Feedback-Analytics

Carlos Brunal
Daryll Lobman
Saeed Najafi
Sander Oostmeyer
Renzo Russel

Technische Universiteit Delft

TU Delft

Feedback-Analytics

Carlos Brunal
Daryll Lobman
Saeed Najafi
Sander Oostmeyer
Renzo Russel

July 7, 2019

1

Preface

This is the report for the completion of our Bachelor degree in Computer Science. It documents a ten week project for the start up company Feedback-Analytics. The project involved a full design and development of the Feedback-Analytics web-app. In this report, we will go over our progress during this project. Our achievements as well as our shortcomings during the implementation of the feedback-analytics system. Regardless of any shortcomings, we still believe our final product holds up to our original ambitions.

We like to thank our supervisor at Feedback-Analytics, Freek van Vliet for giving us the opportunity to work on the project, for believing in us and for creating a good working environment where all of our needs were met.

A Special thank you to our project coach, Mitra Nasri who kindly helped us through the process and coming up with possible improvements on the product.

Contents

2

Summary

Feedback-Analytics strives to have a web application which can help companies to find problems within their company by analyzing consumer reviews. The way this should be done is by providing a web application, where client companies from Feedback-Analytics can create an account. After the account is created the company is able to create questionnaire forms. The company should be able to choose from several type of questions, e.g. being a text question, number question, email question, phone number question, and more. These forms are to be answered by customers and these answers should be stored into a database which is connected with the web page. Aside form collecting data these forms should be able to have branching paths meaning that each question can have different follow up question depending on the answers.The data from each questionnaire will be analyzed by the company in the web page. Answers for each question will be displayed displayed with various charts(e.g. , pie charts, bar graphs or box plots), depending on the types of questions. As this web application is built from the ground up our project group needs to create the database, setup the server, create a web page with a functioning front end and connect it to the database.

3

Introduction

3.1. Context

In today's world, it is important for companies to have a good public image. It is an important factor in order for any company to grow and have a higher customer satisfaction. Achieving these goals requires building trust and authenticity between the clients and the brand, and offering high-quality product and services. Feedback from customers is, therefore, a valuable asset to highlight areas which can be improved for companies. Since they can use this feedback to improve their services and ultimately increase customer satisfaction, which in turn, decreases the number of negative reviews. Negative reviews can potentially damage the companies reputation.[9] Often, it is in the best interest of a company to improve its customer service wherever possible. This project has been set up by Feedback-Analytics, which is a start-up company that offers services to other client companies to improve their customer feedback and at the same time create more positive reviews. Therefore, customer satisfaction can be improved.

3.2. Project

To provide a service to companies, Feedback-Analytics has started this project to create a system where companies can create an account with which they can create a customized questionnaire for their costumers. The system will allow companies to create questionnaires.

These questionnaires will support branching paths. Questions can have different follow-up questions depending on answers given by the consumer. The results of the questionnaire will be compiled and presented in a clear manner to the clients.

4

Problem Definition

4.1. Problem Decomposition

Feedback-Analytics is web application that aims to help companies identify internal problems and help improve customer satisfaction. The purpose is to create a platform that allows companies to visualize and eventually analyze costumer feedback through the use of dynamic forms. The following graphics shows a clear distinction of what we consider to be the input (forms) the system takes, and how this is converted into the desired output (product).

The first step is to gather data from the forms, this data is to be analyzed (e.g. to find correlations and patterns). The analysis of the data is ultimately the end product that we are interested in delivery to companies making use of this software.

For clarification purposes we define 3 parties:

FA (Feedback-Analytics)

Feedback Analytics is both the name of our client and their software. This software will be provided to companies (clients of Feedback-Analytics)

Company

The companies that Feedback-Analytics provides the software for.

Consumer

The users that give feedback to a service or product they consume from a company.

4.2. Problem statement

Who The FA system will target companies that are interested in improving both their online presence, their customer support infrastructure and also the productivity of their employees.

What The importance of consumer online reputation (for both returning and new costumers) cannot be ignored and can be an even bigger influence to the company's profit than other traditional factors (e.g. quality, price, marketing)[9].

