a page served by M.O.R.S.E. The main advantage of this strategy is that deploying websites is rather easy and allows for easy prototyping which is something our client really wanted to include. However, the HTTP requests get forwarded through an additional server that has no purpose other than forwarding. As a result, the cookies and other session data specific to the M.O.R.S.E. domain are not sent along. This means that we need to implement additional tracking into the system. In addition, data from M.O.R.S.E. can not be shared among external websites. This causes unnecessary extra load on the browser's resources.

C.8.5. M.O.R.S.E. within external iframe

Iframes can be used to embed another document inside other web pages [20]. We could use this to display a specific page or set of pages of the M.O.R.S.E. system on an external domain. Using iframes, the requests for the M.O.R.S.E. system are sent directly to the M.O.R.S.E. domain, rather than being routed through another server first. This means that cookies and session data from the M.O.R.S.E. domain are also available within this iframe. This is a major advantage over the other methods mentioned because it does not require explicitly communicating such data outside of M.O.R.S.E. A downside of loading the websites within an iframe, though, is that the browser navigation and history will likely work poorly and the URL of pages within an external site will likely not include the full path. Although the latter can be remedied to some extent, some drawbacks will remain.

C.8.6. Comparison

Having studied several methods for extending the server to be able to serve pages as if they are being viewed from an external domain, we now have a clear overview of the available techniques. Although none of the methods studied are a perfect fit for our purposes, the last method discussed, showing M.O.R.S.E. pages within an iframe on an external domain, has the least drastic drawbacks. For this reason, iframes will be our primary solution for this part of the project. Throughout the project, however, we will try other solutions in practice as well, and try to find workarounds for their drawbacks, such that a truly optimal solution can be delivered to the client.

C.9. Conclusion

To properly formulate a conclusion, we shall reconsider the initial main problem. The M.O.R.S.E. system is currently unfit to produce online escape experiences like DigiHacked, that create a more immersive and realistic experience. Therefore, the M.O.R.S.E. system needed to be adjusted not only to recreate DigiHacked but also to create other new online escape experiences .

We mentioned two important aspects for solving this problem, of which the first was the production of external web pages. This currently was only doable if you had programming experience, which slowed the workflow of the Raccoon employees.

To allow puzzles that contribute to an immersive experience on external websites without programming experience either an off-the-shelf external website builder or a self-implemented website builder should be used. External website builders like Wix and Leadpages offer a professional look with a lot of features to create your web page. Developing our own website builder, however, can offer more flexibility for the integration in M.O.R.S.E. and can be adjusted to the preferences of the customer. Therefore, we decided to choose the

¹¹<https://www.apache.org/>

latter and implement our own website builder.

Another problem is the interaction between M.O.R.S.E. and external websites containing puzzles. The most significant parts of that are the tracking of players across different domains and communication of puzzle states to M.O.R.S.E. and back.

In order to actually serve these created websites to the user's browser, we embed them in an iFrame that is located on the external domain. This way the user will see the domain name of the external site in their browser's navigation bar, while the site itself will still be able to connect to M.O.R.S.E. , because it is, in fact, run within M.O.R.S.E. Because we are integrating the features in the already existing M.O.R.S.E. system, we will also use most of the frameworks M.O.R.S.E. is using. This means that Meteor, Angular, typescript, MongoDB and miniMongo will be used to implement to serve the external pages as well as the website builder that will be integrated into M.O.R.S.E. For testing purposes, the already existing test suite in Mocha will be extended.

With these decisions that resulted in the requirements in section C.10, we are confident the updated M.O.R.S.E. system will be able to host online escape experiences like DigiHacked.

Term	Definition
Event	The configuration of an escape game which contains a predefined set of puzzles and a ruleset which specifies the order in which the puzzles are to be solved
Trigger	Triggers define when a rule in the ruleset should be checked.
Action	Consequences applied to the current state of the event once a rule has been met.
Condition	A condition is a state that should be met before applying the action of a rule.
Progress	Describes the state of the puzzles, i.e. which puzzles are completed and the puzzles that are not yet solved
External web page	A web page that a player can access through a different domain from M.O.R.S.E.
Form puzzle	A puzzle that requires the player to enter information in a form which will be checked by pressing the send / submit button.
Page visit puzzle	A puzzle that requires the player to visit a certain page. Upon page visit, the puzzle is completed.
Button puzzle	A puzzle that requires the player to click a specified button to solve the puzzle.

Table C.2: Definitions for the requirements.

C.10. Requirements

C.10.1. Functional requirements

For all stakeholders the same requirement will stand, which is that they all should keep the functionality they currently have access to in the current M.O.R.S.E. system.

Player

Must

As a player I must be able to ...

- visit an external web page that is part of the event.
- see a puzzle on an external website.
- have their progress saved in the existing M.O.R.S.E database when a player interacts with a puzzle on an external website.
- not do the same puzzle a teammate has already completed, since team progress is saved.
- see the leaderboard in M.O.R.S.E. after they are redirected to the projector page, once they have finished the event.
- see the websites properly formatted on a computer, so that unintended horizontal-scrolling is not required.
- receive a popup where they should enter their player code when the player visits an external website that does not know their player code.
- identify what their role is.

- access resources that are available to all players.
- only access the role-specific resources for their role.

Should

As a player I should be able to ...

- access external websites that are part of the game without needing to enter their player code.
- see the websites properly formatted on all modern web browsers, including browsers on tablets and smartphones.
- No unintended horizontal-scrolling.
- Zooming in or out is not required in order to be able to view and read all content on the page.
- No overlap between web elements.
- UI elements are still operable.
- see the detailed leaderboard, after they finished the event, with specific details for themselves.
- see their team highlighted on the leaderboard.
- see a message for “win”, “lose”, “in progress” on the leaderboard.
- visit an external website that does not know their player code, after having to wait for a redirect to the M.O.R.S.E. system that retrieves their player code.

Could

As a player I could be able to ...

- visit an external website without entering their player code without the redirect.
- see what input their teammates give on input places viewable to the player and their teammates.
- see a chatbot where hints can be received.
- ask questions in the chatbot.

Would

As a player I would be able to ...

- use external websites on a different device without entering their team code at least once on that device.

Admin/Game designer

Must

As a game designer I must be able to ...

- change the style of internal pages in M.O.R.S.E. visible to the player.
- specify that a puzzle must be placed on an external website.
- use actions in rulesets which can be executed on external websites.
- use triggers from external pages in rulesets.
- create a trigger that activates on a button click on an external web page.
- create a trigger that activates when a player visits an external website.
- create a trigger that activates when a player submits a form on an external website.
- specify what web pages should be served by M.O.R.S.E. based on the domain through which the player accesses the system.
- add a stage for a leaderboard display at the end of an event.
- specify where a page created in the UI builder can be visited at a URL located within a dedicated section of M.O.R.S.E.
- add an external page visit puzzle.
- specify the page that needs to be visited in order to automatically solve the page visit puzzle.
- define roles within teams.
- define the available pages which can be viewed by each player based on the role.

Should

As a game designer I should be able to ...

- add custom messages by the leaderboard for winning players.
- add custom messages by the leaderboard for losing players.
- add custom messages by the leaderboard for when the game has not finished.
- add custom messages by the leaderboard for when the game has finished.
- add a form puzzle.
- add a condition on the answer to a form puzzle.
- add multiple inputs to the form puzzle.

- specify the input types for each form input.
- specify the correct answers for each of the form inputs.
- pause the game, such that functionality of the external websites is also paused.

Could

As a game designer I could be able to ...

- specify the style of a page in M.O.R.S.E. that is visible to the player based on triggers that happened during the event.
- create a chat bot on an external website to give predefined hints.
- change the style of the leaderboard.
- add a trigger for when an audio element has finished playing.
- add a trigger for when a video element has finished playing.
- use an action to change a page on an external website.

Would

As a game designer I would be able to ...

Foreman**Must**

As a foreman I must be able to ...

- see given and correct answers for each puzzle on external websites.
- access the external websites a team can see.

Should

As a foreman I should be able to ...

- view instructions for stages on external websites.
- send predefined hints for puzzles on external websites.

Could

As a foreman I could be able to ...

- answer questions asked by players via the chatbot.

