EGA Membership Card System

D. Bridié
P.C.F. van der Knaap
J.T. van Schagen

EGA Membership Card System

D. Bridié
P.C.F. van der Knaap
J.T. van Schagen

Bachelor Thesis
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
July 7, 2017

Preface

This document is the report for the bachelor thesis project created by Dereck Bridie, Paul van der Knaap, and Jurgen van Schagen as the last part of the Bachelor Computer Science at the Delft University of Technology. The goal of the bachelor project is to demonstrate that students can successfully and independently carry out a software project in a professional context, working with a real client.

This report describes the work done and the product delivered to complete the project as commissioned by the E-Sports Game Arena. The project is aimed to investigate how a membership card system can be introduced to enhance user experience at their new venue: the Arena.

We would like to thank Claudia Hauff for her support as our TU Delft Coach and for the effort she has put into our project by providing valuable input on our process. Furthermore, we would like to thank to thank our client Thomas Runhart for providing us with this challenging project.

Summary

The E-Sports Game Arena will launch an esports cafe in the summer of 2017. At this location, the company wants to introduce membership cards to create a sense of community. The goal of the project was to explore the technological possibilities of such cards and to create a working prototype that makes use of these possibilities.

At the start of the project the assignment was still broad: explore the possibilities of a card system at the Arena. The first goal was to set a clear scope for the project by brainstorming with the client. After the brainstorm sessions, some ideas were selected while keeping factors like allotted time and technical practicality in mind. From these ideas, requirements were created to establish the essential functionalities necessary for the system. User scenarios were created to formulate the requirements in real world use cases to get a better understanding of the situations in which the system is used.

Following the design phase, the technical specification was constructed. Together with our client we decided that creating a proper software foundation for the card system was vital to this project, as it would later be expanded with other features and systems. Additionally, we created four prototype applications that make use of this foundation.

During the project, we analyzed the security aspects in detail by listing risks, possible attacks, and solutions that could be used to mitigate such attacks. For security solutions that were not implemented during the course of the project, we made recommendations as a guideline for future improvements.

This report also details the testing process used in this project, describing the use of testing frameworks and approaches. Then, the proceedings of the project with respect to its group members, the coach, the client, and the Software Improvement Group are evaluated. Afterward, the product as a whole is evaluated in retrospect, taking the project requirements into account.

Finally, a conclusion is made, evaluating the success of the project and giving our final recommendations.

Contents

1. Introduction

In recent years, competitive video games have become increasingly popular, resulting in a new form of competition through video games. According to esports market intelligence bureau Newzoo, esports revenue is estimated to reach $696 million in 2017 and is estimated to grow to $1.5 billion by 2020 [36]. For example, in 2016 the popular game 'League of Legends' managed to attract 43 million unique viewers through online media, and sported a total prize pool of $6.7 million [18].

The E-Sports Game Arena (EGA) is a new company currently in the process of opening an esports location in the Netherlands, called 'the Arena'. The Arena will host official esports tournaments and will provide a location for members to play video games together.

The Arena will also house a bar for refreshments, a podium, computers, video game consoles and other gaming related activities.

One of the goals of EGA is to establish a community where esports enthusiasts can gather. To promote a sense of community, EGA would like to use personal membership cards. The goal of the project was to explore the technological possibilities of such a card and to create working prototypes which makes use of these possibilities. This is challenging because the client had very specific purposes in mind and wanted an extensible system for the future.

To facilitate the use of such membership cards, a software framework must be made which enables interaction with digital systems at the Arena.

2. Product Definition

During the first week of the project, the primary goal was to inventorize the client's needs and define the goals of the project. This started with specifying the client's requests through meetings, leading to the product specification. Then, the design goals were described that guided vital decisions made in the product's design. Finally, the success criteria which need to be fulfilled to successfully complete the project were set.

2.1 Product Specification

At the start of the project, the only available information was the original project description on BEPsys (see appendix A: *Problem Description BEPsys*). The project description provided a clear overview of the high-level project goal: implementing a card system for EGA's members. However, from just the original project's description, it was not clear what features the client expected.

During an initial brainstorm meeting with the client in the first week, many possible functionalities of the product were discussed, resulting in a large list of interesting applications for the membership card. After feedback from our coach, this list was reduced to fit in a ten week timeframe.

These functionalities have been separated in four separate systems in the following sections. Each section starts with a problem description, then describes an idea for an implementation by using a diagram. This is then formalized in a feature description and a list of functional requirements.

2.1.1 Check-in System

The EGA would like to have data on the frequency of member visits. This data could, for example, be used to provide rewards for frequent visitors.

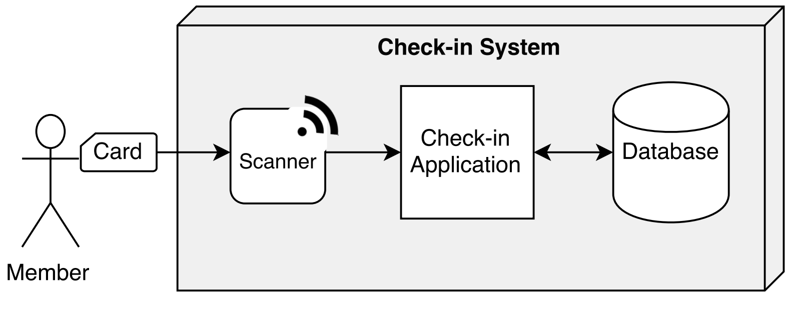


Figure 2.1: Initial concept of the Check-in System.

When a member enters the arena, he or she can swipe their membership card at the entrance to log his or her presence. Similarly, the member can swipe the card to exit the building. However, we will not write software to analyze this data; this is left as future work.

To create this system, the following functionalities are required:

- The system must be able to detect when a card is swiped.
- The system must be able to distinguish between a user entering the building or exiting the building.
- The system must be able to save these events into a database.

2.1.2 Console Locking System

The Arena is open for entry at any time for free. However, using EGA's gaming consoles requires an active subscription to the Arena, meaning that non-members should not be able to use them.

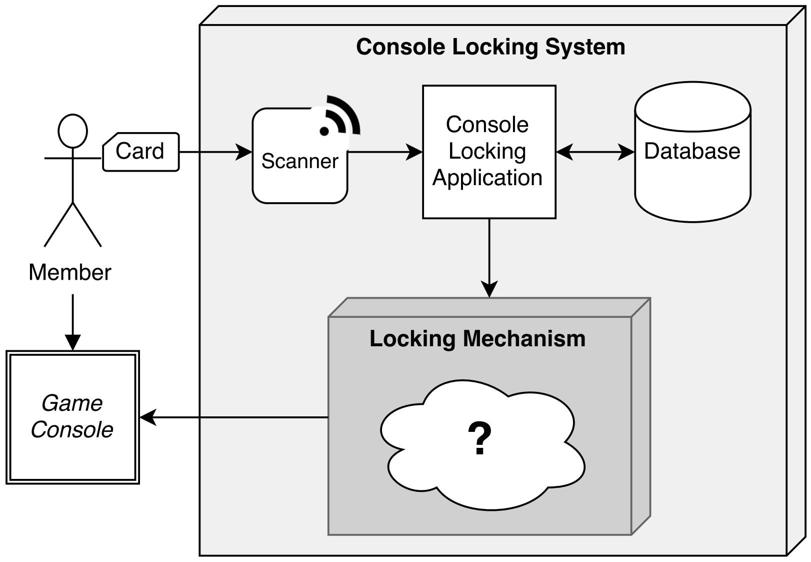


Figure 2.2: Initial concept of the Console Locking System.

To prevent non-members from accessing the gaming consoles, a valid membership card must be used to unlock the console. After the member is finished playing, the card can be used again to lock the console.

To create this system, the following functionalities are required:

- Gaming consoles must be able to be locked.
- Gaming consoles must be able to be unlocked.
- The system must be able to detect when a card is swiped.
- The system must be able to check the validity of a membership card.

2.1.3 Payment System

Besides hosting various gaming activities, the Arena will contain a bar where members can purchase refreshments.

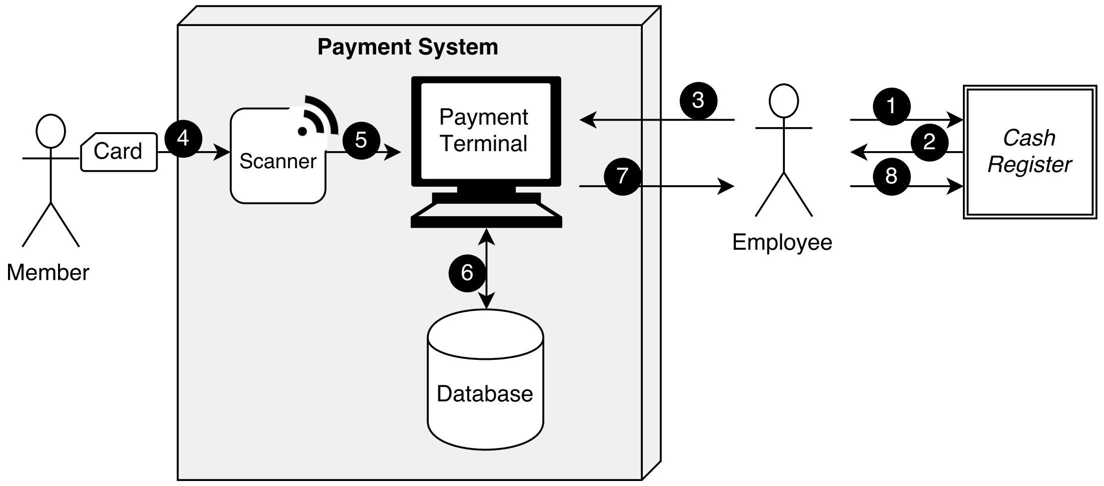


Figure 2.3: Initial concept of the Payment System.

At the bar, the members should be able to use their membership card to make purchases as an alternative to using cash or a bank card. In the future, this would enable EGA to give special discounts to members that frequently make purchases.

To purchase items, members must have a balance assigned to their account. This balance can be assigned by an employee in the administration system (see section 2.1.4). Per request of the client, the balance is allowed to be negative. The reasoning for this can be found in chapter 9: *Ethical Evaluation.*

Figure 2.3 shows the initial concept of the Payment System. To pay for refreshments, the member can choose to use the membership card as payment method. After the employee takes the order from the member **(1)** and calculates the total cost of the order using the existing cash register **(2)**, the employee enters the amount the member has to pay in the payment terminal **(3)**. Then, the member is asked to swipe his card **(4)** over the scanner to complete the payment **(5)**. The transaction is processed by the payment terminal which uses a database **(6)**. The Payment System will show the result of the transaction to the employee **(7)** who can mark the payment as completed in the cash register **(8)**.

A final request from the client was that it should be usable on a tablet device.

To conclude, to create this system, the following functionalities are required:

- Users must be able to have an account balance.
- Users must be able to have an account balance limit.
- The system must be able to detect when a card is swiped.
- The system must be able to check the validity of a membership card.
- The system must be able to check if the user's balance is sufficient for a transaction.
- The system must be able to deduct balance from a user's account.
- The system must be able to log all events which alter the user's balance.

- The system must be able to reply with one of the following responses:
  1. the membership card has been blocked;
  2. the membership card is invalid;
  3. the membership card is valid, but is not linked with an account enabled for payments;
  4. the membership card is valid and allows for payments, but the amount billed exceeds the member's limit;
  5. the membership card is valid, allows for payments, and would not be billed over their limit.
- The system must be able to display the transaction's result.
- The system must be usable on a tablet device.

2.1.4 Administration System

The administrators should be able to manage all the users' data. Employees must be able to use a private portal where user information can be viewed and edited.

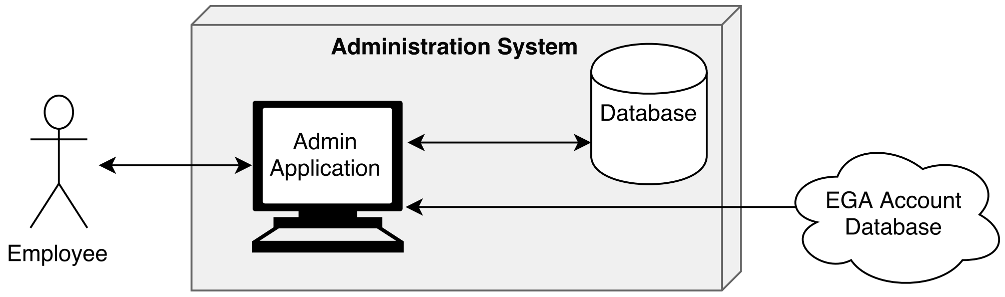


Figure 2.4: Initial concept of the Payment System.

An administrator should be able to assign a card to a customer, by using an existing member profile or allowing the customer to register if necessary. EGA already has a database which contains personal information about the members and would like the card system to work in cooperation with the existing database. This means that an account in the card system should be linked to the user account on the EGA database.

A customer should be able to choose one of the following options:

1. The customer does not want to use the card for payments. In this case the card is blocked from making payments, but works for all systems that require the card as authentication method.
2. The customer wants to opt-in to use this card as payment method for refreshments at the bar.

When a user loses their card, system administrators must be able to block them to prevent abuse. This is to prevent others from using it to make unauthorized payments. Members should ask an employee to block their card.

The treasurer of EGA should be able to access the administration system and should be able to see a list of all members' balances.

Finally, all actions which alter the balance should be logged in the system. Editing the balance as administrator or paying at the bar should always be stored in the database since transactions should never be lost.

To conclude, to create this system, the following functionalities are required:

1. Administrators must be able to assign membership cards to a (new) user using the existing EGA account database.
2. Administrators must be able to block or unblock membership cards.
3. Administrators must be able to allow or disallow payments for membership cards.
4. Administrators must be able to adjust member's balances.
5. Administrators must be able to view all membership cards in the system.
6. Administrators must be able to view all transactions processed in the system.
7. Administrators must be able to generate an overview of the balance of all users.

2.2 Design Goals

Design goals provide a design philosophy for developers that help in making decisions during the product's design and implementation phases. The membership card system will focus on six design goals: security, reliability, availability, maintainability, performance, and usability. Each design goal is briefly described and an explanation is given as to why it is important.

2.2.1 Security

The membership card system can authenticate members and allows members to make financial transactions using their membership card. When dealing with financial transactions it is crucial that transactions and user balance can not be manipulated. Additionally, it is important to prevent unauthorized access to the membership card system and administration system.

2.2.2 Reliability

When handling financial data, it is absolutely essential that no single transaction is lost. Under no circumstances should it be possible for a member to receive a product without actually paying for it. Similarly, the member should never be billed more than once for a single product. In case of a system failure, systems should be able to recover without any loss of data.

2.2.3 Availability

Since the card system will provide a vital part of the Arena's services, the systems should operate without any downtime, as downtime will impact the revenue of the Arena negatively.

2.2.4 Maintainability

The membership card system provides a framework upon which more functionality can be added as new ideas emerge. This means that the products should be extendable and easy to maintain. Adding new functionality should be straightforward for future developers.

2.2.5 Performance

One of the concerns when creating an application is that it can perform well under heavy load. When an employee or EGA member interacts with the system it is expected to work smoothly.

2.2.6 Usability

It is important that the system is easy to use. For EGA members it should be intuitive to use their membership cards as a way to interact with the systems at the Arena. Additionally, the employees should be able to use the management software with only a quick walkthrough of the system.

2.3 Project Success Criteria

One of the success criteria defined by the client is that a prototype payment system and administration system must be completed. Additionally, the client also emphasized that the membership cards must be linked to users in the existing EGA account database.

Although the client has expressed interest in the check-in system and console locking system, they are optional for the project to be a success as they were deemed less important.

3. Design

This chapter contains a complete overview of the system design for the product and discusses the design choices made in the process.

In section 2.1: *Product Specification*, the project has already been split up in four different systems: the Payment System, Administration System, Check-in System and Console Locking System. The parts of these systems that users interact with are defined as *applications* in our design model.

One of the design goals which was vital in designing the product structure was maintainability. An important part of this goal was to make it straightforward for future developers to add new functionalities to the membership card system, without needing prior knowledge of the entire project. To facilitate this, *APIs* were created. APIs provide a consistent way of interacting with our system and can be accessed by multiple applications.