External Host

This role does not exist in M.O.R.S.E. so this has to be added. The role itself is not a necessity and has therefore no must haves. The game could also be played without an external host, if the game designer or (internal) game host in M.O.R.S.E. gives out player codes.

Should

As an external host I should be able to ...

- see player codes
- start the event
- stop the event
- specify a custom message

Could

As an external host I could be able to ...

- share playing codes in an automated way.

Page builder

Although this will most likely be the same person as the game designer, we have defined the functionality of the page builder separately to keep a more clear overview. The responsibility of the page builder is to create the pages for the external web pages.

Must

As an external host I must be able to ...

- select a template from a finite collection to create a web page
- edit text on a web page
- select a puzzle in web page from the existing puzzles
- click on an eye button that shows you a preview of the web page while building that shows you the web page in another screen as a player would see it.

- publish the web page to M.O.R.S.E.
- add custom HTML to a web page
- add custom CSS to a web page.
- change the styling of elements on a web page including:
 - background color, gradient or image
 - foreground color
 - font and font size
 - border size, color and rounded corners
 - padding
 - margin

Should

As an external host I should be able to ...

- drag and drop elements on a web page.
- remove elements from a web page.
- copy a web page from one event to a web page from another event.
- add elements to a web page including:
 - headers
 - paragraphs
 - links
 - empty spaces
 - images
 - videos
 - audios
 - forms
 - input fields
 - input areas
 - buttons
 - checkboxes
 - radio buttons
 - lists
 - tables
 - dividers
 - footers
- undo the last edit.
- redo the last undo.

Could

As an external host I could be able to ...

- save a web page as a custom-made template.
- load custom-made templates.
- add elements though drag-and-drop.
- create a new puzzle in a specified stage in a template.
- create custom classes with styles.
- add the custom classes to elements.
- add bootstrap items including:
 - alerts
 - badges
 - breadcrumbs
 - cards
 - carousels
 - collapses
 - dropdown menus
 - jumbotrons
 - modals
 - navs
 - navbars

- pagination
- popovers
- progress bars
- scrollspies
- tooltips
- add google maps.
- add countdown.
- add typed text.
- add colour picker.
- rotate elements.
 - have a preview for other devices:
 - desktop
 - tablet
 - mobile
- use the following hotkeys:
 - Ctrl + Z → undo
 - Ctrl + Y → redo
 - Ctrl + S → save
 - Ctrl + C → copy
 - Ctrl + V → paste
 - Ctrl + X → cut
 - Del → delete the selected element

Would

As an external host I would be able to ...

- add background audio.
- animate elements.

C.10.2. Non-functional requirements

- One of the important non-functional requirements, is one where some of the functional requirements are based on, is that all functionality in DigiHacked can be created in the final product. These functionalities are described in section C.3. This should be doable without having programming experience.
- As explained in the section on framework C.7, the new functionality should be added using the Meteor framework.
- Front-end tests are performed manually.
- Back-end tests are written using the Mocha ¹² framework, in combination with Meteor testing functionality.
- An escape event can be organised for a few people, but also for enormous groups with more than 100 persons participating. The product must be scalable to allow for these large groups to participate in the escape event without additional issues.
- None of the newly added features should compromise security.
- To protect their privacy, no data of the players is stored persistently, except the given teamcode, a username they enter at the start and their progress in the game.
- During an event, occurring errors must be dealt with properly, displaying a strong robustness in the system.
- The added source code must have a consistent style and must be readable. To prevent bugs, inefficiencies and possible duplicate code.
- Any conflicts in code with team Kirby should be resolved in the final product.
- MongoDB should be used for server side storage.
- The programming languages used are JavaScript, TypeScript.
- The above requirements are to be implemented as much as possible in eight weeks, starting May 4th.

¹²<https://mochajs.org/>

D

Admin manual

Admin Manual

1. Overview	2
2. Admin	3
2.1 Logging in as an admin	3
3. Create a website	4
3.1 Create domains and pages	4
3.2 Create 'triggers' to navigate the pages	5
3.3 Link the triggers to the buttons	6
4. Puzzles	7
4.1 Add new puzzles	7
4.2 Link the puzzles	8
5. Roles	9
5.1 Add new roles	9
6. Create leaderboards	10
6.1 Create one (or more) leaderboards	10
7. Test the event	11
7.1 Go to the MORSE event	11
7.2 Go to the external page	11
7.3 Go to the internal page	12
8. Import and export pages	13
8.1 Get the HTML+CSS of pages outside morse	13
8.2 Import a page	13
8.3 Import a custom HTML block	14
8.4 Export the page	15
9. Design a webpage	16
9.1 Add new blocks	16
9.2 Change the image/video	16
9.3 Change the style	16
9.4 Load a template	16
9.5 Pagebuilder hotkeys	17

1. Overview

This manual describes the steps that are needed to use the additional functionality to MORSE added by team NoRemorse. Specifically, how to create and simulate a fake website which can contain the existing or new puzzles from MORSE; how to create roles and how to add leaderboards.

A **domain** is the address of a website that can consist of multiple **web pages**.

To give a comparison: if the domain is a house, then the web pages are the rooms inside that house.

Or a real example: raccoon.games is a **domain**; and

raccoon.games/wat-is-een-serious-game/ is a **web page** on that domain.

All the domains and pages have to be **created** in the **schedule**, and start out as empty pages without any functionality or visual elements on them. See (3.1) for how to create the domains and pages.

The **design** of the web pages is done in the **page builder** where you can load templates (see chapter 8); drag-and-drop elements into the page; or change the styling of the elements (see chapter 9).

Since you can only see 1 page at the time on each domain, it can be useful to add **triggers** which can be used to **navigate** the pages on the domain. See (3.2) and (3.3) for how to create and link the triggers

Each web page can contain **puzzles**, but they are not present on the page automatically. The puzzles have to be **created** in the **schedule**, and have to be linked manually in the **page builder** afterwards. See (4.1) for how to create and link puzzles.

2. Admin

2.1 Logging in as an admin

1. Go to the URL: **<MORSE Domain>/e/<EventName>/admin**
 - a. *For the playtests, if not noted otherwise, the URL is <https://noremore.escape-room.app/e/playtest/admin>*
2. **Log in** with an admin **code**
 - a. *For the playtests, the code is: "admin"*
3. You should be in the event now, logged in as an **admin**

3. Create a website

This chapter describes how to create a website in MORSE.

All the pages start out as empty pages without any functionality or visual elements on them. The **design** of the web pages is done in the **page builder** where you can load templates (see chapter 8); drag-and-drop elements into the page; or change the styling of the elements (see chapter 9).

3.1 Create domains and pages

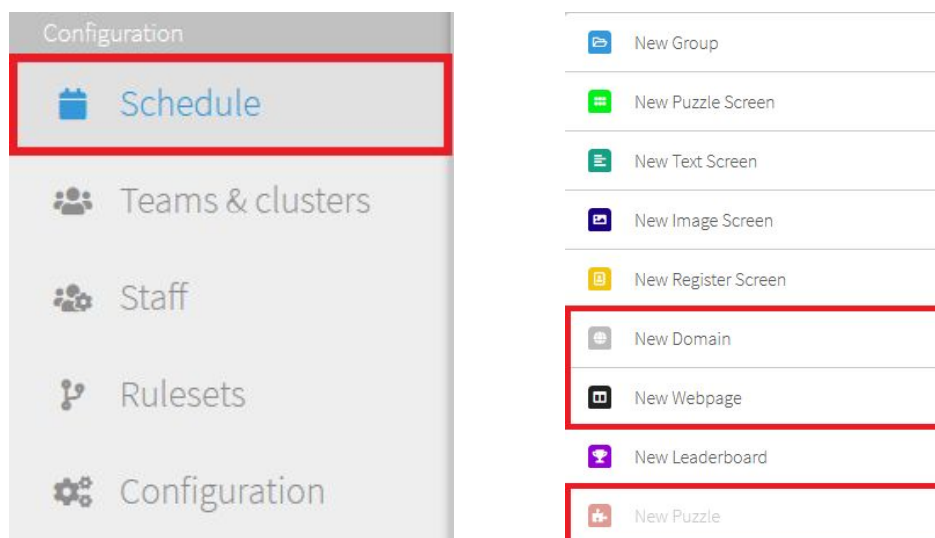
A **domain** is the address of a website that can consist of multiple **web pages**.

To give a comparison: if the domain is a house, then the web pages are the rooms inside that house.

Or a real example: raccoon.games is a **domain**; and

raccoon.games/wat-is-een-serious-game/ is a **web page** on that domain.

1. Go to the **Schedule**
2. Add a new **domain** schedule item
 - a. *The “Domain Name” is the url of the domain*
3. Add new **page(s)** to the domain
 - a. *Click on the domain you want to add the new page to*
 - b. *Now, if you click on “new page”, it will add this page to the selected domain*
4. (Optional) Add **puzzles** to the **pages**
 - a. *See chapter “Puzzles” for more info*
5. (Optional) Add **roles** to the **domain**
 - a. *See chapter “Roles” for more info*

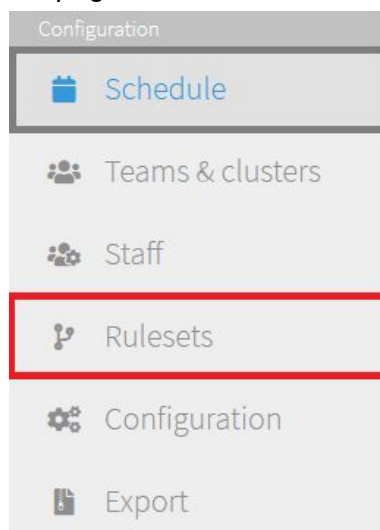


3.2 Create 'triggers' to navigate the pages

Since you can only see 1 page at the time on each domain, it can be useful to add **triggers** which can be used to **navigate** the pages on the domain.

A trigger is linked to a button on the page (see next subsection), and when it is clicked by a player, they can be redirected to another page.

1. Go to the **Rulesets**
2. Add new **rule**
 - a. *Make sure to set the name of the rule before saving*
3. Add a new **trigger**:
 - a. *the button (choose a name) ... on the domain ... was pressed*
 - b. *Choose a logical name for the trigger, this name will be visible in the page builder*
4. Add a new **action**:
 - a. *move ... on domain ... to the page ...*
5. (**Note**) The **condition** is not needed in this case
6. The **next section** explains how to **link** these triggers to the **buttons** on the webpage
 - a. *The triggers you create in the ruleset don't do anything yet. They only describe the 'logic' of what should happen. They have to be linked to an actual button on the page itself*




Navigate to Another page

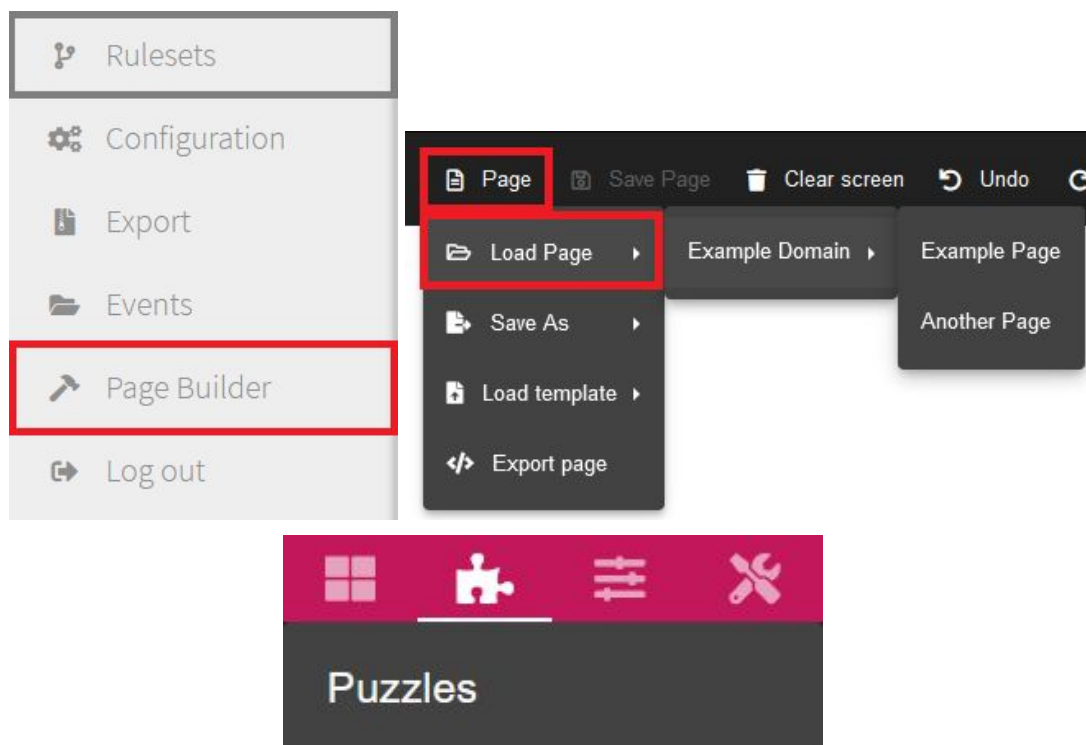
Run this ruleset: infinitely at most time(s) disabled

- if the button (choose a name) on the domain was pressed
- add trigger
- add condition
- then move on domain to the page
- add action

3.3 Link the triggers to the buttons

The triggers you create in the ruleset don't do anything yet. They only describe the 'logic' of what should happen. They have to be linked to an actual button on the page itself.

1. Go to the **Page Builder**
2. **Open** the page you want to edit
 - a. *In the top left of the page builder → “Page” → “Load Page”*
3. Go the **puzzle** tab
 - a. *In the sidebar on the right → the second tab *
4. Drag the **trigger(s)** of the domain into the canvas
 - a. *An existing element on the page can be linked to a trigger by first selecting the element on the page, then clicking the “Link to selected” button*
 - b. *The functionality of the triggers can only be changed in the “Rulesets”*
 - c. *It is not possible to create new triggers in the “Page Builder”*
5. (Note) You can read more about **designing** the page in chapter “Design a webpage”

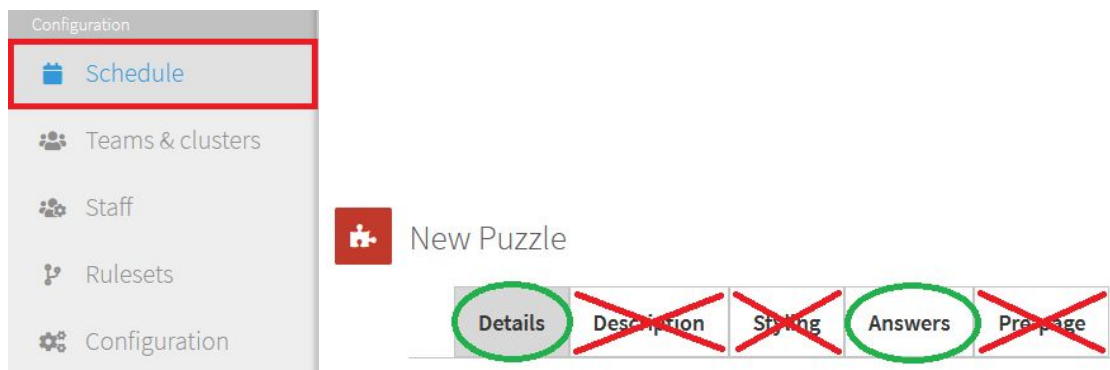


4. Puzzles

The puzzles are exactly the same as they were before in the existing MORSE system. They can now also be added to the new webpages.