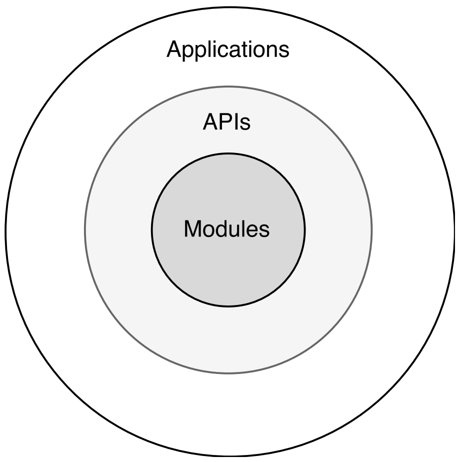


Figure 3.1: Layered product design.

Since different APIs can perform similar operations, *modules* have been introduced to prevent code duplication. These modules have their own responsibilities, which are discussed in section 3.2: *Modules*.

Using this hierarchy of software, a layered diagram can be created of the structure as can be seen in figure 3.1.

To summarize, applications rely on backend APIs, which incorporate different modules, and each module provides a small piece of functionality.

3.1 Technologies and Frameworks

The systems are implemented in node.js using TypeScript as the main programming language. The framework used for the APIs is Express.js. Communication between applications and APIs is done using HTTPS and JSON. Applications make use of either web technologies in the browser or the command line on a Raspberry Pi to run. The details of the decisions on the programming language and frameworks details are discussed in the Appendix C: *Technologies and Framework Choices*.

3.2 Modules

3.2.1 Token Manager

The Token Manager will be responsible for doing operations in relation to the unique identifiers (tokens) recorded on the membership cards. This module will provide a layer of abstraction for token information from the database.

3.2.2 Transaction Manager

The Transaction Manager will be responsible for doing operations in relation to transactions performed in the card system. Similar to the *Token Manager*, the Transaction Manager will provide a layer of abstraction for transaction information from the database.

3.2.3 User Manager

The User Manager will be responsible for doing operations in relation to users in the card system. The User Manager will also provide a layer of abstraction for user information from the database.

3.3 APIs

3.3.1 Authentication Server

The Authentication Server will provide an API which can be used to check the validity of a token. It will use the *Token Manager* to read data from the database. If a valid token has been supplied, the Authentication Server will return the user to which the token is assigned.

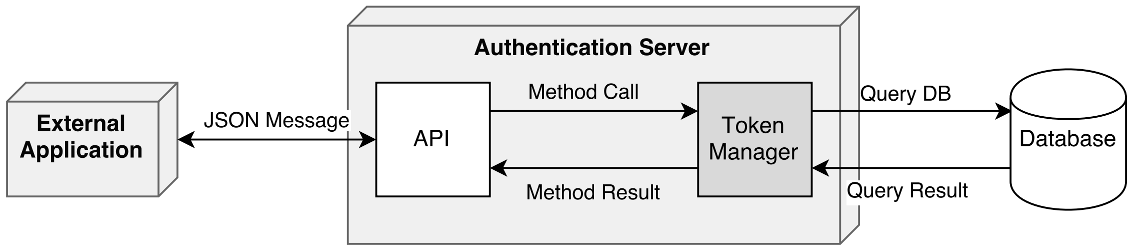


Figure 3.2: Internal infrastructure of the Authentication Server.

3.3.2 Payment Server

The Payment Server will provide an API for the *Payment Terminal Application* (3.4.3), but could also be used for other applications which require payments. It will provide the logic to handle transactions by making use of the *Transaction Manager* and the *Token Manager*.

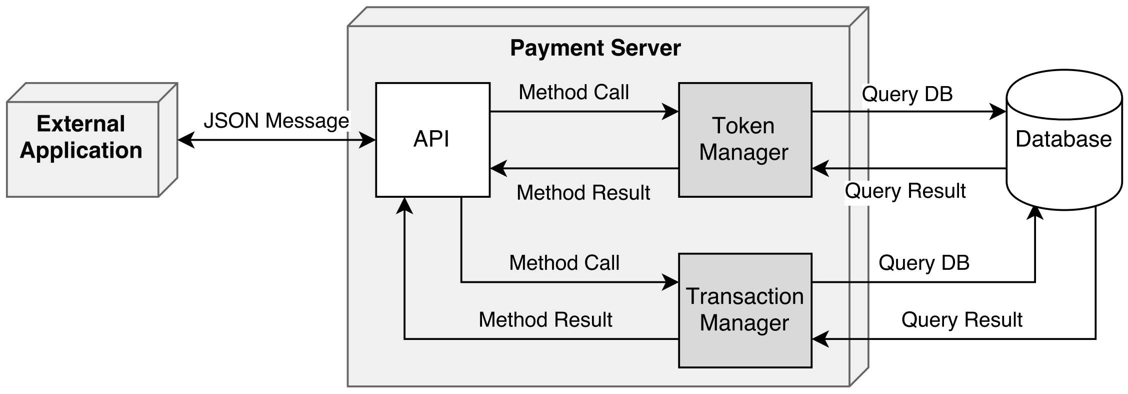


Figure 3.3: Internal infrastructure of the Payment Server.

3.4 Applications

3.4.1 Check-in Application

The Check-in Application will run on a Raspberry Pi and will register which cards were swiped by members entering and leaving the building. It will use the *Authentication Server* to verify swiped cards and will write these events into a database.

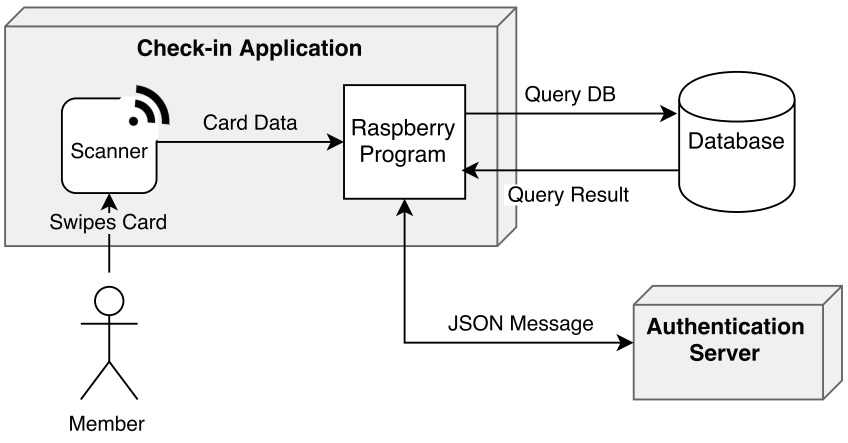


Figure 3.4: Internal infrastructure of the Check-in Application.

3.4.2 Console Locking Application

The Console Locking Application will run on a Raspberry Pi and will prevent non-members from using the gaming consoles at the location if there is no valid card swiped. It will use the *Authentication Server* to verify swiped cards. As locking mechanism, the application will control a power switch to toggle the access state to the console. Because turning the consoles on and off without properly shutting them down is bad for the hardware [53], the power to the television attached to the console will be controlled instead, leading to the same result of preventing and allowing console access.

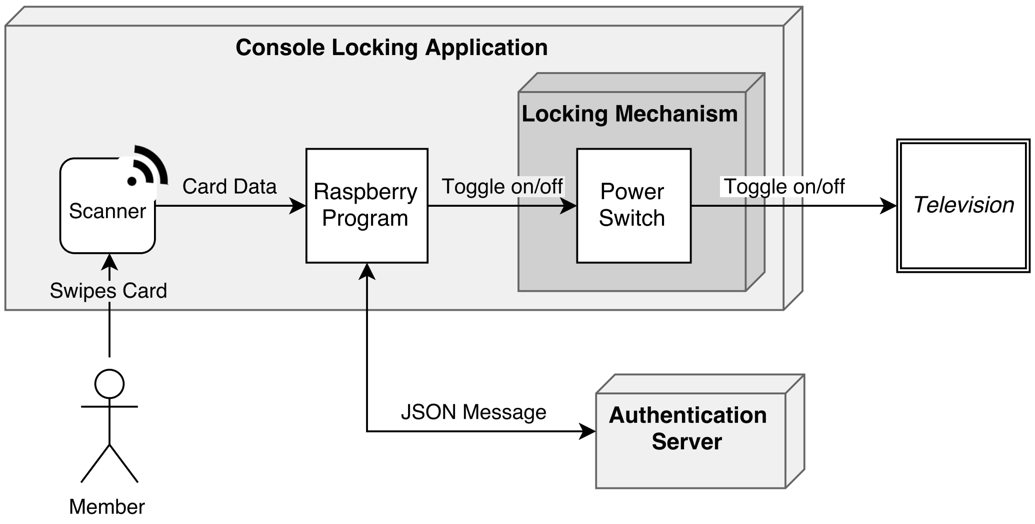


Figure 3.5: Internal infrastructure of the Console Locking Application.

3.4.3 Payment Terminal Application

The Payment Terminal Application will be a front-end application used by employees to enter bar transactions. It will allow employees to enter amounts and will let customers scan their cards to make a payment. It will use the *Payment Server* to process these requests.

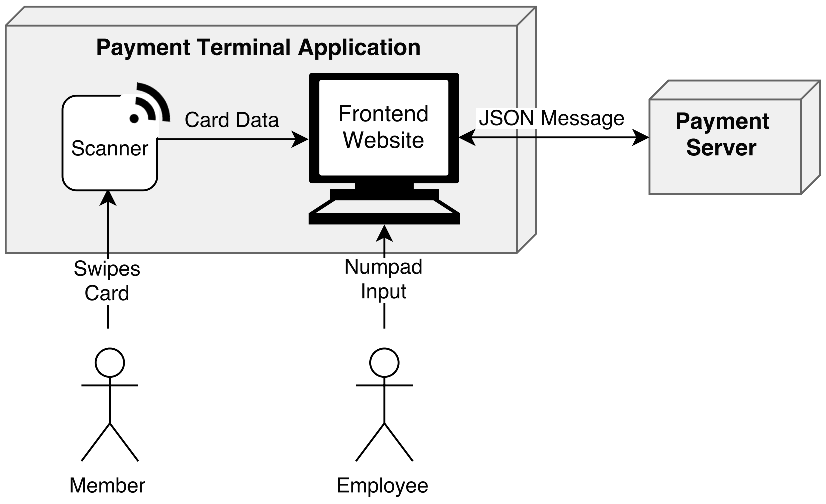


Figure 3.6: Internal infrastructure of the Payment Terminal Application.

3.4.4 Administration Application

The Administration Application will be a front-end web application for employees used to handle administrative functionality for the membership cards. In case of any outages, employees should still be able to manage the system, therefore the Administration Application will use its own API.

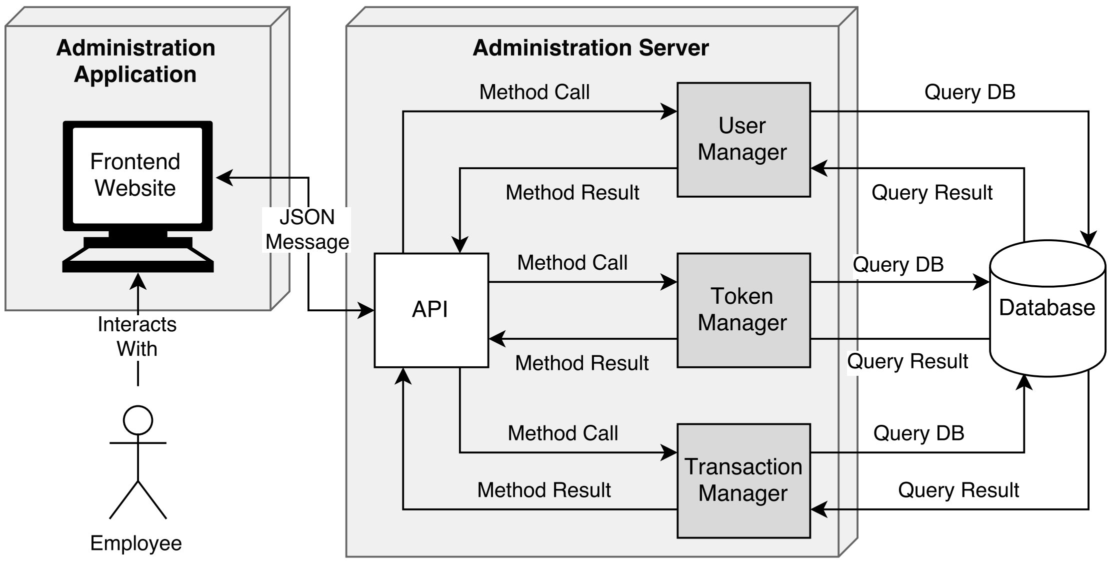


Figure 3.7: Internal infrastructure of the Payment Terminal Application.

4. Implementation

After the structure of the systems was defined, the development of the product was started based on the design. In this chapter, the implementation of each component will be elaborated upon. As in chapter 3: *Design*, the modules, servers and applications are discussed separately.

4.1 Modules

4.1.1 Token Manager

The Token Manager provides a data class *Token* which can be used by other systems. The definition of this data class can be found in figure 4.1.

```
Token {
    blocked: boolean;
    paymentsAllowed: boolean;
    token: string;
    userID: number;
}
```

Figure 4.1: Data class definition for Tokens.

The Token Manager can be used to execute the actions listed in table 4.2.

Method	**Description**
`create(`*token*`: `**`Token`**`): void`	Adds a new token to the database.
`unblock(`*token*`: `**`string`**`): void`	Unblocks the token, enabling the token to be used for authentication.
`block(`*token*`: `**`string`**`): void`	Blocks the token, disabling the token for authentication.
`allowPayments(`*token*`: `**`string`**`): void`	Enables transactions to be done with the card this token is assigned to.
`disallowPayments(`*token*`: `**`string`**`): void`	Disables transactions to be done with the card this token is assigned to.
`remove(`*token*`: `**`string`**`): void`	Removes the token from the database.
`get(`*token*`: `**`string`**`): `**`Token`**	Returns the **Token** information from the database.
`getAll(): `**`Token[]`**	Returns all existing **Tokens** from the database.
`getAllWhere(`*column*`: `**`string`**`, `*value*`: `**`any`**`): `**`Token[]`**	Returns all existing **Tokens**, where the database *column* is *equal* to the *value* parameter.

Figure 4.2: Token Manager actions.

4.1.2 Transaction Manager

The Transaction Manager provides a data class *Transaction* which can be used by other systems. The definition of this data class can be found in figure 4.3.

```
Transaction {
    amount: number;
    date: string;
    description: string;
    id: number;
    userID: number;
}
```

Figure 4.3: Data class definition for Transaction.

The Transaction Manager can be used to execute the actions listed in table 4.4.

Method	**Description**
`add(`*userid*`: `**`number`**`, `*amount*`: `**`number`**`, `*description*`: `**`string`**`): `**`void`**	`Adds a transaction to the user’s account, adjusting the user's balance.`
`getAllTransactions(): `**`Transaction[]`**	`Gets a list of all Transactions that exist in the database.`
`getUserTransactions(`*userid*`: `**`number`**`): `**`Transaction[]`**	`Gets a list of all Transactions that belong to a specific user.`

Figure 4.4: Transaction Manager actions.

4.1.3 User Manager

The User Manager provides a data class *Users* which can be used by other systems. The definition of this data class can be found in figure 4.5.

```
User {
    balance: number;
    balance_limit: number;
    name: string;
    userID: number;
}
```

Figure 4.5: Data class definition for User.

The User Manager can be used to execute the actions listed in table 4.6.

Method	**Description**
`newUser(`*name*`: `**`number`**`, `*egaID*`: `**`number`**`): `**`void`**	`Creates a new user in the database, which contains `*egaID*` as a link to the already existing account database.`
`getAll(): `**`User[]`**	`Gets a list of all `**`Users`**` that exist in the database.`
`getUser(`*id*`: `**`number`**`): `**`User`**	`Gets a specific `**`User`**` from the database.`
`putUser(`*user*`: `**`User`**`): `**`void`**	`Updates user’s information in the database.`

Figure 4.6: User manager actions.