4.1 Add new puzzles

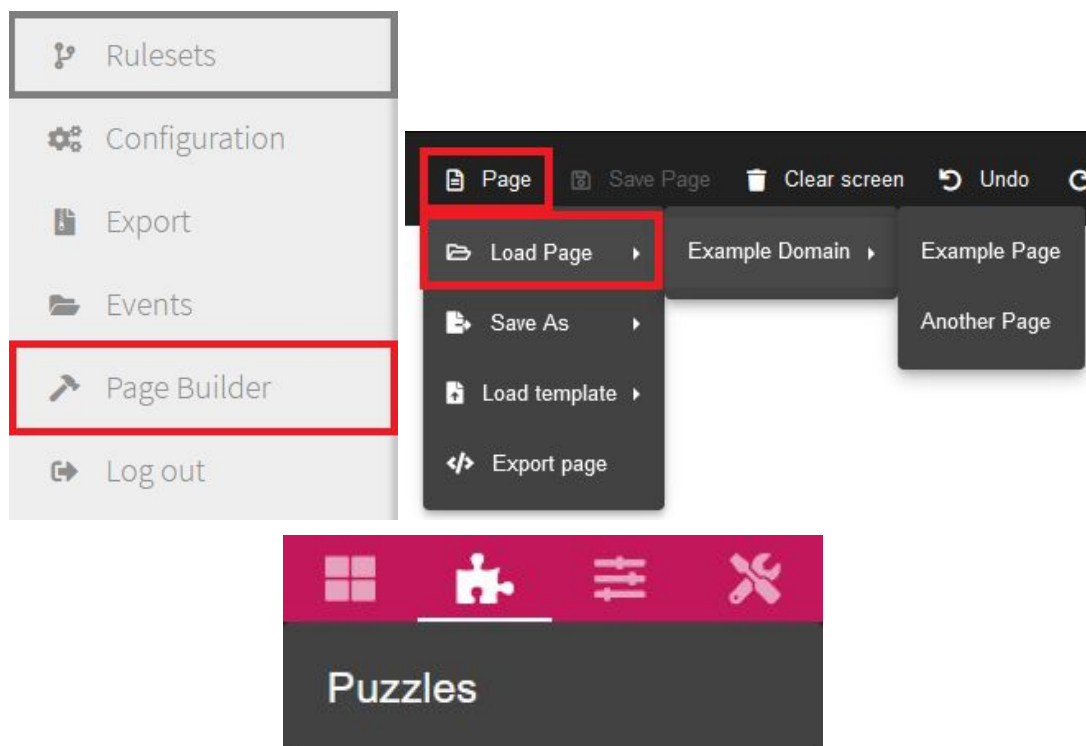
1. Go to the **Schedule**
2. Go to a **page** on a **domain** and add a **puzzle**
3. Only the **Details** and **Answers** tab are needed for the external puzzles.
 - a. In the **Details** tab, you select the puzzle type
 - b. In the **Answers** tab you can set the answer(s) of the puzzle
4. The **next section** explains how to **link** these puzzles to the **elements** on the page
 - a. *The triggers you create in the schedule don't do anything yet. They only describe the 'logic' of the puzzle. They have to be linked to actual element on the page itself*



4.2 Link the puzzles

The triggers you create in the schedule don't do anything yet. They only describe the 'logic' of the puzzle. They have to be linked to actual element on the page itself.


1. Go to the **Page Builder**
2. **Open** the page where the puzzle should be
 - a. *In the top left of the page builder → "Page" → "Load Page"*
3. Go the **puzzle** tab
 - a. *In the sidebar on the right → the second tab *
4. Drag the **puzzle** of the page into the canvas
 - a. *An existing element on the page can be linked to a puzzle component by first selecting the element on the page, then clicking the "Link to selected" button*
 - b. *You can drag the individual components of the puzzle into the canvas*
 - c. *The functionality of the puzzles can only be changed in the "Schedule"*
 - d. *It is not possible to create new puzzles in the "Page Builder"*
5. (Note) You can read more about **designing** the page in chapter "Design a webpage"

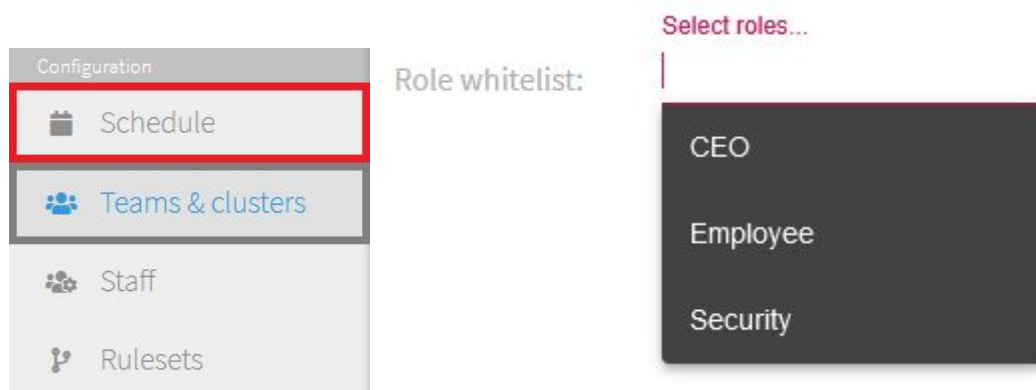
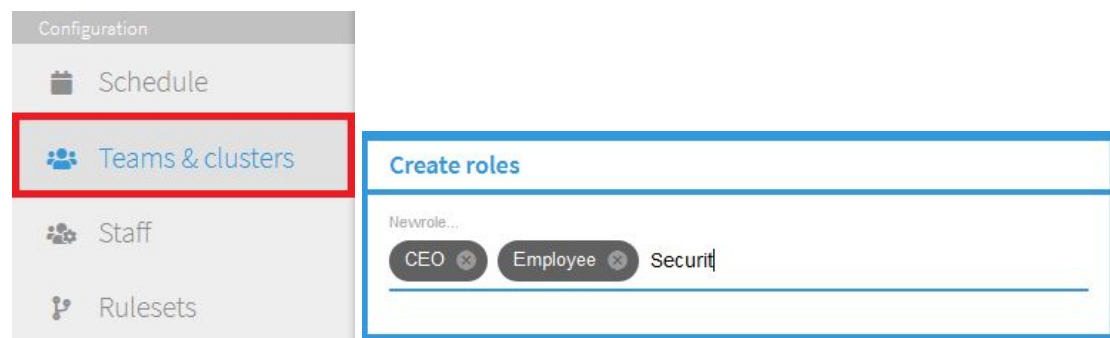


5. Roles

Each player in a team can be assigned to a role. For each domain, you can set the roles which are allowed to visit the pages on that domain. Every player which doesn't have an allowed role cannot visit this page.

5.1 Add new roles

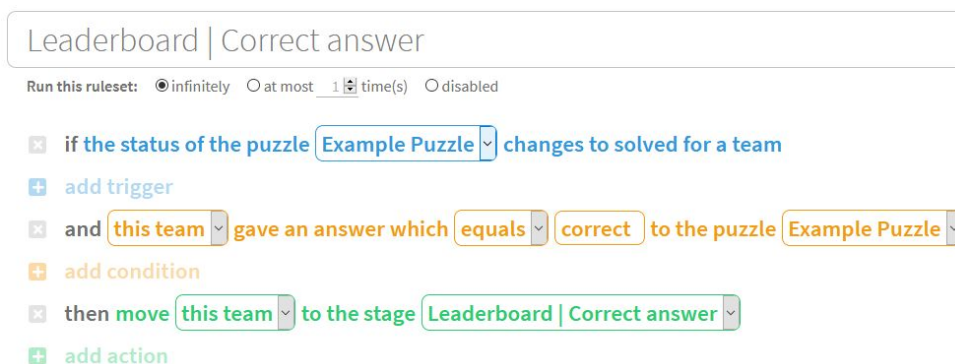
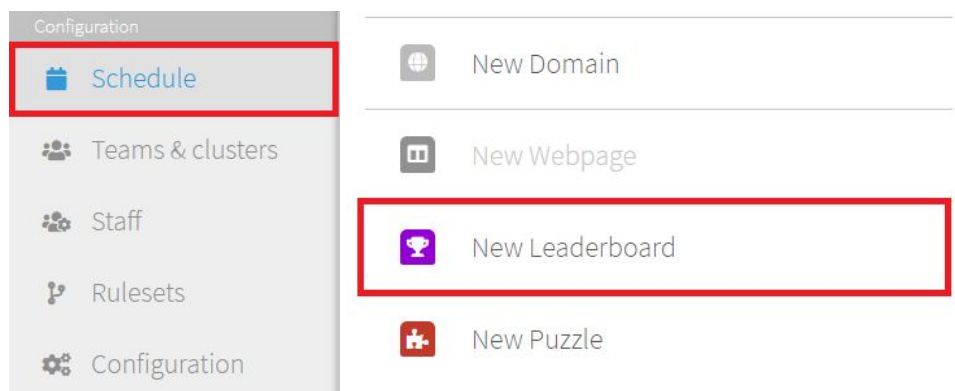
1. Go to the **Teams & Clusters**
2. Type a **role** and press enter or comma to add it to the roles
 - a. *The roles can be found on the right side all the way at the bottom*
3. Go to the **Schedule**
4. Select a **domain** 
5. Add the roles allowed to enter the domain to the **Role whitelist**
 - a. **(Important)** *If everybody is allowed to visit the page, you can simply leave this empty*