4.2 APIs

4.2.1 Authentication Server

The Authentication Server replies whether a token is valid or not, and if it is, it will respond with the information about that token. For example:

`GET` */users/token/0711159812*	`{ "token_data": { "blocked": 0, "token": "0711159812", "userID": 2 }, "text": "Authorized!" }`
`GET` */users/token/0711159813*	`{ "token_data": null, "text": "The specified token cannot be found!" }`

4.2.2 Payment Server

The Payment Server will process a request for a payment. Example requests and responses:

`PUT /api/payments { "token": "0000000000", "amount": "2.50" }`	`STATUS 404 { "text": "Token could not be found!" }`
`PUT /api/payments { "token": "0711159811", "amount": "250.00" }`	`STATUS 403 { "text": "Transaction Failed: Not enough balance!" }`
`PUT /api/payments { "token": "0711159811", "amount": "2.50" }`	`STATUS 201 { "text": "Payment succeeded" }`

4.3 Applications

4.3.1 Check-in Application

The Check-in Application runs on a Raspberry Pi. The Pi requires an internet connection and automatically starts the Check-in Application after the Pi has been turned on. The scanner is connected to the Pi by USB. The devices can be seen in figure 4.7.

If this is to be used in a real-world setting, a case needs to be built to house these components. Since this case would be seen by end-users, it should look nice.

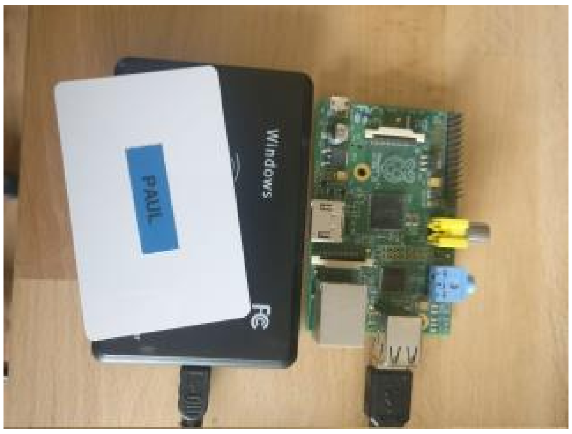


Figure 4.7: Raspberry Pi with RFID Scanner.

4.3.2 Console Locking Application

To create the Console Locking Application, we used another Raspberry Pi. Similar to the *Check-in Application*, it requires an internet connection and automatically starts the Console Locking Application. Figure 4.10 shows usage of the Console Locking Application.

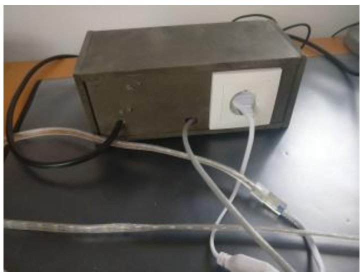
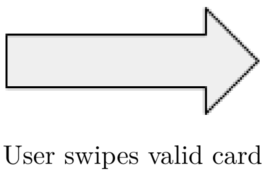


Figure 4.8: Turned off



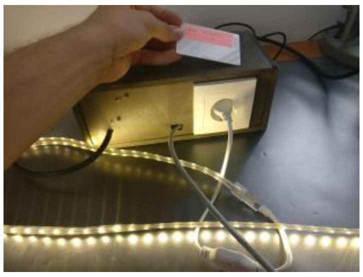


Figure 4.9: Turned on

Figure 4.10: The Console Locking Application in use.

For the prototype, we built a case to house the components: the Pi, scanner, and power switch. These components are shown in figure 4.11.

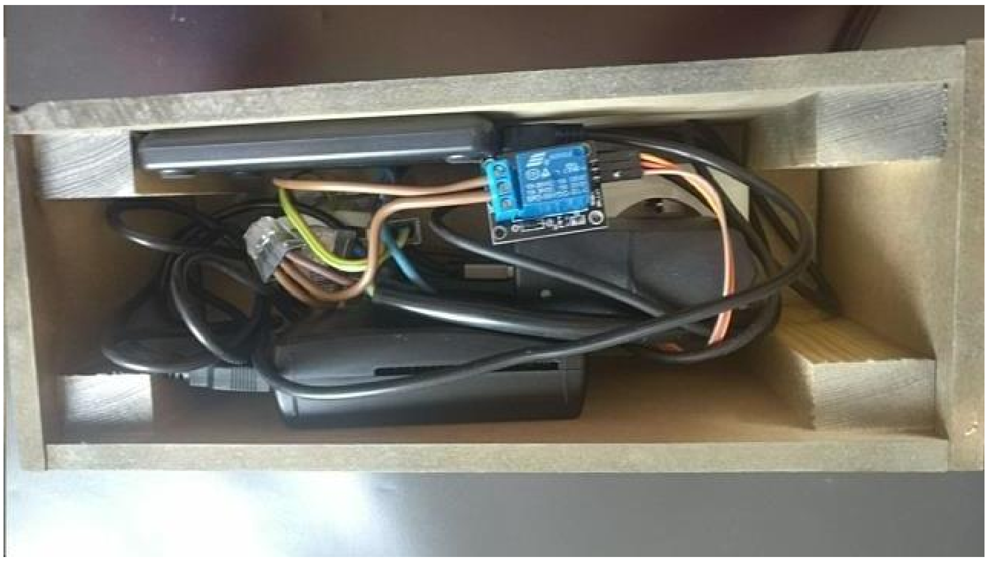


Figure 4.11: Inside the Console Locking Application case.

Power is supplied by a normal power cable and is split to supply power for the Pi and the external power socket. The external power socket is controlled by the 230 volt power switch which can be turned on or off by the Raspberry Pi. A schematic diagram of the mechanism can be seen in figure 4.12.

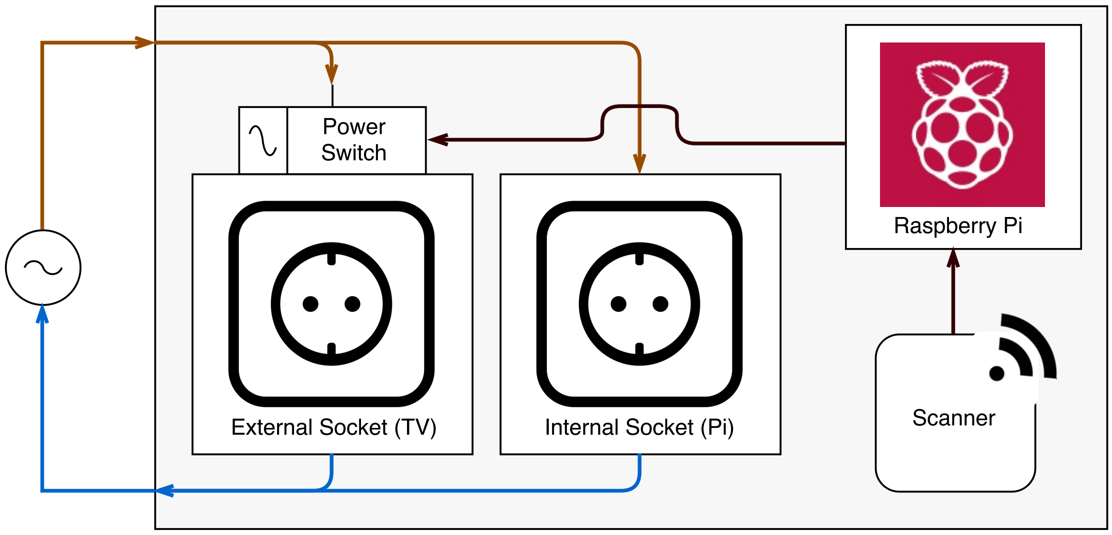


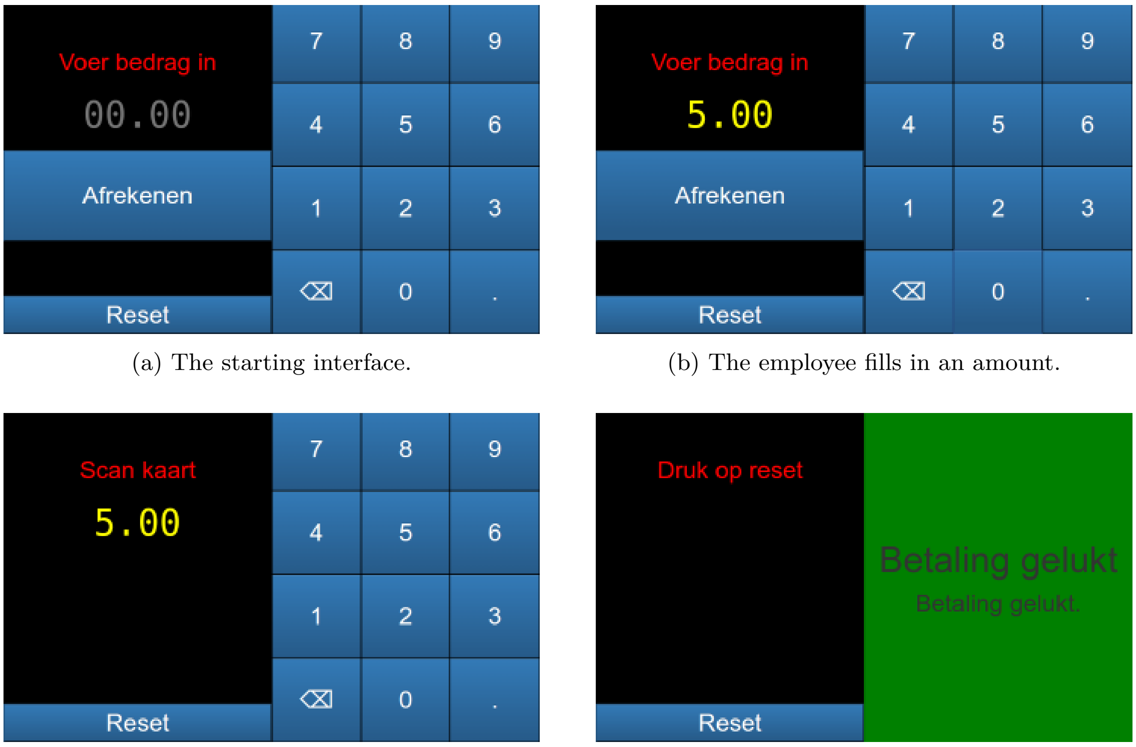
Figure 4.12: Schematic diagram of the Console Locking mechanism.

4.3.3 Payment Terminal Application

The Payment Terminal Application is used by employees to handle transactions. The user interface allows the employee to enter an amount on the on-screen numpad. After entering the amount the employee can press ‘Afrekenen’ which enables the member to scan their card.

After the member scans his or her card, the payment can either succeed or fail. When the payment succeeds, the member may receive his refreshments and the transaction is complete. In case the payment fails, the reason is shown to the employee. It is then possible to rescan a card to try again. Finally, there is a reset button, which will take the application back to its default state to allow a new payment.

The user interface was designed to be simple and intuitive for employees so little explanation is required before an employee can use the system. In addition, the system needed to be touch-friendly as the interface will run on a tablet (see section 2.1: *Product Specification*). The feedback from a real-world test can be found in section 6.5: *Real World Usability Testing*.



(a) The starting interface.

(b) The employee fills in an amount.

(c) The customer is asked to scan the card.

(d) The payment has been processed.

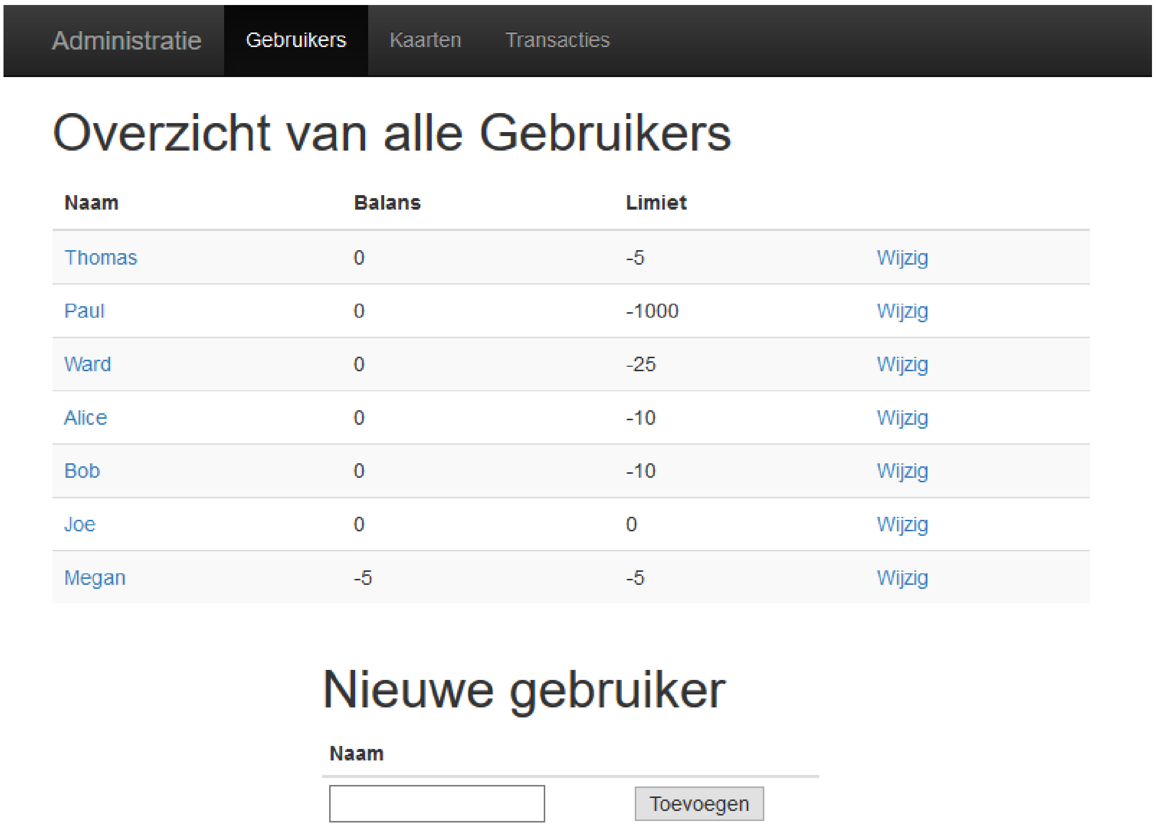
Figure 4.13: Overview of the Payment Terminal interface

4.3.4 Administration Application

The Administration Application contains three tabs: users, cards and transactions. The users section, as seen in figure 4.14, contains a list of all the users in the card system, their balance, and their balance limit. Each user can be examined in detail, as is shown in figure 4.15. The user details page shows information such as their balance, their balance limit, their cards and a history of their transactions. Additionally, employees can modify user information.

The cards section, as seen in figure 4.16, shows a list of all cards in the system and their current status. From this overview, the employee can click to see the details of the corresponding user.

Finally, the transactions section seen in figure 4.17 shows a history of all transactions.



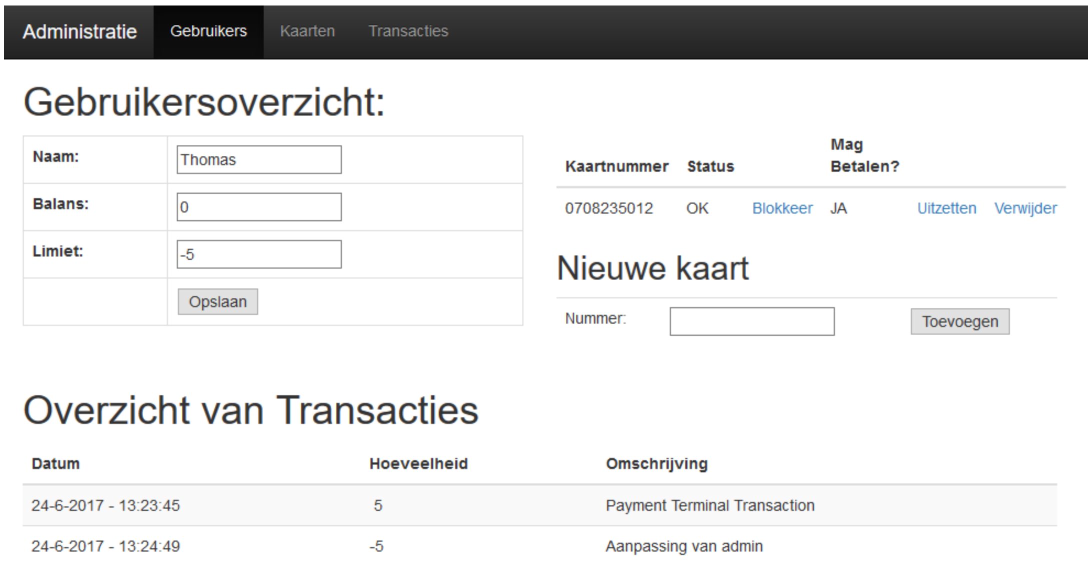
Overzicht van alle Gebruikers

Naam	Balans	Limiet	
Thomas	0	-5	Wijzig
Paul	0	-1000	Wijzig
Ward	0	-25	Wijzig
Alice	0	-10	Wijzig
Bob	0	-10	Wijzig
Joe	0	0	Wijzig
Megan	-5	-5	Wijzig

Nieuwe gebruiker

Naam

Figure 4.14: Interface of the users section in the administration system.



Gebruikersoverzicht:

Naam:	Thomas
Balans:	0
Limiet:	-5
	Opslaan

Kaartnummer	Status		Mag Betalen?		
0708235012	OK	Blokkeer	JA	Uitzetten	Verwijder

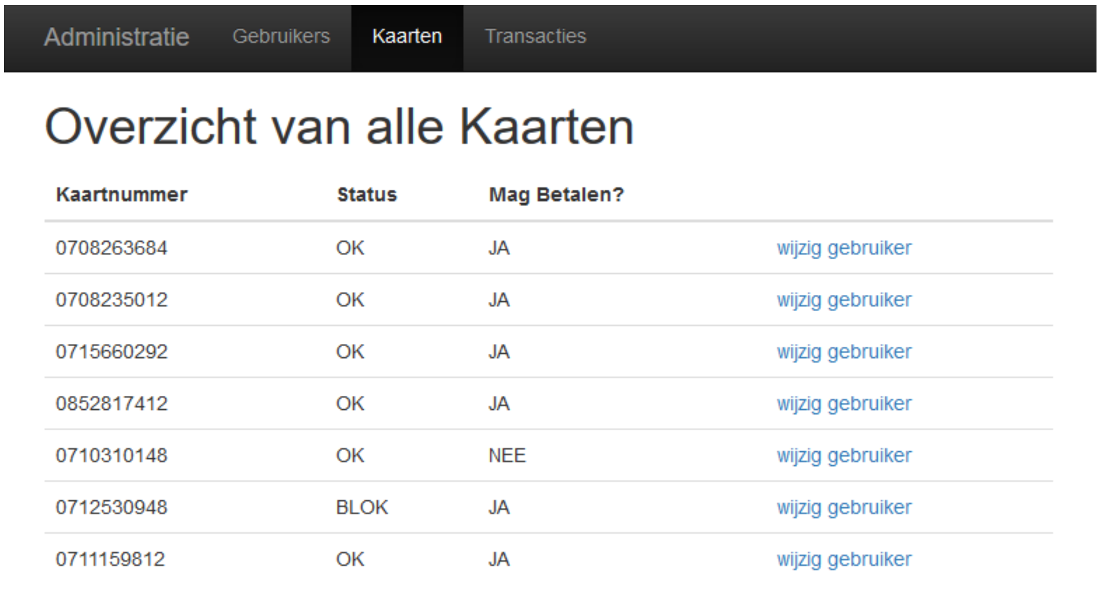
Nieuwe kaart

Nummer:

Overzicht van Transacties

Datum	Hoeveelheid	Omschrijving
24-6-2017 - 13:23:45	5	Payment Terminal Transaction
24-6-2017 - 13:24:49	-5	Aanpassing van admin

Figure 4.15: Interface of the user details section in the administration system.

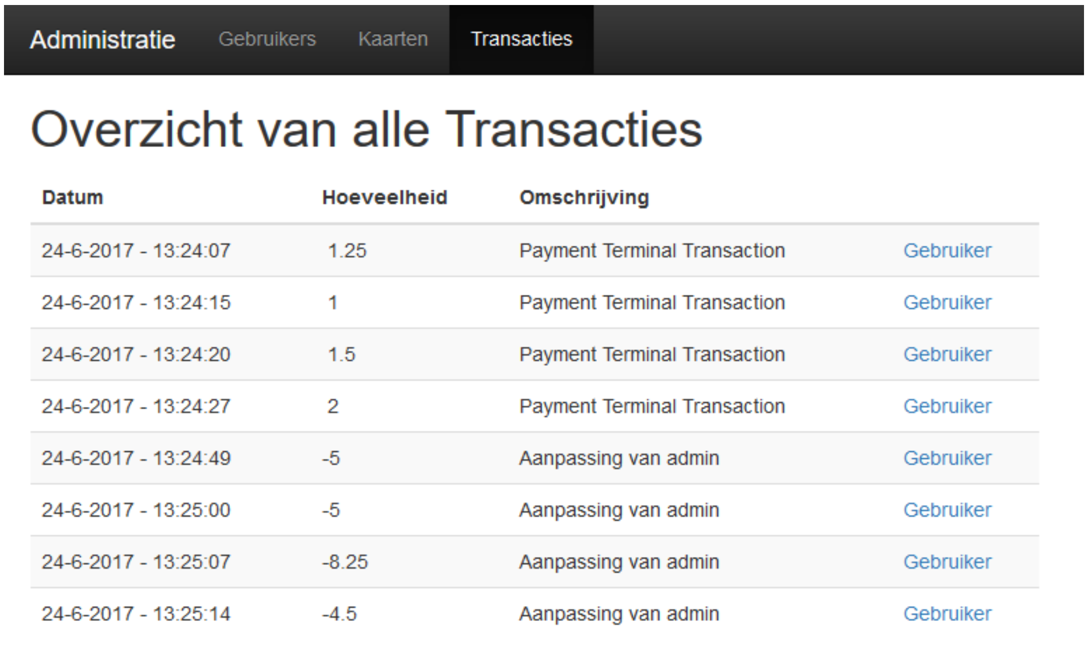


Administratie Gebruikers Kaarten Transacties

Overzicht van alle Kaarten

Kaartnummer	Status	Mag Betalen?	
0708263684	OK	JA	wijzig gebruiker
0708235012	OK	JA	wijzig gebruiker
0715660292	OK	JA	wijzig gebruiker
0852817412	OK	JA	wijzig gebruiker
0710310148	OK	NEE	wijzig gebruiker
0712530948	BLOK	JA	wijzig gebruiker
0711159812	OK	JA	wijzig gebruiker

Figure 4.16: Interface of the cards section in the administration system.



Administratie Gebruikers Kaarten Transacties

Overzicht van alle Transacties

Datum	Hoeveelheid	Omschrijving	
24-6-2017 - 13:24:07	1.25	Payment Terminal Transaction	Gebruiker
24-6-2017 - 13:24:15	1	Payment Terminal Transaction	Gebruiker
24-6-2017 - 13:24:20	1.5	Payment Terminal Transaction	Gebruiker
24-6-2017 - 13:24:27	2	Payment Terminal Transaction	Gebruiker
24-6-2017 - 13:24:49	-5	Aanpassing van admin	Gebruiker
24-6-2017 - 13:25:00	-5	Aanpassing van admin	Gebruiker
24-6-2017 - 13:25:07	-8.25	Aanpassing van admin	Gebruiker
24-6-2017 - 13:25:14	-4.5	Aanpassing van admin	Gebruiker

Figure 4.17: Interface of the transactions section in the administration system.

5. Security

Startups and small businesses often assume that they will not be a target of cyber crime, or at least not as often as bigger enterprises. A report by Symantec, a company leading in cybersecurity, published in 2016 [54], shows that small businesses (1 to 250 employees) were only a target 18% of the time in 2011 for targeted phishing attacks, but this has grown to 43% in 2015. On the other hand, the targeted attacks on larger enterprises dropped from 50% to 35% in the same timeframe. The report states *"small businesses have smaller IT budgets, and consequently spend less on cybersecurity than their large enterprise counterparts"* which could explain this trend. Since attackers are more frequently looking at smaller businesses, which includes the Arena, security is even more important than before.

While creating the software, we had a meeting with an expert in the domain of cyber security, Christian Doerr, an Assistant Professor of the Cyber Security Group at the TU Delft. This helped us shape the security research by giving us valuable recommendations.

First, the cyber security risks and recommendations will be explored. This is followed by the risk analysis of our software and finally the security implementations in the product and other implementations will be examined.

5.1 Risks and Recommendations

While researching the security implications of our project we used the CIA triad [6] to create a risk analysis of the project. The CIA triad is a model which can help guide cybersecurity decisions and consists of the following elements: confidentiality, integrity and availability. Confidentially means to prevent unauthorized users to access information they should not have access to. Integrity is the ability to make sure there is no unauthorized modification of information. Availability states that the system should always be accessible for legitimate users.

While implementing security solutions, a layered security design [58] is essential in case any single defense is flawed. By adding layers in secure systems, an attacker must find multiple vulnerabilities to compromise a system. Bugs and leaks can always be present in any system. An example of this is Heartbleed, a security bug in the OpenSSL cryptographic library which was introduced into the software in 2012 and only publicly disclosed in 2014 [7]. The Heartbleed bug impacted two thirds of the worldwide internet [1].

In the following sections, product risks are listed as attacks on these three elements of the triad and their solutions are discussed.

5.1.1 Eavesdropping

During a transmission, it is possible for malicious third parties to listen in on messages. If the interception of information is possible, the confidentiality of the data transmission is compromised.

In cybersecurity such an attack is called a man-in-the-middle attack and can be executed by using techniques such as *ARP Poisoning* and *DNS Spoofing* [12].

The *Address Resolution Protocol* (ARP) is a protocol which is used to link IP addresses to physical computer addresses, allowing computers to communicate in networks. During an ARP attack, an attacker pretends that a certain IP address belongs to him, resulting in the sender sending the packets to the attacker instead of the intended recipient. There are defense mechanisms that prevent these attacks on a network level, which should be configured by the network administrator.

The *Domain Name System* (DNS) is used for name resolutions. It links domain names to IP addresses so a computer knows which server to send a request to. By hijacking these requests, an attacker replies with a different IP address. DNS Hijacking can be prevented by DNSSEC [51]. This ensures that the DNS records are digitally signed and can be verified for legitimacy.

HTTPS

In case the solutions above are not adequate, the system is directly exposed. An extra security layer lowers the risk of system exposure. HTTPS [22, 35] creates a secure connection by using an encrypted exchange of data. In case an attacker intercepts the messages, they cannot be read.

Certificate Authorities

HTTPS uses certificates to encrypt messages. In addition to encrypting the data, the sender must be sure that the receiver is the correct one. By validating these certificates the sender can be sure that the receiver's identity is correct. This problem is solved by using Certificate Authorities.

A Certificate Authority [30] is an entity that issues certificates. By creating a Certificate Authority, a list of trusted certificates can be maintained. This means that in the event of a compromised component, this component can be removed from the chain of trust so that uncompromised components will not accept connections from it.

5.1.2 Forged Messages

In case of a network breach, malicious systems could send or modify messages to systems components. It is vital for systems to have the ability to determine which message are genuine and which are not, to preserve the integrity of the system and messages sent.

HMAC

A *hash message authentication code* (HMAC) provides a way to check the integrity of a message sent over a public network by hashing the message with a cryptographic key. To accomplish this both the sender and receiver have the same secret key. The sender generates a hash of the original message with the key, and the receiver can verify the contents of the message by hashing the message again with the same key. If the HMAC is the same, the message contents are legitimate and untampered. If they differ, someone has altered the contents of the message and it should not be trusted.

Whitelisting IPs

Another layer of defense to prevent forged messages from being processed by the system is to whitelist IPs. By limiting the IPs which are allowed to connect to the systems, it becomes much harder for an attacker to do any harm, preventing attacks and brute force attempts from unknown sources.

5.1.3 Code Injection

By providing malicious input to the system, an attacker could execute code with malicious side-effects. Examples of these kind of attacks are SQL injections and buffer overflow attacks. Code Injection is an attack on the integrity of the system.

Input Validation

A way to prevent abuse of the system is to perform input validation in every step of the system. In addition, SQL queries should make use of prepared statements to prevent user input from being processed as a SQL query.

5.1.4 Denial of Service

The goal of a Denial of Service attack [59] is to attack a system in such a way, that legitimate usage is no longer possible. Such an attack is detrimental to the availability of the system.

A common form of a Denial of Service attack is a DDoS (Distributed Denial of Service). During a DDoS the attacker sends enormous amounts traffic to the system so the maximum capacity of the internet connection becomes saturated. This causes the system to become unresponsive. Another type of DoS attack is a SYN flooding attack [59], where the attacker starts up a lot of TCP sessions so that the maximum amount of sessions is quickly reached.

Rate Limiting

Rate limiting sets a cap on the amount of requests which can be made within a given time. It can be used as a prevention for brute-force attacks. An example could be to only allow a few logins from the same origin in an hour.

Whitelisting

Some types of DoS attacks do not rely on an overflow of traffic, but uses specialized packets to cause a denial of service. By using a firewall to process and to reject these packets from unknown sources these types of DoS attacks can be mitigated.

Redundancy

Denial of Services attacks are capable of rendering an instance of a server unable to reply. In case one of our servers is under a Denial of Service attack, a host of back-up servers could be used that could process requests instead.

5.1.5 Social Engineering

Aside from digital security, social engineering is also a security risk. If employees have access to administrative functions this poses a new opening for attackers to intrude into the system and compromise the integrity of the system. In 2014, IBM [24] found that 95 percent of all security incidents happen because of a human error or mistake. This means that it is valuable to train the employees to identify potential risks at the Arena.

Phishing [61] is often used to lure employees into installing malware. Malware [60] is software which executes a program or executes other actions against the user's intent. It could be installed accidentally by following malicious links, or by opening an attachment of a malicious email.

Employee Training

To help prevent employees from installing malware, it is important to train employees with potential system access to be careful and watch for tell-tale signs of malicious messages. Some instructions may be very clear: do not open malicious attachments. Recent developments show that malware attacks on the masses are dwindling in favor of more targeted attacks to raise less suspicion [54]. For employees and system engineers this means that it becomes harder to distinguish these malicious attacks.

Customer Awareness

Aside from the damage an attacker can do with an employee account, a stolen card can lead to a loss of funds for a customer. This means it is important that a customer is aware of what they can do when their card has been stolen.

5.1.6 System Breach

If a third-party application which runs on the same server has a security leak, this could be enough to compromise the entire system. Symantec's Internet Security Threat Report [54] shows us that 75% percent of the sites and systems on the internet are not fully patched and contain security loops, with 15% having critical security issues between 2013-2015. An incredible amount, given the fact that in 2015 there was a new zero-day vulnerability found every week on average [54].

Update Policy

Because it is impossible to verify that software is exploit-free, it is important to make sure that components are updated as quickly as possible to fix newly publicized exploits. This lowers the time an attacker has to compromise the systems.

Database Access Control

In case an attacker breaches all other layers of security, the damages should be limited. To limit the damage any given compromised component can inflict upon the database, each system should run within their own restrictions on the database.

5.1.7 Replay Attack

In a replay attack, an attacker will repeat a transmission by reusing captured messages. Since the contents of the transmission is valid, the system will execute the same action. This is an attack on the integrity of the system.

One time token

By using one time tokens, a signature can be added to each message. This signature will be unique for each sent message, and can only be used once. In case the attacker tries to reuse a captured message, the system will ignore it.

5.2 Payment System Security Analysis

To analyze the security concerns in our application, the Payment System is used as an example. An interaction diagram of the Payment System can be seen in figure 5.1. Since the system consists of several different elements and dependencies, there are multiple locations an attacker could focus on. There are two different parts of the system in terms of attack possibilities: an attack on the technical part of the system, or an attack on the users of the system: the employees and members.