6. Create leaderboards

6.1 Create one (or more) leaderboards

1. Go to the **Schedule**
2. Add a new **leaderboard** schedule item
 - a. You can enter a **custom text** for this leaderboard
3. (Optional) Repeat step 2 if you want **multiple** leaderboards with **different** text
 - a. So teams can be shown different leaderboards based on their result
4. Go to the **Rulesets**
5. Add a **rule** that moves players to a **leaderboard** with an **action**
 - a. The trigger and (optional) conditions can be decided by the admin.
6. **Example** to display a leaderboard based on the answer of the puzzle
 - a. (**trigger**) if the status of the puzzle ... changes to solved for a team
 - b. (**condition**) and ... gave an answer which ... to the puzzle ...
 - c. (**action**) then move ... to the stage ...
7. (Important) To compare the answer given by a form puzzle, you need to separate each answer by a comma and a space like this `, `
 - a. For example: "and ... gave an answer which equals 'username, password' to the puzzle ..."




7. Test the event

7.1 Go to the MORSE event

1. If you're logged in in the same browser, either **logout** or open a browser in a new **incognito** window
2. Go to the URL: **<MORSE Domain>/e/<EventName>/login**
 - a. **<MORSE Domain>** is the website where MORSE is hosted
 - b. For the playtests, the **<MORSE domain>** is <https://noremorse.escape-room.app>
 - c. **<EventName>** is the event you are currently editing
3. **Login** with a team or role **code**
 - a. If this is the first time logging in with a code, you have to enter a **team name**
 - b. You can find the codes in the **"Teams and Clusters"** tab
4. You should be in the event now, logged in as a **user**

7.2 Go to the external page

1. If you're logged in in the same browser, either **logout** or open a browser in a new **incognito** window
2. Go to the URL: **<MORSE domain>/domain/<domainName>**
 - a. **<MORSE Domain>** is the website where MORSE is hosted
 - b. For the playtests, the **<MORSE domain>** is <https://noremorse.escape-room.app>
 - c. **<domainName>** is the domain of your domain schedule item
3. If you are not logged in then
 - a. **Login** with a team or role **code** and the **event code**
 - b. If this is the first time logging in with a code, you have to enter a **team name**
 - c. You might need to repeat step 1 after logging in



Example domain

Details


Domain Name:
example

External: Yes

Delete schedule item

7.3 Go to the internal page

1. If you're logged in in the same browser, either **logout** or open a browser in a new **incognito** window
2. Go to the URL: **<MORSE domain>/game/domain/<domainName>**
 - d. **<MORSE Domain>** is the website where MORSE is hosted
 - e. For the playtests, the **<MORSE domain>** is <https://noremorse.escape-room.app>
 - f. **<domainName>** is the domain of your domain schedule item
4. If you are not logged in then
 - a. **Login** with a team or role **code** and the **event code**
 - b. If this is the first time logging in with a code, you have to enter a **team name**
 - c. You might need to repeat step 1 after logging in

 Example domain

Details

Domain Name:	example
External:	Yes




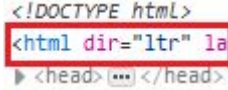
[Delete schedule item](#)

8. Import and export pages

8.1 Get the HTML+CSS of pages outside morse

The best way to get the HTML+CSS of pages made outside MORSE, is to have the original source files available. If the content is spread out over multiple files, you'll need to manually combine all the code to a single file.


Option 1: copy to clipboard

1. Go to the web page you want to get the HTML of
2. Open the devtools of you browser by pressing **ctrl+shift+i**
 - a. Except **Edge**: Use *f12* instead
 - b. For **macOS**:  +  + 
 - c. Alternatively: right-click anywhere on the page → *Inspect Element*
3. Go the the **Inspector** tab
 - a. (*Note*) each browser is different and things might look the same.
https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_are_browser_developer_tools
4. Right click on the **<html>** tag
 - a. Scroll all the way to the top to find this 
5. Press **Copy** → **Outer HTML**
6. The **next section** describes where to paste this HTML in the page builder
7. Find **CSS** files in “Style Editor” (only tested for **Firefox**)

Option 2: download the page



1. Go to the web page you want to get the HTML of
2. Save the page with **ctrl+s** (or **cmd+s**)
3. Save as “Web Page, complete”
4. The **next section** describes where to upload this HTML file in the page builder

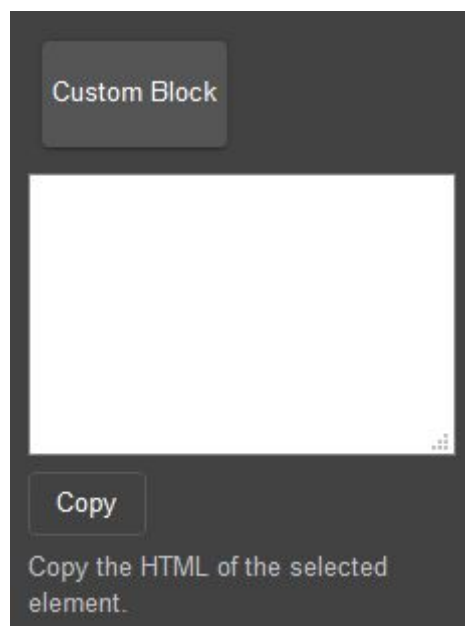
8.2 Import a page

1. In the right menu, go the “**Customization**” menu, marked by the  icon
2. Under “**Import HTML**”, either choose to import an HTML file or paste the HTML code into the text field
3. Press “**Apply**”
 - a. *All the old element will get overwritten by the new page*
4. (Optional) Under “**Import CSS**”, you can import a CSS file or paste the CSS code into the text field
 - a. *Sometimes the CSS is included in the HTML, sometimes it is in a separate file*



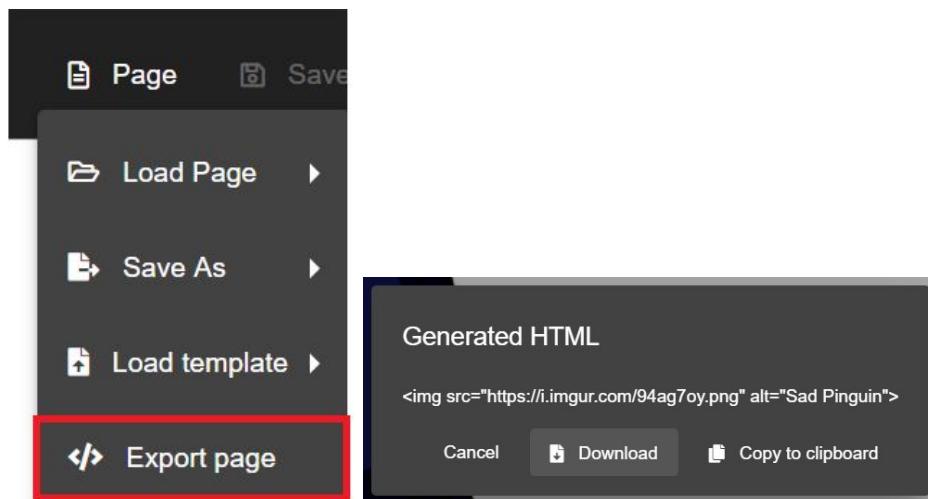
8.3 Import a custom HTML block

1. In the right menu, go the “**Blocks**” menu, marked by the  icon
2. Open the “**Custom Block**” 
 - a. *This is the last of the block groups on the bottom*
3. **Paste** the custom HTML in the textarea
4. **Drag-and-drop** the custom block into the page the same way all the other blocks are dragged into the page.
5. (Note) If you import a script as a custom block, you might need to save and reload the page before it is loaded correctly.