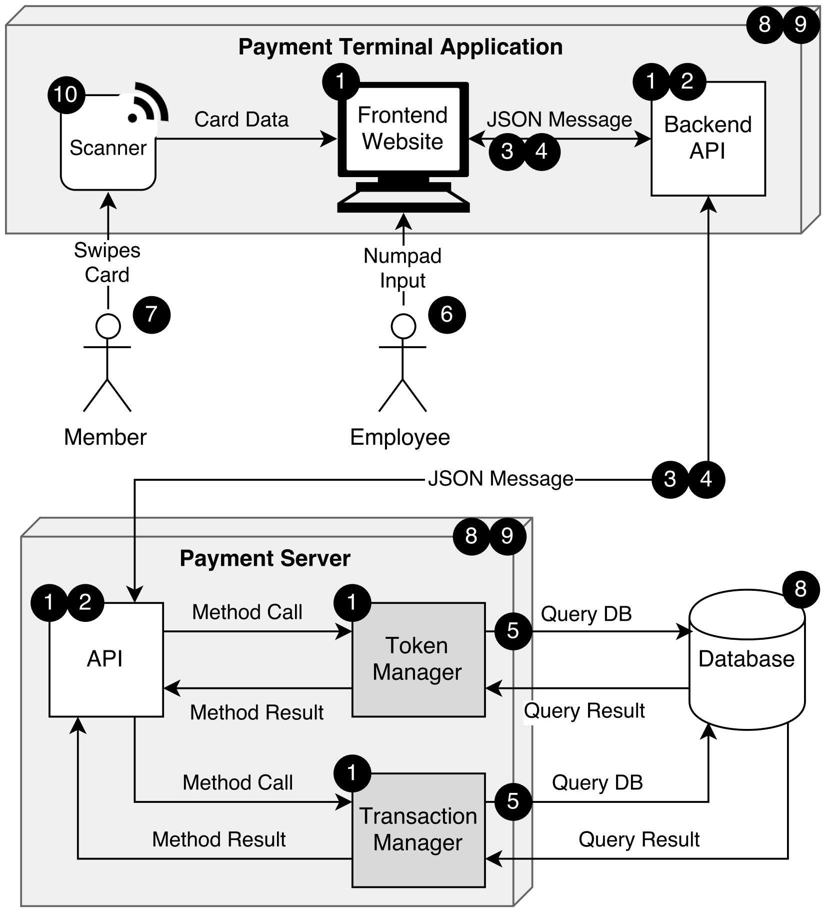


Figure 5.1: Security analysis of the Payment System.

Security systems have been either implemented or have been listed as a recommendation. The decision on which measures should be implemented within this project depended on three factors: the impact of the risks, availability of time, and whether the solution is a part of the product or the system the product runs on. Figure 5.1 shows security implementations and recommendations. Below, each measure is elaborated upon.

1. Input validation - Implementation

To prevent user input in the system to do any harm, SQL prepared statements are used, provided by the `mysql` package. By making use of prepared statements, user input is handled separately from the query, and thus the user input cannot be parsed as code. Additionally, all inputs are checked on validity for extra safety; for example, transaction amounts must be a decimal number.

2. Whitelisting - Recommendation

All APIs should only be accessible by known sources. Additionally, the Payment Terminal Application should only be accessible by EGA's tablet. Whitelisting can be achieved by using `iptables` [5], a system firewall available on Linux systems.

3. HMAC - Implemented

HMAC support is provided by the node.js standard library in the Crypto module [37]. The HMAC is sent as a header along with every request and is validated using the same library.

4. HTTPS - Implemented

HTTPS support is provided by Apache's reverse proxy capabilities [15]. The reverse proxy is configured to retrieve requested pages and to wrap them in HTTPS requests.

5. Database Access Control - Implemented

The systems must use different database credentials for the different applications and servers. In the table below the permission each system has been specified.

Component	**Privileges**
authentication-server	Read rows in `tokens`
payment-server	Create new rows in `transactions` table Update column balance in `users_balances`
check-in-application	Create new rows in `location_events`
admin-application	Create rows in `users` Edit rows in `users` Create rows in `users_balances` Edit rows in `users_balances` Create rows in `transactions` Edit rows in `transactions` Create rows in `tokens` Edit rows in `tokens`

6. Employee training - Recommendation

Employees can be another risk. By means of social engineering, hackers can try to access the systems by pretending to be an employee or administrator. As discussed in the social engineering paragraph, the best way to mitigate the risk is by properly teaching them about such attacks.

7. Customer Awareness - Recommendation

If a card is lost, the card could be abused by anyone and could be used to empty the account. Thus, the user should be aware of this and know what to do if he loses his card, for example by letting the card be blocked as soon as possible.

8. Update policy - Recommendation

The servers which run the systems should be properly maintained and updated frequently.

9. Redundancy - Implemented

The Payment Terminal Application has a list of Payment Servers and is capable of switching to an alternative server in case a server fails.

10. One time token - Recommendation

One time tokens should be used. The current RFID card system does not support this functionality and should be replaced for a system which does provide this feature. See the next section on Card System Flaw for more information.

5.3 Card System Flaw

In our prototype design for the project, RFID technology is used to carry and transfer data. Our cards only contain an identifier which cannot be changed. Another property of the RFID cards is that they can be read from a distance. In our typical testing scenario, the card only responds within 3-4 centimeters, but by using specialized hardware the ranges can be much larger [27].

This poses several security risks, as attackers could copy cards from other users, merely by placing high ranged scanners in the arena. This collected data could then be used to create their own cards, impersonating the actual account holders. Additionally, if the Payment Terminal is compromised, it is possible to repeat messages of a previously succeeded transaction, draining a user's balance. By using high-range scanners, attackers can copy a lot of cards and empty customer accounts, inflicting serious damage on EGA.

This finding has severe implications for our project. It means that for anything financially or otherwise related to value, these cards must not be used. A solution is to replace the RFID card system with another system which supports a form of one time tokens.

Smart cards [48] are commonly used for this purpose. These cards contain components which prevent tampering with the cards, such as clock fluctuation and unpredictable behavior [44]. The smart cards can be programmed to contain a secret key which only the card and the card issuer know. This secret key is used to generate responses from the card and card issuer which only they can generate. With each new transaction, a different response is given. This system of requests and responses is called a challenge-response authentication system [32].

The challenge-response authentication system solves the problem of attackers being able to read the cards because an attacker would need to know the key used to generate challenge responses. Even if an attacker could capture a message of a valid payment, both the card and the issuer would generate a different response. Since it is computationally hard to guess the key used to generate these responses, the system is more secure.

Figure 5.2 shows the current sequence diagram for the Payment System. By replacing the RFID cards for the smart cards and integrating this in the system, the new sequence diagram can be seen in figure 5.3. Since these smart cards cannot be copied and because the Payment Server now requires a signed message which only the card can produce, the introduction of smart cards solves the vulnerability.

Several commercial smart card systems exist. One of the widely available systems is JavaCard, which can be programmed and used for this purpose. If a smart card system is used, then the Payment System could be used in a real-world setting.

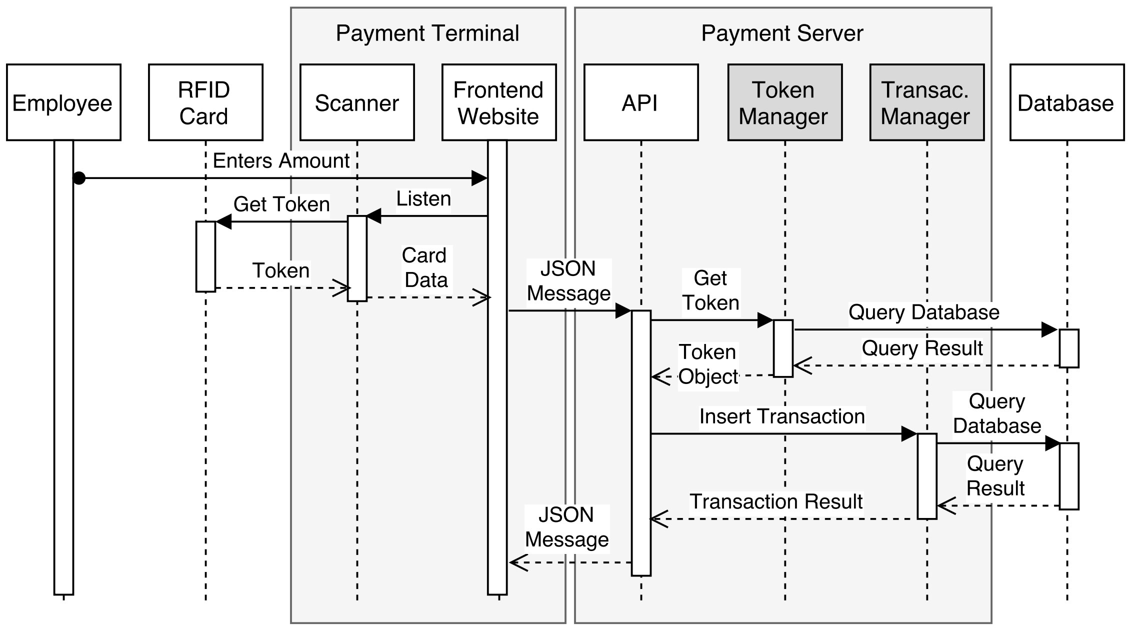


Figure 5.2: Sequence diagram showing communication in the current Payment System.

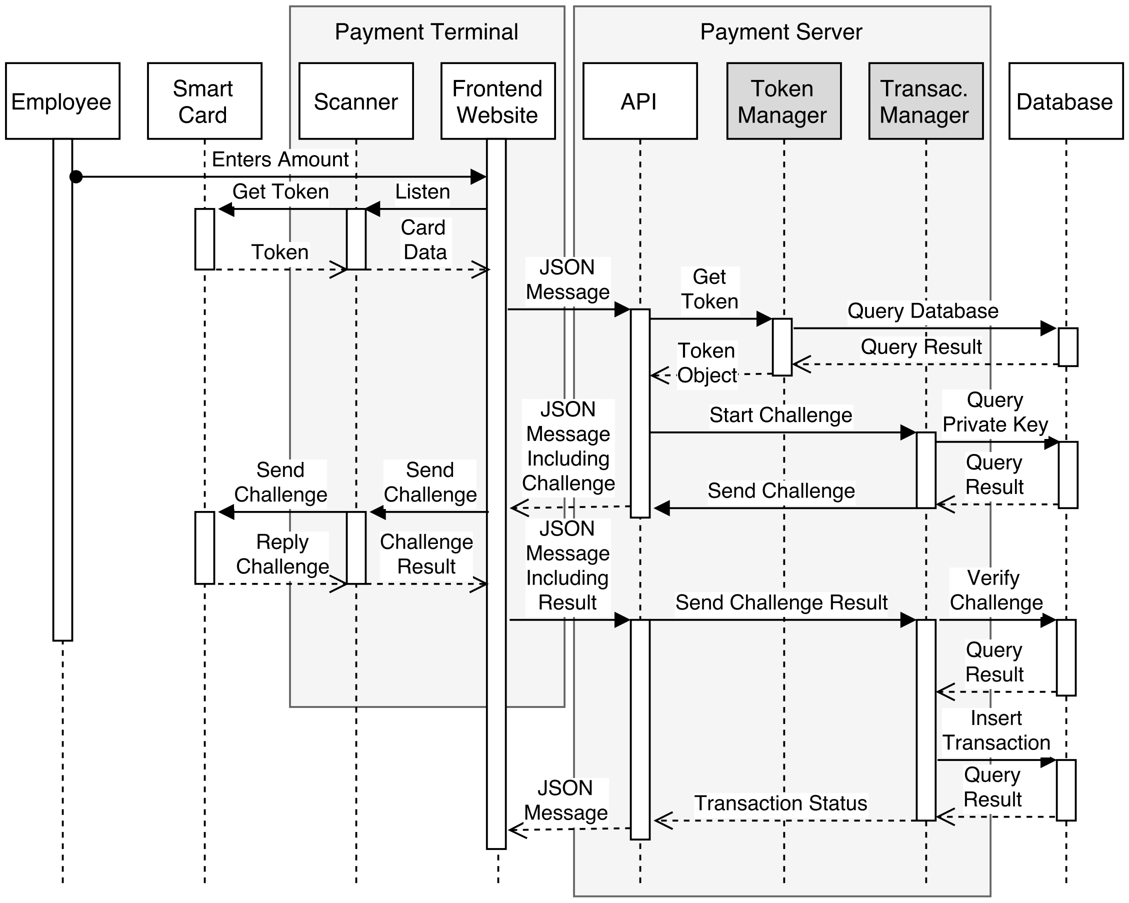


Figure 5.3: Sequence diagram showing the usage of the challenge-response system.

5.4 Security Conclusions

In this section cybersecurity principles were introduced by means of the CIA-triad. Confidentiality, integrity and availability form the key pillars in any cybersecurity system. Risks were examined to which solutions were listed.

Aside from the technical challenges, end-users also play a key role in keeping the system secure. Customers and employees can be targeted by social engineering attacks and should be properly educated on the risks such that they can play an active role in keeping the system secure.

Database access control, HMAC, input validation, and redundancy have been implemented for the final product. Furthermore, the systems use HTTPS communication between components to prevent eavesdropping.

With all the knowledge about risks, attacks and solutions, we examined the Payment System and concluded that it cannot safely be used in its current state. The RFID card technology used does not provide a secure base for a Payment System as it is susceptible to replay attacks. A system which uses smart cards and a challenge-response system must be implemented before the Payment System can be used securely.

Finally, we have other security recommendations which should be implemented when the systems are deployed. These include setting up proper firewall rules, setting up a certificate authority, having a good update policy, and training customers and employees.

6. Testing

Testing is a vital part of any software project. The primary purpose of testing is to validate the implementation of the requirements and to detect mistakes made in the code of the projects. Our client does not have technical experience, so we had set our own standards for testing the software.

This section describes the use of testing during the development of the project and lists the approaches used. The details of the decisions on the test frameworks are discussed in appendix C: *Technologies and Framework Choices*.

6.1 Framework

We used Mocha [33] in our project as the test framework to run code tests. Mocha runs tests and displays their result. An example of mocha's output can be seen in figure 6.1. The output can be used by the programmer to quickly verify if the program performs correctly, and if not, which functionality does not work as intended. It also shows the total amount of tests.

```
PaymentPage
  #paymentCompleted
     ✓ should reset form when status = 200
     ✓ should reset token when status = 404
     ✓ should not reset amount when status = 404
  #getToken
     ✓ should delegate token data properly
  ....
 233 passing (15s)
```

Figure 6.1: A part of the test results from mocha of the entire card system.

6.2 Unit Testing

Throughout the project, we wrote unit tests alongside production code to verify correct behavior of existing code. Using these unit tests, each component of our system was checked as soon as it was created. By adding unit tests into the test suite, regression bugs are less likely to occur [57] as the codebase is tested with each new implementation.

We used a mocking framework called mockery [8] to create objects that imitate the behavior of complex modules the code under test uses. This enabled us to isolate the code which was being tested from those complex modules.

6.3 Test Coverage

Coverage is provided with the istanbul [39] framework. Istanbul can generate a report of code coverage, which shows programmers the parts of programs that have not been covered by tests and therefore need attention. In general, 60 to 75% statement coverage is considered reasonable [23]. At the end of our project, the measure statement coverage was 93,21%, meaning that the code is well-tested. A complete coverage report generated by istanbul can be found in appendix D: *Istanbul coverage report.*

6.4 End-to-end Testing

End-to-end tests were used to validate the functional requirements. These end-to-end tests were run manually by the team using the user scenarios (see appendix B: *User Scenarios*) created after settings the functional requirements. In chapter 8: *Product Evaluation* the results of these tests are listed.

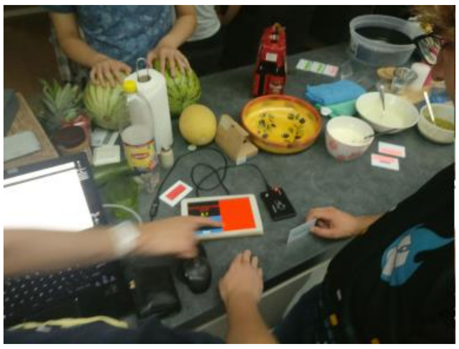
6.5 Real World Usability Testing

In order to test the usability of the developed products, we created a demo setting in which we set up our product. Photographs of this setting can be found in figure 6.2. Using a tablet similar to the one EGA might use, our scanner, cards, and some drinks were used to simulate the real world use of our product.

We asked friends and family to play the role of customers and employees. Without any instructions on how to operate the system, we let them use the system and observed what happened in the process. Different cards were handed out: blocked cards, cards which did not allow for payments and regular cards with (in)sufficient balance. Some instructions were given to the customers, for example: “please ask the employee to unblock your card, after that, try to use it for a payment”.

The experiment provided us with useful insights. The attendees had no trouble using the system at all. They managed to find and use all the functionality without the need to ask for our help. They were also content with the speed and performance of the system: it performed as they expected.

Also, the attendees managed to find some bugs in the system. Most of the bugs were found due to the initial attitude: the attendees experienced great joy in trying to break the system instead of using the system in the way it would be used in a live situation. For example, the users found out that it was possible for a customer to scan and complete


(a) Receiving feedback on the UI.

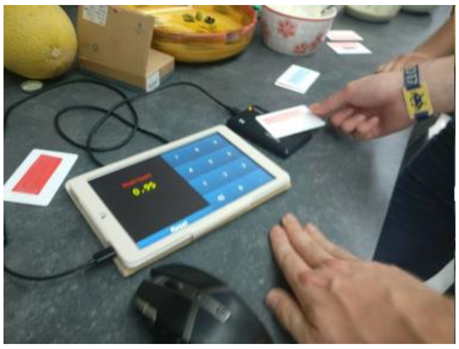

(b) Customer scans his card.

Figure 6.2: Photographs made while testing.

a transaction before the customer was allowed to. Furthermore, it was possible to change the transaction amount even after the pay button was pressed. We had not anticipated these situations, and fixed these bugs soon after.

In addition to the discovered bugs, the attendees also suggested some features, for example to be able to enter amounts without the need to enter full decimals, such as ".8" instead of "0.80" and "2.4" instead of "2.40". In a real world situation they also prefer to have a direct form of feedback for the customer as to whether the transaction succeeded or not, for example, with an additional monitor facing the customer. These suggested features were added to section 10: *Recommendations*.

6.6 Stress Testing

In order to find out how the systems would perform under heavy load, we developed a script that would request payments as quickly as the computer could send them. Using this, we discovered that the platform could handle a maximum of 406 payments per second averaged over five tests. However, these measurements were done in a test environment with only a single computer, meaning that the actual throughput could be much higher as the test was running on the same computer as the tested software. For reference, this computer used had the following specifications: Intel(R) Core(TM) i5-3450 CPU @ 3.10GHz with 16GB RAM.

Since 406 payments per second is far above the expected usage of the system, we can conclude that the system does well under stress.

7. Process Evaluation

This chapter describes the manner in which the team worked during the course of the bachelor project. The project started with a two week discovery phase to conduct research, to set the product definition, and to create a system design. In the weeks following this, our time was spent on creating the first working prototype. In week six we had the feature freeze and shifted our attention towards the security research and implementation. During the last week we finished the report.

7.1 Team

From the start of the project, we had agreed to work together at the same location as much as possible, maximizing contact hours. This proved to be very valuable as we could directly ask questions and get responses immediately.

Due to the nature of the project, the actual development time was only about six to seven weeks, meaning that a suitable development strategy had to be chosen. We started with a global planning for the entirety of the project, and we had a weekly planning review to adjust it where needed.

Since the design was structured into three layers, the modules, APIs and applications, the development process started with the modules, followed by the APIs and finally the applications. This ensured that there was no unnecessary delay when implementing components as their dependencies were already developed.

From the start of the project we had a Gitlab [26] server at our disposal, which we used for version control [19] and project management. Gitlab was configured to use Jenkins' pipeline options. Jenkins is a continuous integration and development platform to build, test and deploy our software. When our team wanted to add functionality to the products, Jenkins automatically checked the quality of code by running a pipeline with multiple steps such as building, testing, validating the code style and producing coverage reports. The team could see the status of this pipeline and would not accept any changes if Jenkins showed any failed steps. When changes were accepted, Jenkins would deploy the software into our online test environment.

7.2 Coach

Our coach, Claudia Hauff, was very helpful during the project. In the first week, we had ambitiously drafted a requirements document with too many requirements. We estimated that it would be feasible to complete the project within the timeframe using these requirements, but Claudia advised to cut down on the amount of requirements. We followed her advice, and at the end of the project, we were glad we did because we would not have been able to complete the project if we had kept our initial requirements.

In the beginning, our team had frequent meetings with her in order to make sure we remained on schedule. Also, she gave us a lot of constructive criticism on our research report and our final report. This improved our ability to write a cohesive report.

Claudia also suggested that we meet with a cybersecurity professor at the university, Christian Doerr, which was a very valuable suggestion. Thanks to the meeting with Christian Doerr, we learned a lot about potential attacks on our system and how we could mitigate them.

7.3 Client

Our contact with the client was rapid and efficient. Within the first week of the project, we had quickly set up a meeting with him from which we distilled his wishes and requirements. Based on those requirements, scenarios were created which described the functionality of the system for end-users. Our client then reviewed and accepted these scenarios. This was useful because it quickly became clear which features were necessary to implement to call the project a success.

After the two week research phase, we started working on the first working prototype. When the first prototype was finished, we had it approved by the client to check that it met the client's needs.

In agreement with the client, we aimed to have a feature freeze on the requirements in week six. This meant that all features would be finalized so that our attention could be moved towards further research on the project's security risks and implementing measures to counteract them.

During the final weeks we kept the client informed about the progress on the security research and the recommendations that would follow. Overall the client was content with our communication and progress. He also expressed his appreciation for the independence of the team.

7.4 Software Improvement Group

The bachelor project required us to send our produced code to the Software Improvement Group (SIG) twice during the course of the project, once in week six and again in week nine. Each time SIG gave us feedback as to how we would could improve our code, and gave us a score based on the quality of our code.

7.4.1 First Evaluation

In the first submission, we received a score of three and a half out of five stars on their maintenance model, which, as they stated, was above average. According to their feedback, we would have received the highest score if not for one case of code duplication. SIG had pointed out one class in our code which was duplicated between two modules. We had quickly fixed this by extracting the duplicate code to a new module.

Aside from this, SIG said that the tests we had written were promising and we should keep up the testing mentality. This was good to hear because we had put a lot of effort into properly testing our code base.

7.4.2 Second Evaluation

In the second evaluation, SIG looked at the progress we made since the previous evaluation. They specified that the size of the project as well as the score for maintainability have both increased. The increase in score is mainly caused by reducing code duplication, but also an improvement of the software architecture.

Additionally, SIG was glad to see that the amount of test code increased with the addition of more production code.

8. Product Evaluation

In this chapter the final product will be evaluated. First, the *Product Specification* which was set at the start of the project will be evaluated, followed by the design goals. Finally, the success criteria will be re-accessed.

8.1 Functionality Evaluation

In tables 8.1, 8.2, 8.3, and 8.4, the functionalities from section 2.1: *Product Specification* are repeated and reviewed.

Functionality	**Met?**
Check-in System	
The system must be able to detect when a card is swiped.	✓
The system must be able to distinguish between a user entering the building or exiting the building.	✓
The system must be able to save these events into a database.	✓

Table 8.1: Table of product requirements for the Check-in System

Functionality	**Met?**
Console Locking System	
Gaming consoles must be able to be locked.	✓
Gaming consoles must be able to be unlocked.	✓
The system must be able to detect when a card is swiped.	✓
The system must be able to check the validity of a membership card.	✓

Table 8.2: Table of product requirements for the Console Locking System

Functionality	**Met?**
Payment System	
Users must be able to have an account balance.	✓
Users must be able to have an account balance limit.	✓
The system must be able to detect when a card is swiped.	✓
The system must be able to check the validity of a membership card.	✓
The system must be able to check if the user's balance is sufficient for a transaction.	✓
The system must be able to deduct balance from a user's account.	✓
The system must be able to log all events which alter the user's balance.	✓
The system must be able to reply appropriately.	✓
The system must be able to see the transaction's result.	✓
The system must be usable on a tablet device.	✓

Table 8.3: Table of product requirements for the Payment System

Functionality	**Met?**
Administration System	
Administrators must be able to assign membership cards to a (new) user using the existing EGA account database.	✓
Administrators must be able to block or unblock membership cards.	✓
Administrators must be able to allow or disallow payments for membership cards.	✓
Administrators must be able to adjust member's balances.	✓
Administrators must be able to view all membership cards in the system.	✓
Administrators must be able to view all transactions processed in the system.	✓
Administrators must be able to generate an overview of the balance of all users.	✓

Table 8.4: Table of product requirements for the Administration System

As seen in tables 8.1, 8.2, 8.3, and 8.4, all the functionalities have been implemented and thus meet the requirements of each product in this project.

For the functionality evaluation the end-to-end tests were used to determine if all required functionalities were implemented correctly from a story perspective. Again, these user scenarios can be found in appendix B: *User Scenarios*. The result of these end-to-end tests can be seen in table 8.5.

User Scenario	**Done?**
Visitor wants a new membership card	✓
Payments at the bar using membership card	✓
Lost membership card	✓
Managing finances	✓
Entering and leaving the arena as a member	✓
Accessing the gaming consoles	✓

Table 8.5: The results of the end-to-end tests.

8.2 Design Goal Evaluation

8.2.1 Security

Due to the nature of financial transactions, security became the key pillar in this project. In chapter 5: *Security*, we have dedicated significant time and effort in researching potential security threats as well as their solutions to mitigate these security risks. Of these solutions, some were implemented in the products, the rest were left as recommendations.

An important conclusion made in chapter 5: *Security* was that the current card technology cannot be safely used as a payment method and should be replaced by a more secure card technology. However, for the Check-in and Console Locking application, the current cards are satisfactory because the risk impact is limited.

8.2.2 Reliability

The system had to be reliable, meaning that the system should guarantee that no information is lost and should make sure that information is never incomplete. The system structure is built so that a transaction is either fully executed or not at all by using SQL transactions. This means that, it is not possible for a transaction to be stored, and that the user balance is not altered.

8.2.3 Availability

Downtime will impact the revenue of the Arena negatively which must be avoided. The Payment Terminal Application can automatically use a different server in case one of them fails. On top of this, all crashed servers will restart themselves automatically.

8.2.4 Maintainability

The products should be extendable and easy to maintain. Thanks to the layered structure of our Product Design, the product is highly maintainable and extendable. The feedback from the Software Improvement Group reinforces this claim by stating that the code is *"above average maintainable"*.

8.2.5 Performance

In section 6.6: *Stress Testing*, it was shown that in a very basic test setup the system could handle over 400 transactions per second. For the use case of this product this is far above satisfactory. According to user tests the system also felt smooth.

8.2.6 Usability

It is important that the system feels easy to use. Section 6.5: *Real World Usability Testing* evaluated the usability and shows that end-users felt the system was easy to learn and use.

8.3 Project Success

During the start of the project the project's success criteria were defined as follows:

> *"One of the success criteria defined by the client, is that a prototype payment system and administration system must be completed. Additionally, the client also emphasized that the membership cards must be linked to users in the existing EGA account database"*

In section 8.1: *Functionality Evaluation*, the required functionalities (Payment System and Administration System) were completed. The prototypes of the Check-in System and Console Locking System were also completed.

Since the Payment System cannot be securely used in a live environment, this prototype has not been a complete success. However, since the goal of the project was to discover and build feasible prototypes, we would say that that the Payment System is, as a product, not a success, but that the project as a collection of feasible prototypes has met its expectations.

In addition, as can be seen in section 8.2: *Design Goal Evaluation*, all design goals were properly taken into account, with specific attention to the security of the system.

Because both the functional requirements and the design goals have been completed, we conclude that the goal of the project has been met and thus the project is a success.

9. Ethical Evaluation

When building applications which interact with customers, an ethical evaluation is needed to discuss the impact on the life of these users. The most important issue in our product concerns the customer's money. For this, the implications of allowing credit and debit balances are discussed. Another issue faced by the system is how to handle the personal data obtained from the membership card system. What information should EGA be allowed to collect, and how important is consent in this regard?

9.1 Financial Implications

Any system which allows users to have a balance will have the option to let users have a credit balance and/or a debit balance.

When users are allowed to have a debit balance, this means that the user first has to add money to the balance before products or services can be paid for. The ethical problem which arises is that the company already obtained the money without immediately providing a product or service in return. If the company is unable to deliver a product or service afterwards, it may be impossible for a customer to get their money back.

For this purpose, external trust funds exist. This trust fund is a third party with the purpose of 'holding' the money of a customer. In case the company is unable to provide their products or services, this eliminates the risk for customers to lose their money. In such a case, the trust funds can return the money to the customers. If EGA allows users to have a debit balance, EGA should make use of such an external trust fund.

A credit balance allows the user to make purchases even though their balance is not sufficient. This means that the user owes the company a debt which has to be paid. It can be argued whether it is wise to allow customers to owe money, as it provides a way to spend money which the customer might not have. This does not yet take into account that the card can be used by malicious actors, creating unforeseen debt.

The payment system we have built allows for both credit and debit balances. In short, the administrator can set the limit on the balance of a member to any real number. This way, EGA will be able to determine how they want to manage their finances themselves, and it is easy to change policies without changing the systems infrastructure.

9.2 Personal Data Implications

The membership card system allows for information about members to be collected. For EGA, or any company in that regard, it is valuable to know the behavior and patterns of their customers. However, the customer might not be eager to share this information. Additionally, in the current design of the system a membership card is *required* to be able to access the consoles. This means that members that do not have a card will have a reduced availability of hardware at the Arena, which can be seen as forcing the customer to use a membership card.

10. Recommendations

Even though the product has been delivered, we still have recommendations for EGA that should be considered before the product becomes operational.

A number of security recommendations are discussed in chapter 5: *Security.* The most important conclusion drawn was that the card system will have to be replaced in order to be used for the payment system. Further recommendations include security measures during deployment and good practices for system users.

In section 6.5: *Real World Usability Testing*, additional feature requests were made by attendees of the real world test. For example, to change the way the amounts are processed, as some intuitive amounts are not recognized, for example ".8" instead of "0.80". Another suggestion was to add an additional monitor for the customer which provides feedback on the transaction.

Also, the materials used in the prototypes are currently not suitable for live use. The console-locking device is currently made from wood which does not conform to fire safety standards. Since the device uses electricity and generates heat, it would be better to use more suitable material. Also, the Check-in Application also has no casing yet. To use the Check-in Application in a real world setting, it needs to be visually attractive.

Finally, as mentioned in chapter 9: *Ethical Evaluation*, a trust fund is vital if EGA decides to allows users to have a debit balance.

11. Conclusion

EGA wanted to give their members a branded membership card. This project was created to discover the possibilities such a card could offer.

After nine weeks of work the project was fully researched, designed, implemented and tested. The efforts resulted in a working prototype which included all of the systems the client requested. The security implications posed by this product played a large role in making the project an interesting, unique, and educative experience.

Multiple systems have been built which EGA can use. These allow users to unlock consoles, pay for refreshments, and to check in and out of the building. Additionally, the card system can be managed by employees in the administration software. The layered design of the system allows for easy maintainability and extension.

Although we have met all requirements of the products, we believe the payment components of the product cannot be used as-is in a real world scenario due to fundamental flaws in the chosen card system for the prototype. By replacing this prototype card system with a more secure system, the Payment System could be used in a production environment. Despite the security implications for the Payment System, we believe the other systems can safely be used as intended.

The project success criteria have been met, resulting in the success of the project. Finally, we advise to take into account the security and project recommendations for future deployment.

Bibliography

[1] Ionut-Daniel Barbu and Ioan Bacivarov. Heartbleed-the vulnerability that changed the internet. *Int'l J. Info. Sec. & Cybercrime*, 3:49, 2014.

[2] Fatna Belqasmi, Roch Glitho, and Chunyan Fu. Restful web services for service provisioning in next-generation networks: a survey. *IEEE Communications Magazine*, 49(12), 2011.