8.4 Export the page

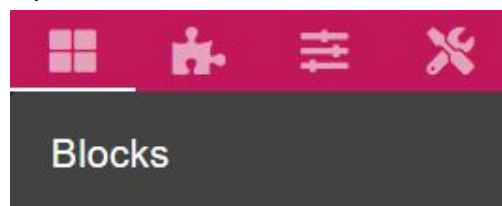
1. To **export** the current page, go to the top menu → “Page” → “Export page”
2. On the **popup**, either press the “Download” button or the “Copy to clipboard” button
 - a. “Download” will create a `.html` file on you machine
 - b. “Copy to clipboard” will copy the code of the page to your clipboard
3. You can now **import** the page again as an HTML file as described in the previous section
 - a. *This is useful for transferring an existing page to another event*



9. Design a webpage

9.1 Add new blocks


1. You can **drag-and-drop** new block from the “**Blocks**” tab
2. The blocks are grouped per **category**
3. It is recommended to add **layout** components first before the other blocks
4. On the bottom, there is a “**Custom Block**” which can be used to:
 - a. **Import/paste** certain element from outside the Page Builder.
 - b. **Copy** the current selected element
 - c. **Modify** the copied element



9.2 Change the image/video

1. Make sure to **select** the element you want to change first
 - a. For some elements, this icon  can be used to select the element
2. You can change the image or video in the “**Properties**” tab
3. Under “**Attributes**”, you can set the “**src**” of the image
 - a. For example: “https://arealbaank.nl/assets/img_logo_baank.png”

9.3 Change the style

1. Make sure to **select** the element you want to change first
 - a. For some element, this icon  can be used to select the element
2. You can change the style of the selected element in the “**Properties**” tab
3. Under “**Style**”, you can change the style attributes of the selected element

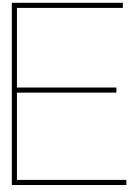
9.4 Load a template

1. You can search for **templates online**, for example on this website:
 - https://www.w3schools.com/css/css_rwd_templates.asp
 - a. The pagebuilder only supports HTML + CSS + JS
2. **Copy/download** all the code of the template
 - a. For the example site above, press “Try it Yourself” to see the code
 - b. If the code is spread out over multiple files, you need to manually modify the files to merge it into a single file (this is not the case for the examples above)
3. **Import** the code as a page (see chapter “Import and export pages”)
 - a. Images will most likely not be present in the imported template

9.5 Pagebuilder hotkeys

Below is a list of all the hotkeys in the pagebuilder:

Hotkey	Description
ctrl+z (or cmd+z)	Undo the previous action
ctrl+y (or cmd+y)	Redo the last undo
ctrl+c (or cmd+c)	Copy the selected element
ctrl+x (or cmd+x)	Copy and delete the selected element
ctrl+v (or cmd+v)	Paste the copied element below the selected element
ctrl+d (or cmd+d)	Copy and paste the selected element below
delete (or fn+backspace)	Delete the selected element
t	Select the 'parent' of the selected element (if it exists)
g	Select the first 'child' of the selected element (if it exists)
h	Select the next element (if it exists)
f	Select the previous element (if it exists)



Scalability Test

In this document, we provide details regarding our plan for the scalability test. We will start with what we roughly wanted to test. Then provide context to how we have tested these objectives. We conclude with everything we noticed and learned about the scalability test.

E.1. Testing objectives

With this scalability test, we wanted to test the following:

- Adding web pages containing a lot of data
- Creating many teams with many roles
- Sending a page hint to everyone
- Many players submitting an answer to the same puzzle simultaneously
- Many players loading the same web page
- Many players firing the same trigger simultaneously

E.2. Testing set-up

We hosted our event using a server provided to us by Raccoon Serious Games that we could manually control. For this server, we could choose what container to use while it was running, which decided the amount of workload it could handle from the product. We could opt out of 3 different containers, with each having a different amount ECU [21]:

- Standard container - 1 ECU
- Double container - 2 ECU
- Quad container - 4 ECU

The Quad container gets employed at real events, so checking the results on the Quad container will give us the final verdict regarding if something is too slow or not, whereas the other two containers will give us a better estimation of which parts of the system cause delay and if so to what extent.

We used the following to simulate a large amount of players:

- 5 real people
- 11 laptops (Windows / Ubuntu)
- 2 Android phones
- 1 iPad

On each device, we opened all the different browsers installed on them. These could all be used as a unique user. Then we also opened an incognito tab for each browser, which again were all potential unique users. So for every used browser on a device, two unique users could be participating. However, we used less unique player codes than the amount of unique players we could simulate, since communicating so many codes across so many accounts was too cumbersome to do via an online meeting. This meant that some of the browsers we used, shared the same player code, therefore, being treated by M.O.R.S.E. as the same user. Chrome, Firefox, Edge and Safari were used for testing. To simulate more user workload, we also made sure to open a lot of new tabs that also were participating with the event. Similar to some the unique browser and incognito tabs, this did not lead to more unique users, but simulated more users putting workload onto the product.

E.3. Results

We discuss each of the items on our list of tests we wanted to perform, to tell how it went and what we concluded from each of them.

E.3.1. Data-heavy web pages

We inserted a page containing roughly 5 to 10 MB of data to test. These pages need a little bit longer to load, and after looking into it, the data is already present when entering the page, but the rendering of all the data takes more time. If we look at the escape event DigiHacked as a baseline, the sites will not get that complicated, since an increase of complexity within a website is also an increase on the workload for the Raccoon employees. It is, therefore, also in the interest of the Raccoon employees to keep the web pages simple and separate them in smaller web pages.

E.3.2. Creating many players with many teams

This was a test where the performance was quite worse than expected. When adding 10 teams with 20 roles, you did get the result you wanted, but you had 100% CPU usage during roughly 4 minutes. However, the creation of teams and the generation of codes does not need to happen live during the event but can be done before-hand with no time limit. Therefore, this is a problem with relatively low impact.

E.3.3. Sending a page hint to everyone

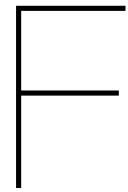
This test went a lot better than expected. All players on all tabs received a page hint all roughly the same time (there was a little difference between browsers, but that was roughly half a second as a maximum delay).

E.3.4. Stress-test

There are two more objectives to test. Firing a lot of triggers at the same time and loading the same web page simultaneously for a lot of players. For this, we created a stress-test that created a loop of redirects. Once you entered the first site, you fired the trigger that you visited that site and then you get redirected to another site that needs to be loaded. Then on that other site, we did the same with a redirect back to the first site. If you have more than 200 players stuck in this loop, the standard container would not be recommended at all. We barely saw screens loading at all, with the majority of the tabs remaining a white screen. If you used the double container then it barely passed as a playing experience, where most of the tabs did perform a redirect from one page to another, but there was a pause of 5 to 10 seconds very often before the next redirect came through. Only when using the quad container, you get a decent performance in the eyes of the player. It still was not as fast as when only playing when 10 players and performing simple actions, where everything happened instantly. However, because the max delay that we have seen is roughly a second, we are quite content seeing as 200 players putting the maximum workload on the server.

E.3.5. Browser difference

Using Edge as your browser seemed to have slightly better results overall. When we looked at our computer's task manager, we saw that this was caused by Edge claiming a lot more of idle resources to achieve this, while other browsers were way more efficient achieving just a slightly worse result. With 8 idle Edge tabs running, roughly 2.6 GB of memory was used and 12% of CPU usage, while with Firefox or Chrome only around 0.9 GB of memory was used and an average of 2% of CPU usage.