[3] THE NPM BLOG. how many npm users are there? `http://blog.npmjs.org/post/143451680695/how-many-npm-users-are-there`. Accessed: 25-06-2017.

[4] Joe Casad. *Sams teach yourself TCP/IP in 24 hours*. Sams Publishing, 2004.

[5] CentOS. Iptables. `https://wiki.centos.org/HowTos/Network/IPTables`. Accessed: 25-06-2017.

[6] Yulia Cherdantseva and Jeremy Hilton. A reference model of information assurance & security. In *Availability, reliability and security (ares), 2013 eighth international conference on*, pages 546–555. IEEE, 2013.

[7] Codenomicon. The heartbleed bug. `http://heartbleed.com/`. Accessed: 25-06-2017.

[8] Martin Cooper. Mockery - simplifying the use of mocks with node.js. `https://github.com/mfncooper/mockery`. Accessed: 25-06-2017.

[9] Oracle Corporation. Simple object access protocol overview. `https://docs.oracle.com/cd/A97335_02/integrate.102/a90297/overview.htm`. Accessed: 25-06-2017.

[10] Erik DeBill. Module counts. `http://www.modulecounts.com/`. Accessed: 25-06-2017.

[11] Priority 1 Design. Em4100 protocol description. `http://www.priority1design.com.au/em4100_protocol.html`. Accessed: 25-06-2017.

[12] Yvo Desmedt. Man-in-the-middle attack. In *Encyclopedia of cryptography and security*, pages 368–368. Springer, 2005.

[13] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[14] Klaus Finkenzeller. *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. John Wiley & Sons, 2010.

[15] Apache Software Foundation. Reverse proxy guide. `https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html`. Accessed: 25-06-2017.

[16] Node.js Foundation. Node.js: A javascript runtime built on chrome's v8 javascript engine. `https://nodejs.org/en/`. Accessed: 25-06-2017.

[17] Marco Franssen. Jasmine vs. mocha. `https://marcofranssen.nl/jasmine-vs-mocha/`. Accessed: 25-06-2017.

[18] Riot Games. 2016 league of legends world championship by the numbers. `http://www.lolesports.com/en_US/articles/2016-league-legends-world-championship-numbers`. Accessed: 25-06-2017.

[19] GitHub. Getting started - about version control. `https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control`. Accessed: 25-06-2017.

[20] Google. Chrome v8 - google's high performance, open source, javascript engine. https://developers.google.com/v8/. Accessed: 25-06-2017.

[21] Google. Google chrome downloaden en installeren. https://support.google.com/chrome/answer/95346?co=GENIE.Platform%3DDesktop&hl=nl. Accessed: 25-06-2017.

[22] Walter Goralski. *The Illustrated network: How TCP/IP works in a modern network.* Morgan Kaufmann, 2017.

[23] Dorothy Graham, Erik Van Veenendaal, and Isabel Evans. *Foundations of software testing: ISTQB certification.* Cengage Learning EMEA, 2008.

[24] IBM. 2017 ibm x-force threat intelligence index. https://www.ibm.com/security/data-breach/threat-intelligence-index.html?ce=ISM0484&ct=SWG&cmp=IBMSocial&cm=h&cr=Security&ccy=US. Accessed: 25-06-2017.

[25] StrongLoop IBM. Express: Fast, unopinionated, minimalist web framework for node.js. https://expressjs.com/. Accessed: 25-06-2017.

[26] Gitlab Inc. Gitlab - the platform for modern developers. https://about.gitlab.com/. Accessed: 25-06-2017.

[27] SkyRFID Inc. Rfid tag maximum read distance. http://skyrfid.com/RFID_Tag_Read_Ranges.php. Accessed: 25-06-2017.

[28] Smita Kumari and Santanu Kumar Rath. Performance comparison of soap and rest based web services for enterprise application integration. In *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on*, pages 1656–1660. IEEE, 2015.

[29] Erik Meijer and Peter Drayton. Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages. In *OOPSLA'04 Workshop on Revival of Dynamic Languages*, 2004.

[30] Microsoft. How certificates work. https://technet.microsoft.com/en-us/library/cc776447(v=ws.10).aspx. Accessed: 25-06-2017.

[31] Microsoft. Typescript - javascript that scales. https://www.typescriptlang.org/. Accessed: 25-06-2017.

[32] Chris Mitchell. Limitations of challenge-response entity authentication. *Electronics letters*, 25(17):1195–1196, 1989.

[33] Mocha. Mocha: simple, flexible, fun. https://mochajs.org/. Accessed: 25-06-2017.

[34] Gavin Mulligan and Denis Gračanin. A comparison of soap and rest implementations of a service based interaction independence middleware framework. In *Winter Simulation Conference*, pages 1423–1432. Winter Simulation Conference, 2009.

[35] Network Working Group. Http over tls. https://tools.ietf.org/html/rfc2818. Accessed: 25-06-2017.

[36] Newzoo. Esports revenues will reach $696 million this year and grow to $1.5 billion by 2020 as brand investment doubles. https://newzoo.com/insights/articles/esports-revenues-will-reach-696-million-in-2017/. Accessed: 25-06-2017.

[37] Node.js. Crypto class hmac. https://nodejs.org/api/crypto.html#crypto_class_hmac. Accessed: 25-06-2017.

[38] NPM. blanket. https://www.npmjs.com/package/blanket. Accessed: 25-06-2017.

[39] NPM. istanbul. https://www.npmjs.com/package/istanbul. Accessed: 25-06-2017.

[40] NPM. jscover. https://www.npmjs.com/package/jscover. Accessed: 25-06-2017.

[41] NPM. sinon. https://www.npmjs.com/package/sinon. Accessed: 25-06-2017.

[42] NPM. testdouble. https://www.npmjs.com/package/testdouble. Accessed: 25-06-2017.

[43] NPM, Inc. Npm.js: Build amazing things. https://www.npmjs.com/. Accessed: 25-06-2017.

[44] University of Chicago. Smart card technology and security - hardware security. http://people.cs.uchicago.edu/~dinoj/smartcard/security.html#Hardware%20Security%20Features. Accessed: 25-06-2017.

[45] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.

[46] Pivotal. Jasmine: Behavior-driven javascript. https://jasmine.github.io/. Accessed: 25-06-2017.

[47] Jack Radzikowski. Convergence of payments and government identification cards. https://www.securetechalliance.org/secure/events/20030715/BusinessTrack/WB09c_Jack_Radzikowski.pdf. Accessed: 25-06-2017.

[48] Wolfgang Rankl and Wolfgang Effing. *Smart card handbook*. John Wiley & Sons, 2004.

[49] Katherine M Shelfer and J Drew Procaccino. Smart card evolution. *Communications of the ACM*, 45(7):83–88, 2002.

[50] Joe Gibbs Politz Shriram Krishnamurthi, Benjamin S. Lerner. Programming and programming languages. http://papl.cs.brown.edu/2016/. Accessed: 25-06-2017.

[51] SIDN. Dnssec. https://www.sidn.nl/a/veilig-internet/dnssec. Accessed: 25-06-2017.

[52] Sinon.JS. Sinon.js: Standalone test spies, stubs and mocks for javascript. http://sinonjs.org/. Accessed: 25-06-2017.

[53] Sony. Turning your ps4^{TM} system on and off. http://manuals.playstation.net/document/en/ps4/basic/power.html. Accessed: 25-06-2017.

[54] Symantec. Internet security threat report, volume 21, april 2016. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf. Accessed: 25-06-2017.

[55] Symfony. Why should i use a framework? http://symfony.com/why-use-a-framework. Accessed: 25-06-2017.

[56] David Tang. Jasmine vs. mocha, chai, and sinon. http://thejsguy.com/2015/01/12/jasmine-vs-mocha-chai-and-sinon.html. Accessed: 25-06-2017.

[57] Xinye Tang, Song Wang, and Ke Mao. Will this bug-fixing change break regression testing? In *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on*, pages 1–10. IEEE, 2015.

[58] TechRepublic. Understanding layered security and defense in depth. https://www.techrepublic.com/blog/it-security/understanding-layered-security-and-defense-in-depth/. Accessed: 25-06-2017.

[59] TechTarget. denial-of-service attack. http://searchsecurity.techtarget.com/definition/denial-of-service. Accessed: 25-06-2017.

[60] TechTarget. malware (malicious software). http://searchsecurity.techtarget.com/definition/malware. Accessed: 25-06-2017.

[61] TechTarget. phishing. http://searchsecurity.techtarget.com/definition/phishing. Accessed: 25-06-2017.

[62] Michael Zur Muehlen, Jeffrey V Nickerson, and Keith D Swenson. Developing web services choreography standards—the case of rest vs. soap. *Decision Support Systems*, 40(1):9–29, 2005.

Appendices

A. Problem Description BEPsys

The following description is the original project description as found on BEPsys:

Project Title

RFID Card system for internet café

Project Description

A Game cafe / Arena will be opened in July. Members of this internetcafé will be given an RFID key card, which could be used to identify members - similar cards as for example your bank card or the OV-chipcard. The project's goal is to create a system which will make use of this identification with the RFID cards. One application of this could be that members could identify themselves with this card when entering and leaving the building. Other potential features of this card are: payments at the bar, identification for tournaments and logging in to the computers on location.

To complete this project you will:

- Research which features are viable along with the client.
- Work with both hardware and software in a real-world setting.
- Lead the entire development trajectory from idea to prototype.

Company Description

The E-Sports Game Arena (EGA) is a new concept in the Netherlands. The E-Sports Game Arena will facilitate computers and gaming equipment to host tournaments and also provides daily slots for casual gaming. During the weekends or evenings professional tournaments take place which will be livestreamed on popular video platforms. The goal is to bring gamers together and grow the esports community in the Netherlands.

B. User Scenarios

This chapter contains scenarios based on the functionalities in the Product Specification. These scenarios provide a context for the functionalities in a real-world scenario and are used as guidelines for creating the user experience. Additionally, they are used for the *End-to-end Testing*, seen in section 6.4, to make sure no use cases are missed.

B.1 Visitor wants a new membership card

If a customer does not have a membership card yet, they are asked if they want one. The customer can choose one of the following options:

1. the customer wants a membership card, but does not want to use the card for payments. In this case the card is blocked for payments, but works for all systems that require the card as authentication method;
2. the customer wants a membership card, and wants to be able to use the card as payment method for refreshments at the bar.

The employee registers the card for the customer in the card system through the *Administration Application*. This requires the customer to provide their existing EGA username which is required for the account creation in the card system. If the customer does not have an EGA account, an employee can assist him or her in creating a new EGA account. If the customer has agreed to make payments using the membership card, the employee can define a limit to the amount the customer can be in debt.

B.2 Payments at the bar using membership card

A member wants to buy some refreshments at the bar. To pay for the refreshments, the member chooses to use the membership card as payment method. After the employee takes the complete order and calculates the total cost of the order, the employee enters the amount the member has to pay in the Payment Application. Then, the member is asked to 'swipe' his card on the reader to complete the payment.

After the membership card is read, the payment system will process the transaction. A request to the Payment Server is made and the server will respond as follows:

1. the membership card has been blocked;
2. the membership card is invalid;
3. the membership card is valid, but is not linked with an account enabled for payments;
4. the membership card is valid and allows for payments, but the amount billed exceeds the member's limit;
5. the membership card is valid, allows for payments, and would not be billed over their limit.

For cases one through four, the employee is shown why the transaction failed. The member is required to use another payment method. In case five, the payment system creates a transaction and reduces the balance of a user. The employee can see that the payment has been successfully processed and the member can enjoy his or her refreshments.

B.3 Lost membership card

To prevent the abuse of a lost membership card, the member who lost his card can contact EGA to ask for their lost card to be blocked. The employee handling the issue opens the Administration Application and checks the member's account by asking some questions to verify if the member is actually the rightful owner. Then, the employee blocks the card. The blocked card can now no longer be used for authentication.

Upon blocking the card, the member is asked if he or she wants a new membership card. If the member agrees, the employee can go through the process of assigning a new membership card as is described in the "Visitor wants a new membership card" user story.

B.4 Managing finances

The treasurer can access the administration system and is able to see a list of all members' balances. The treasurer can use this data to manually send invoices to members in debt. Also, he can reset the member's balance.

B.5 Entering and leaving the arena as a member

When a member with a card enters the Arena, he or she can swipe their membership card at the entrance to log his presence. Similarly, the member can swipe the card to exit the building.

B.6 Accessing the gaming consoles

The Arena provides free entry. Consoles require an active subscription to the Arena and should thus be prevented from being used by non-members. A member can access the console by swiping his or her card against the scanner, which is attached to the Console Locking Application. After the member is finished playing, he or she can swipe again to lock the console again.

C. Technologies and Framework Choices

This section is dedicated to discover what hardware and software best fit the requirements of the product. We will first take a look at the card system used and how this works. This is followed by deciding the communication infrastructure for the product. Finally, we will take a look at the different programming languages to be used and which technologies we could use for the different parts of the systems.

C.1 Card System

The card system provides users with a way to interact with the system. Several card technologies exist which could be used for our products. First we will take a look at smart cards, which have been used for over twenty years [47]. Then we will examine RFID, a technology which enables cards to be used wirelessly. Finally, the choice we made will be elaborated upon.

C.1.1 Smart Cards

Smart cards are electronic data storage systems that can have a microprocessor for additional computing capacity. For convenience, smart cards are usually incorporated in a plastic card and have found a use in a variety of applications including health insurance cards, debit and credit cards, digital signature, and personal ID cards [48].

The smart card, is supplied with energy and a clock pulse from the smart card scanner via the contact surfaces. On top of this, some smart card systems also have support for a contactless interface by using RFID technology, as discussed in the next section. Although easy in use, contactless cards can allow malicious actors to extract and manipulate data without the knowledge of the cardholder [48].

Smart cards can be divided in two categories: memory cards and processor cards [14, 48, 49].

Memory cards store data on the chip. A part of the data is stored in the public part of the card, whereas another part is stored in private memory, protected by simple algorithms such as stream ciphering [14]. The cards are usually optimised for a specific application. An advantage of memory cards is that they are very cost effective [49].

Processor cards contain a microprocessor with data storage, an operating system and temporary working memory [14]. This allows processor cards to store secret keys securely and allows the implementation of modern cryptographic algorithms [48]. These cards are generally more expensive than the memory cards previously mentioned.

C.1.2 RFID

Radio Frequency IDentification (RFID) is the technology which enables contactless interfaces to be used with smart cards. RFID makes use of electromagnetic fields to power the card. The underlying technical procedure is drawn from the fields of radio and radar engineering [14].

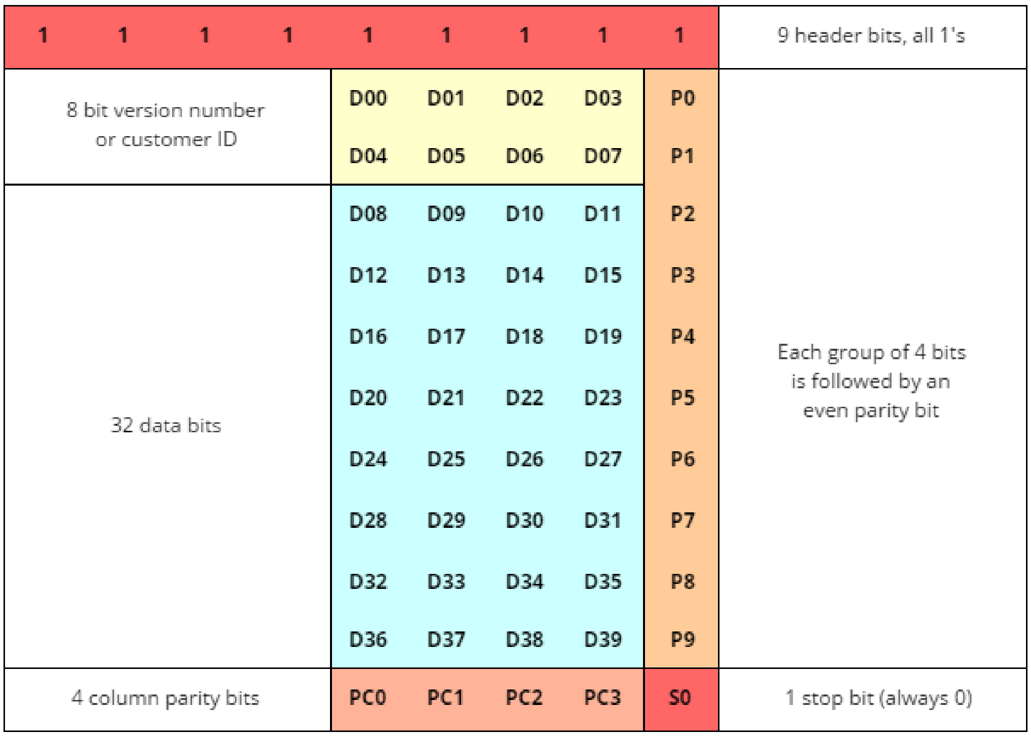
An important distinction can be made between different RFID cards: cards with a form of read-only memory cards containing a unique identifier, and smart cards, which as described earlier often have a microprocessor that protects personal information and secures transactions.