Project description

F.1. Het project

Raccoon Serious Games heeft verschillende software systemen wat ze gebruiken voor het hosten van grootschalige en fysieke escape rooms. Op dit moment heeft dit systeem slechts één front-end, met altijd dezelfde structuur, met minimale aanpassingsvariabelen. Omdat wij steeds op zoek zijn naar het maken van immersievere ervaringen, lopen wij tegen de limieten aan van dit systeem, M.O.R.S.E. We hebben verschillende front-ends nodig, gekoppeld aan verschillende domeinen om de gebruiker een bijzondere ervaring te bieden. We hebben ook al een spel die dit gebruikt, maar nog op legacy code draait. De uitdaging en opdracht in dit project is het flexibel en modulair maken van MORSE, zodat we met verschillende front-ends en cross-domain kunnen werken. Daarnaast kan het zijn dat er extra uitdagingen te voorschijn komen onderweg waardoor het nodig is om aanpassingen te maken in het MORSE systeem. Hiervoor wordt een kritische en adviserende blik verwacht.

F.2. About Popup-escape

Popup-escape is a young company/startup started by a Delft Computer Science student two years ago. It started out as a hobby and it is now grown to a full-time job with multiple part-time employees. (Also some Computer Science students). We have designed and made about 30+ escape rooms over the past two years and at least 3000 people have played the different escape rooms.



Playtest Webshop

You will be following a scenario of creating an online escape event. Because of limited time, we will make all the web pages already available on M.O.R.S.E. for you. It is your job to edit some texts, add some minor styling, and link the web pages to M.O.R.S.E. with the puzzles. For the specifics on linking everything to M.O.R.S.E., you will need the admin manual which is also sent to you.

G.1. Escape event scenario

Giuseppe Mcdougall of your company has been fired because he leaked company secrets. The CEO is starting to panic because he was working on project X, which launches in a few days and the preparations have not yet been finished. We still need some cameras for the event. Those still have to be ordered! We don't know what camera it should be and only Giuseppe has this information in his mailbox. We've asked him politely to forward the mail, but he refuses to cooperate. Luckily we have the company hacker with us today... Good luck!

G.2. What to make:

G.2.1. Roles:

- Hacker
- Shopper

G.2.2. Mailbox domain: (Access: Hacker role)

This is the mailbox where the player will find information on what and how many cameras to buy.

TODO:

- You have to put the emails at the bottom of this document in the mailbox web page.
- Give only the hacker role access to this page.

G.2.3. Webshop domain: (Access: Everyone)

The webshop domain has a home page. On this page, the player has to select a camera to buy. Depending on the selected camera, they are redirected to the checkout page for that specific camera. When they ordered the camera, they will go to the order placed page.

TODO:

- Home page with several cameras that can be 'ordered'. This page has been made for you.
- We have created one checkout page for a camera for you.
 - You have to create two more checkout pages for the other two cameras.

- ◊ Those two pages should have the same puzzles as the already existing checkout page.
- ◊ You can export the already existing page and import it for the new pages. You will be able to find how in the admin manual.
- ◊ You have to link the created puzzles in the **Page Builder** to the correct elements (input and button).
- ◊ You have to create a new rule in the **Ruleset** that redirects the players from the camera 2 and 3 pages to the order placed page.
- ◊ (*optional*): If you have time left: Link the logo to the home trigger, that is already made for you.
- We have created one trigger for you in the **Rulesets** that redirects you to the first camera checkout page when you click the first camera.
 - You have to create two similar triggers that have a different name and that go to the other two pages.
 - You have to link those two triggers to the other two images in the home page. This can be done in the **Page Builder**.
- Give every role access to this page.

G.2.4. Leaderboard stage

When the player has ordered any camera, their MORSE screen will redirect to the leaderboard. Depending on what (*optional*): and how many) cameras they bought they will go a different leaderboard with different texts.

TODO:

- Create two 'leaderboard stages', one for players who ordered the correct cameras, and one for the players that didn't.
 - Add a custom text for the winners and losers.
- Based on the puzzle that was solved, redirect the player to the 'Win' or 'Lose' leaderboard stage. This can be done in the **Rulesets**. More information on this in the admin manual. [Note: for the basic version of the leaderboards (correct or incorrect camera) you do not need the *condition* in the admin manual example.]
 - Hint: you can add an action the already existing rules for redirecting the players to the order placed screen. Decide which rulesets need to go to which leaderboard.
- (*optional*): Add a third leaderboard stage for players who ordered the wrong amount of cameras but did select the correct camera type.
 - Add a custom text for that leaderboard.
 - Add a new *condition* in the **Rulesets** of the already existing rule for when Camera 1 puzzle is solved. (the answer of puzzle: 'Camera 1 puzzle' 'equals' 4)
 - Add a new *condition* in the **Rulesets** that is a copy of the previous one but now with condition: (the answer of puzzle: 'Camera 1 puzzle' 'does not equal' 4)

G.3. Email texts

G.3.1. Email 1:

Subject: Project X

From Borge Refsnes, Jun 10, 2020

Hello Guiseppe,

I just wanted to let you know that we will be needing 4 cameras of type Canon ZX350. We have a budget of \$3000 dollars so make sure you use this website because it is the cheapest: www.camerawebsho
For the rest of the project launch, we have covered all the remaining tasks. Thanks a lot for the help!

Best Regards,
 Borge Refsnes

G.3.2. Email 2:

Subject: You forgot your lunch!
From Jane Mcdougall, Jun 2, 2020.

Hi Seppy,

You seem to have forgotten your lunch box this morning! I can bring it to your office if you want?

Love,
 Mom

G.3.3. Email 3:

Subject: Super Special Sexy Sunglasses discount! Don't miss your chance!
From Anthony Alves, May 25, 2020.

Hi Guiseppe Mcdougall,

Thank you so much for being a customer of Sexy Sunglasses.

It's because of people like you we have been able to be in business for such a long time. To thank you, we have created a discount coupon especially for you.

Use the code DISCOUNT2020 to get a discount of 50% on any product in our store <http://www.sexysunglasses.com/>.

But hurry! The offer is only available for the first 100 people who make the purchase.

Thank you,

Anthony Alves of the Sexy Sunglasses support team

G.4. Playtest results

During the webshop playtest, we had participants who were using M.O.R.S.E. for the very first time. Performing the tasks given to them, required a lot of effort and help from us. We could relate and easily understand since not too long ago we were just as unfamiliar with the product and trying to separate our left from rights.

They didn't have the way of thinking required to use M.O.R.S.E. yet. First time use is very difficult if you're not used to the workflow. They did find it use full that there is an admin manual. The imported templates were very easy to use, but they would still want some templates already in M.O.R.S.E. that are simple to use and would expect to be less error prone. Linking all the things is a lot and difficult, since there are: schedule items, pages, triggers, rulesets, puzzles, roles. They also missed the ability to copy schedule items and rulesets, but that was already in the old system. It also wasn't very clear how the roles worked with the existing users.

Below is a list of new requirements filtered from all the feedback given from the participants combined with our observations. We checked those which we did finish, while those unchecked are not done due to time constraints or the problem being solved another way.

- Add a 'Save as' confirm dialog.
- Confirm 'Unsaved changes'.
- Dragging of schedule items did not work consequently.
- If you have a trigger on a puzzle submit button then the puzzle still has to be solved on click.
- Another way for setting a domain name than having 'domain' as default.
- Add an intuitive way of recognising what player role codes are meant for (and where they come from).
- A duplicate button for schedule items.
- A method to open pages in the page builder directly from the schedule editor.
- Unlink puzzles in the page builder.
- Do not use double-click in the page builder to edit text.

- Insert 'Delete Element' button.
- View and edit multiple rulesets at once.



DigiHacked Playtest

You will be following a scenario of creating an online escape event called DigiHacked. We will be providing you with the link to the admin manual, the link to M.O.R.S.E. as well as a login code.

H.1. Creating Creedo Industries

Link: <https://creedoindustries.com/>

- A domain for Creedo Industries, containing:
 - A home page.
- A page visit puzzle on the home page of Creedo Industries.

H.2. Creating ARealBaank

Link: <https://arealbaank.nl/login.php>

- A domain for arealbaank, containing:
 - A home page.
 - A login page.
 - A verification page.
 - A transaction page.
- Puzzles
 - A login puzzle on the login page of arealbaank using a form puzzle.
 - A page visit puzzle on the home page of arealbaank.
- Navigation triggers
 - A redirect from the home page of arealbaank to the login page.
 - A redirect from the login page of arealbaank to the home page.
 - A redirect from the login page of arealbaank to the logged in page once the previously created form puzzle is solved.

H.3. Creating Sixtrix

Pdf: https://localcdn.popup-escape.nl/Company_details_form.pdf

- A domain for Sixtrix, containing:
 - A home page.

- ◊ 4 buttons: email, chat, explorer, settings.
- An email page.
 - ◊ Template url: https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_templates_mail&stacked=h
- Puzzles
 - A login puzzle on the login page of arealbaank using a form puzzle.
 - A page visit puzzle on the home page of arealbaank.
- Navigation triggers
 - A redirect from the home page explorer button of Sixtrix to the pdf file.
 - A redirect from the home page of Sitrix to the email page.

H.4. Creating leaderboards

Since recreating the DigiHacked endscreen where the result of all teams is viewable is not possible in M.O.R.S.E., we want you to create an alternative that at least give team-specific customization.

Please make the following in in M.O.R.S.E.:

- A leaderboard stage for all teams who won.
 - Add a custom text to congratulate the winning teams.
- A leaderboard stage for all teams who lost.
 - Add a custom text to console the losing teams.
- A redirect to the winner leaderboard if a team solved all puzzles.
- A redirect to the loser leaderboard if a team did [insert condition to your liking].

H.5. Playtest results

In general, the volunteer was very happy with the product. Not everything was clear at first, but easy to use once explained. Importing was very nice, leaving room for personal creations as well as having the option to import from other page-builders.

Below is a list of new requirements filtered from all the feedback given from the participants combined with our observations. We checked those which we did finish, while those unchecked are not done due to time constraints or the problem being solved another way.

- The use of the domain names that the user has to enter in the domain schedule item is unclear.
- The admin manual lacks a description of custom blocks that can be inserted in the page builder.
- The admin manual lacks a description of hotkeys.
- The confirmation of actions is a bit camouflaged.
- Many actions are not clear at first, but clear after having them explained once.
- Bootstrap containers have a max-width, making it difficult to make the width bigger.
- Creating conditions in a ruleset with form puzzles was unclear since multiple answers are given and all existing rules are created for a single answer puzzle.
- Finding and extracting entire HTML and CSS blocks from existing sites that need to be imported is quite difficult and the approach is different for every browser.
- Imported pictures have a relative position and not absolute, therefore, correctly displaying on the site but not on the preview in the page builder.
- Testing your existing event on one device using an incognito window is unclear in the admin manual.
- Creating rulesets for separate leaderboards was confusing. This could be improved by adding a new puzzle status for when a team answers a puzzle with a wrong answer, an 'incorrect' puzzle status.



Infosheet

Enabling the creation and hosting of cooperative online escape events in M.O.R.S.E. without programming experience

Client

J.W. Manenschijn, BSc
CTO of Raccoon Serious Games

Coach

T.A.R. Overklift Vaupel Klein, ir.
Supervisor, TU Delft

Contact

Bram Verboom
bramv44@gmail.com

Description

To create and manage large escape events, Raccoon Serious Games created a system called M.O.R.S.E. This system was developed to locally run escape events and therefore has very limited support for creating online escape events. Creating online escape events could be done outside of the M.O.R.S.E. system, but that would require a lot of programming. Raccoon Serious Games does not have the required technical capacity for this, making it a very hard and time-consuming process.

To solve this problem, we added new features to the existing M.O.R.S.E. system, such that professional-looking online escape events could be made with no to minimal programming experience. During the research period, we tried to come up with a good solution for all the problems that are included with adding online escape events. We learned a lot about the privacy constraints around user tracking in this research phase and were, therefore, able to come up with a plan for hosting web pages as well as the user tracking.

The setup of the project was not as easy as expected. Making the old system run took a few days because of limited documentation. Afterwards, the CI and the ESLint static analysis still had to be set up as well. These issues took a big part of the time in the first development week.

The product that we created contains a page builder in which a web page can be built, styled and linked to puzzles and the rulesets in M.O.R.S.E. using a GUI and drag-and-drop. Web pages can also be imported and exported in the page builder. We also created domains on which those web pages can be placed. Those domains are run on the M.O.R.S.E. server and have to serve on external domains with the delivered iFrame. In addition to that, we also added leaderboards and roles, to enforce teamwork in the online experiences.

Recommendations for future development of this product are refactoring of the code of the old M.O.R.S.E. system, improved user code distribution and more role options. Raccoon Serious Games will likely use the product to create online escape rooms.

Team members

Elwin Duinkerken

*Lead testing, lead code documentation, **page builder developer***
Likes programming, as well as computer and board games.

Gijs Groenewegen

*Lead programmer, lead tooling, **page builder developer, styling***
Likes sushi and pizza. Enjoys programming and playing games.

Wessel Thomas

*Team leader, lead process documentation, **leaderboard developer***
Likes football and plays the bass guitar.

Bram Verboom

*External communicator, lead code quality, **domains developer***
Plays guitar and likes programming.

Timo Verlaan

*Scrum master and secretary, **role developer, hosting***
Enjoys listening and making music, programming and gaming.

All team members enjoy escape rooms and contributed to preparing the reports and the final project presentation


TU Delft

raccoon
serious games

Bibliography

- [1] Raccoon Serious Games, “Over ons.” Available at <https://raccoon.games/over-ons/>.
- [2] U. Ritterfeld, M. Cody, and P. Vorderer, *Serious games: Mechanisms and effects*. Routledge, 2009.
- [3] E. Boeve, L. Barfield, and S. Pemberton, “Wysiwyg editors: And what now?,” in *Human-Computer Interaction* (L. J. Bass, J. Gornostaev, and C. Unger, eds.), (Berlin, Heidelberg), pp. 68–82, Springer Berlin Heidelberg, 1993.
- [4] M. Bakker, A. Braam, W. Morssink, T. Nederveen, and A. Sterk, “Supporting large-scale escape rooms with a modular system,” 2019.
- [5] S. S. R. Ahamed, “Studying the feasibility and importance of software testing: An analysis,” *CoRR*, vol. abs/1001.4193, 2010.
- [6] Scrum, “What is scrum?.” Available at <https://www.scrum.org/resources/what-is-scrum>.
- [7] World Wide Web Consortium, “Media queries.” Available at <https://www.w3.org/TR/css3-mediaqueries/>.
- [8] Bootstrap. Available at <https://getbootstrap.com/>.
- [9] Grapesjs. Available at <https://grapesjs.com/>.
- [10] S. Douglas, E. Doerry, and D. Novick, “Quick: a tool for graphical user-interface construction by non-programmers,” *The Visual Computer*, vol. 8, pp. 117–133, 03 1992.
- [11] GridstackJS. Available at <https://gridstackjs.com/>.
- [12] jQueryUI. Available at <https://jqueryui.com/>.
- [13] P. Leon, B. Ur, R. Shay, Y. Wang, R. Balebako, and L. Cranor, “Why johnny can’t opt out: A usability evaluation of tools to limit online behavioral advertising,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’12, (New York, NY, USA), p. 589–598, Association for Computing Machinery, 2012.
- [14] Google. <https://support.google.com/analytics/answer/1034342?hl=en>.
- [15] Mozilla, “Http cookies.” <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>.
- [16] S. J. Murdoch, “Hardened stateless session cookies,” in *Security Protocols XVI* (B. Christianson, J. A. Malcolm, V. Matyas, and M. Roe, eds.), (Berlin, Heidelberg), pp. 93–101, Springer Berlin Heidelberg, 2011.
- [17] M. E. Chalandar, P. Darvish, and A. M. Rahmani, “A centralized cookie-based single sign-on in distributed systems,” in *2007 ITI 5th International Conference on Information and Communications Technology*, pp. 163–165, 2007.
- [18] Contently, “xdomain-cookies.” <https://github.com/contently/xdomain-cookies>.
- [19] NameCheap, “Types of domain redirects - 301, 302 url redirects, url frame (and cname).” Available at <https://www.namecheap.com/support/knowledgebase/article.aspx/9604/2237/types-of-domain-redirects--301-302-url-redirects-url-frame-and-cname>, 2019.
- [20] W3schools, “Html <iframe> tag.” https://www.w3schools.com/tags/tag_iframe.asp.
- [21] J. Read, “What is an ecu? cpu benchmarking in the cloud.” Available at <http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html>, 2010.