The client has provided us with an RFID scanner and a dozen RFID tags that can be used for prototyping purposes. The scanner and tags make use of a protocol called EM4100 by EM Microelectronic. These tags are read-only and do not have any form of encryption or private data.

EM4100 Tags

The EM4100 tags are read-only, which means that once data has been written to the tag, it cannot be changed. When the tag enters the electromagnetic field transmitted by the scanner, it uses the power generated from this field to transmit the data on the card repeatedly.

Such a tag can hold 64 bits of memory. Figure C.1 shows the structure of the data on a tag.



1 1 1 1	1	1	1	1	1	9 header bits, all 1's
8 bit version number or customer ID	D00	D01	D02	D03	P0	Each group of 4 bits is followed by an even parity bit
	D04	D05	D06	D07	P1	
32 data bits	D08	D09	D10	D11	P2	
	D12	D13	D14	D15	P3	
	D16	D17	D18	D19	P4	
	D20	D21	D22	D23	P5	
	D24	D25	D26	D27	P6	
	D28	D29	D30	D31	P7	
	D32	D33	D34	D35	P8	
	D36	D37	D38	D39	P9	
4 column parity bits	PC0	PC1	PC2	PC3	S0	1 stop bit (always 0)

Figure C.1: Overview of the EM4100 memory implementation [11].

The row parity bits (P0, ... , P9) and the column parity bits (PC0, ... , PC3) allow for error checking within the scanner. When two tags are held against a scanner at the same time, the error checking capabilities within the protocol will prevent a faulty read with very high probability. However, this comes at a cost: of the 64 bits of readable memory, only 40 of those bits are usable data.

Scanner

The scanner is a device made by AuthenTec, Inc. It identifies itself as a Human Interface Device of the interface protocol Keyboard. When a tag enters its range, it can read the data in the tag and inputs it as if a keyboard had input the data. For example, if a tag contains the data "0004749726", then the device will input the corresponding keystrokes followed by a carriage return. The scanner has an advertised range of 3 to 10 centimeters, however, in practice, we have found that it can only read our tags from a distance up to four centimeters.

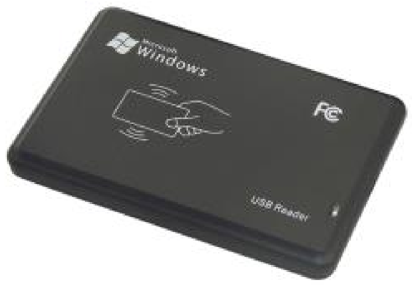


Figure C.2: Concept of the entire Arena.

Prototype Card System

The decision to use these EM4100 Tags instead of smart cards is for prototyping purposes. Due to the limited time available, the tokens will be sufficient for identification purposes. Chapter 5: *Security* has a dedicated section which discusses security and how smart cards can be used to safely make financial transactions, as the RFID card system is not secure enough to use for financial transactions.

C.2 Programming Implementation Decisions

The membership card system will consist of different applications that communicate with each other. For example, the Check-in System will communicate with the Authentication Server. First, the decision for using HTTP as the communication protocol is described. After this, the decision to use REST instead of SOAP is discussed.

C.2.1 HTTP

Communication between computers or devices can, for applications, be done using TCP or UDP sockets. Both are able to send packets of data, but only TCP guarantees delivery of the packet from sender to receiver, making TCP more accurate but slower than UDP [22]. As reliability is one of our design goals, TCP will be used as this will guarantee reliability of transmission.

Another design goal is the maintainability of the system. The system should be usable with a variety of other technologies if the system is extended. For this purpose, the HTTP protocol is used. The HTTP is the standard protocol used on the web [22].

HTTP is a protocol built on top of TCP as a simple request and response protocol to retrieve information from a server in a stateless manner [22]. After a request has been made and a response has been given, the connection between client and server is closed. As our system uses single request for data or verification, the stateless nature of HTTP is sufficient.

HTTPS is an extension on top of HTTP which uses TLS to encrypt the connection [22, 35]. This extension should always be used, as it improves security by preventing third parties from reading or modifying the transmission, which is another part of our design goals.

C.2.2 SOAP or REST

To improve maintainability of the system an additional communication protocol on top of HTTP is preferred to set a model for communication between client and server. Two different categories of protocols can be distinguished: SOAP based, and REST based protocols [62, 2]. We explore them in this section to determine which protocol best suits the requirements.

SOAP

SOAP, or Simple Object Access Protocol, is a protocol that uses XML messages to communicate between the server and the client [4]. SOAP defines the structure of a message from the sender to the receiver. A SOAP message contains the instruction for the server, and the server then dispatches the message as a service invocation to an appropriate server-side application providing the requested service [9].

REST

Representational State Transfer, or REST, is an architectural style described by Roy Fielding [62, 13]. Strictly speaking, REST is not a protocol but a design philosophy for creating simple, clean, and portable web-based applications [4]. REST looks at what makes the World Wide Web scalable, and provides an abstraction on how to interact with the web [62]. An important part of the REST system is that the client does not instruct the server; it just requests resources.

REST rejects the conventional sense of an application programming interface, where a client invokes processes on the server. Instead, the client sends a resource identifier in the form of a URI identifying the resource it wishes to add, view, or modify and provides the necessary information within the body of the URI to complete the request [4]. The only actions specified through a basic REST request are the standard HTTP methods: GET, PUT, POST, DELETE.

Consideration

SOAP tightly couples the client with the server [62]. It has a strict contract between client and server, and if either fails to follow the contract it is expected to break. SOAP specifies built-in error handling so the client is notified of any false requests or errors on the server side. REST is more loosely coupled, allowing the client to continue even when the service is unavailable [45].

Another downside of SOAP is that it requires the client to know the full specification of the server to be able to make requests. This results in increased complexity for developers using the system, and adds additional overhead in the system.

In contrast, REST is easier to use and overall a lot simpler. REST emphasizes, among other things, scalability of component interactions, generality of interfaces, and independent deployment of components. Resources can be found at their specific URI, and a small set of methods is available for the develop to view and manipulate these resources.

Different papers have been published showing how REST and SOAP compare to each other [28, 34] in terms of performance. The papers test for example latency, throughput and packet size. The conclusion is that REST outperforms SOAP. An argument for this result is given in the fact that SOAP uses XML for its communication, which has a significant overhead over standard HTTP requests [34].

From these findings, we have decided to use REST as the protocol we want to work with. The primary reason for us to use REST is its simplicity and because it does not require us to work with a new protocol. Although SOAP provides a more reliable solution, it also tightly couples the client and server, and requires strict specifications to be written. REST in contrast is designed to be expanded upon over time, which is exactly what we want.

C.3 Programming Languages

With the timespan of only ten weeks, we did not want to use any programming languages which we were not confident with in using. By doing so, we can focus on the actual project instead of learning a new language. This left us with a decision between Java, JavaScript, and PHP as the primary programming language.

One of the design goals was to make a reliable system. For this purpose we wanted to use a programming language that is statically typed, where the compiler checks if the code contains type errors [50]. This helps in reducing the chances of errors occurring in the system from mistakes made by the developer. JavaScript and PHP are dynamically typed languages, and were therefore not an option.

TypeScript [31] is a typed superset of JavaScript that compiles to JavaScript. It introduces a type system to JavaScript which helps developers avoid many common errors and thereby increases developer productivity [29]. TypeScript is also easy to learn for developers who are familiar with JavaScript [31].

Both Java and TypeScript provide the functionalities needed to build the system. Java runs in the Java Virtual Machine and JavaScript (and therefore TypeScript) runs in the V8 engine [20], which means both programming languages can run on any machine.

In our experience, neither Java or TypeScript had a clear benefit above the other, which left the decision to how much experience we had working with the languages. Because we have already created web applications in TypeScript prior to the project and setting up the workflow required little effort, we decided to use TypeScript as the main programming language.

An additional benefit is that TypeScript can be used both client and server side when building web applications, meaning that the programming language is consistent in throughout the systems.

C.4 Other frameworks

After the language of choice was determined, we needed to choose the frameworks we would use. Making use of frameworks lets programmers focus on tasks instead of on reinventing the wheel [55]. Since these frameworks will become a vital part of the product, it is important that they are considered carefully.

C.4.1 Testing

When choosing a testing framework, two frameworks were considered: mocha [33] and Jasmine [46]. We used comparison blog posts [56, 17] to list the advantages and disadvantages of both. Both posts conclude that "both frameworks are perfectly able to do the job" and "you can't really go wrong with either choice". Since our team had more experience with mocha, it was chosen as our testing framework.

The comparison articles name sinon [52] in combination with mocha. Sinon's popularity [41] was a vital factor when compared to other frameworks like testdouble.js [42]; with 3.2 million downloads in the past month at the time of writing, a community can provide answers to questions we might have.

As for generating coverage reports, again we looked at downloads in the last month as popularity statistics: istanbul's 3.3 million [39] is very high compared to the alternatives: blanket's 43 thousand [38] and jscover's 1906 [40].

C.4.2 Backend

Servers

As stated in section C.3: *Programming Languages*, the programming language used for the system will be TypeScript. Node.js [16] was used on the servers because it has a large ecosystem for server applications [10, 3]. Node.js' standard library means developers do not need to implement basic functionalities, and where that is not enough, it has a rich selection of community libraries which can be used [43]. One of these libraries is Express [25], a web framework which provides features aimed towards creating an API easily and rapidly.

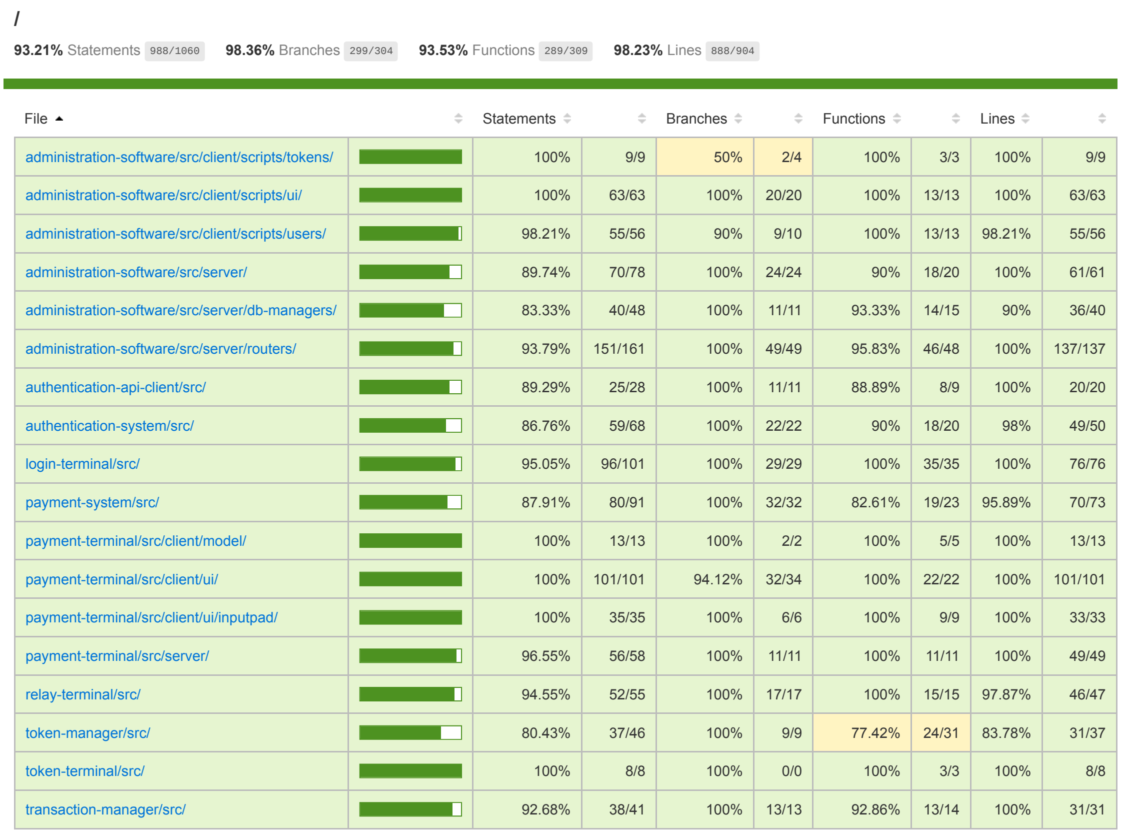
Database

The client's existing database server ran on MariaDB. Because MariaDB contains all required functionalities and because libraries for node.js exist, there is no need to change the database infrastructure the client is using.

Front-end Applications

Because the specifics of the platform on which the front-end applications will run are unclear, we chose use web frameworks. Web frameworks provide us the option to do rapid prototyping and development without the need to keep a specific platform in mind. Our target browser is Chrome, which runs on all commonly used devices [21] such as Windows, Mac, Linux, Android 4.1+ and iOS 9+. The application will be able to run in other browsers, but by selecting Chrome as target platform we can focus on getting it to work within a specific application and can be sure that the future target device supports this.

D. Istanbul coverage report



/

93.21% Statements `988/1060` **98.36%** Branches `299/304` **93.53%** Functions `289/309` **98.23%** Lines `888/904`

File		Statements		Branches		Functions		Lines	
administration-software/src/client/scripts/tokens/		100%	9/9	50%	2/4	100%	3/3	100%	9/9
administration-software/src/client/scripts/ui/		100%	63/63	100%	20/20	100%	13/13	100%	63/63
administration-software/src/client/scripts/users/		98.21%	55/56	90%	9/10	100%	13/13	98.21%	55/56
administration-software/src/server/		89.74%	70/78	100%	24/24	90%	18/20	100%	61/61
administration-software/src/server/db-managers/		83.33%	40/48	100%	11/11	93.33%	14/15	90%	36/40
administration-software/src/server/routers/		93.79%	151/161	100%	49/49	95.83%	46/48	100%	137/137
authentication-api-client/src/		89.29%	25/28	100%	11/11	88.89%	8/9	100%	20/20
authentication-system/src/		86.76%	59/68	100%	22/22	90%	18/20	98%	49/50
login-terminal/src/		95.05%	96/101	100%	29/29	100%	35/35	100%	76/76
payment-system/src/		87.91%	80/91	100%	32/32	82.61%	19/23	95.89%	70/73
payment-terminal/src/client/model/		100%	13/13	100%	2/2	100%	5/5	100%	13/13
payment-terminal/src/client/ui/		100%	101/101	94.12%	32/34	100%	22/22	100%	101/101
payment-terminal/src/client/ui/inputpad/		100%	35/35	100%	6/6	100%	9/9	100%	33/33
payment-terminal/src/server/		96.55%	56/58	100%	11/11	100%	11/11	100%	49/49
relay-terminal/src/		94.55%	52/55	100%	17/17	100%	15/15	97.87%	46/47
token-manager/src/		80.43%	37/46	100%	9/9	77.42%	24/31	83.78%	31/37
token-terminal/src/		100%	8/8	100%	0/0	100%	3/3	100%	8/8
transaction-manager/src/		92.68%	38/41	100%	13/13	92.86%	13/14	100%	31/31

Figure D.1: Istanbul coverage report for all our code.