Why Costumer feedback goes hand in hand with online reviews. Identifying and addressing negative feedback is important not only for the company's public image but also to help identify what internal problem may need to be address within the company.

How FA aims to relieve these hurdles by making it easier for companies to do the following:

- Making specific forms for consumers, tailored to each of the services and/or products the company provides. e.g. each company can make a form they can send to a costumer after said costumer has been helped to measure consumer satisfaction. The data from this form will be stored for further analysis.

- Possible prevention of bad reviews, e.g. Depending on how the form was filled, if the system "notices" the consumer is unhappy, he or she can be asked if they want to be approached by the company to be helped solve the issue.
- Visualize the gathered data using various charts to identify areas of possible improvement within the company. E.g. gather data of shipments that are not arriving on time, or a member of staff that is often late to appointments with consumers.

4.2.1. Use Case Diagram

We need to have a clear objectives for our software. Therefore, Figure 4.1 represents a high-level overview of some of the most fundamental goals a company has, when using our software, from creating, archiving, viewing a form to sending it to a consumer and fetching the results so they can analyze them. Figure 4.1 has 4 main components, these are:

- **Actors**
- **Use Case**
- **Relationship**
- **System Boundary**

Actors Actors are individuals which are involved with our system, based on the role they have. In our software we have 3 mains actors:

- **FA**
- **Consumer**
- **Company**

4.3. Requirements

4.3.1. Functional Requirements

With the help of the project giver, a clear Moscow list was made. Naturally The project giver wanted to add as many features as we could implement. This lets us and him know know exactly which feature of the software will not be implement during this project. The "Wont have" points were all mutually agreed upon.

Must Have

- **Create Company**: FA must be able to create a new company with an admin user.
- **Company Login**: A company must be able to login to the dashboard.
- **Create Form**: A company must be able to create a new form.
- **Edit Form**: A company must be able to edit a form if it's unpublished.
- **Form Logic**: A company must be able to add logic jumps to the form questions when editing.
- **Test Form**: A company must be able to test a form before publishing it.
- **Form URL**: A company must be able to get a link to a published form for sending to a consumer.
- **Publish Form**: A company must be able to publish a yet to be published draft form.
- **Archive Form**: A company must be able to archive a form if it has been published.
- **Delete Form**: A company must be able to delete a form.
- **Form Answers**: A company must be able to see all the answers for any specific form.
- **View Form**: A consumer must be able to view a form.
- **Fill Form**: A consumer must be able to fill in a form and submit it.

Should Have

- **Form Data**: A company should be able to visualize response data for any form.
- **Querying Data**: A company should be able to query form response data to help with analyzing the responses.
- **Create Users**: A company must be able to create new users.
- **Delete Users**: A company must be able to delete users.
- **Edit User Rights**: A company must be able to edit an user's form editing rights (e.g. admin or read-only).
- **Copy Form**: A company should be able to to create a new form based on an old form.
- **Send Form**: A company should be able to send a form to a consumer via email.

Could Have

- **False Form Alert**: A company could reject any suspiciously filled forms (e.g. a form that has been filled in in too fast).
- **Automated Email**: A company must be able to send automatic email to consumers with a link to their form.

Won't Have

- **Welcome Section**: A company won't be able to add a welcoming section to the forms when creating them.
- **Edit Form**: A company won't be able edit any published form.
- **Styling Form**: A company won't be able to add markdown styling to forms.
- **Theme Styling**: A company won't be able to set a company-wide theme.
- **Analytics in PDF**: A company won't be able to create a PDF report of all the response data of a form.
- **End Section**: A company won't be able to add a end section to the forms when creating them.

4.3.2. Technical Requirements

Must Have

- **Testing** Our front-end code must be properly tested with unit tests. (at least 70% average code coverage)
- **CI** Our project must use Continuous Integration to keep our master branch clean and working at all times.

Should Have

- **CD** Our project should have Continuous Deployment to keep the live version up to date with all of our changes.
- **Code Quality** Our code should quality should get a good rating from SIG (a score greater than 3.5).

Could Have

- **Integration Tests** Our project could have automated integration tests to ensure its components work correctly.

Won't Have

- **E2E** Our project wont have automated End to End tests to ensure the entire system runs smoothly.

4.3.3. Use case

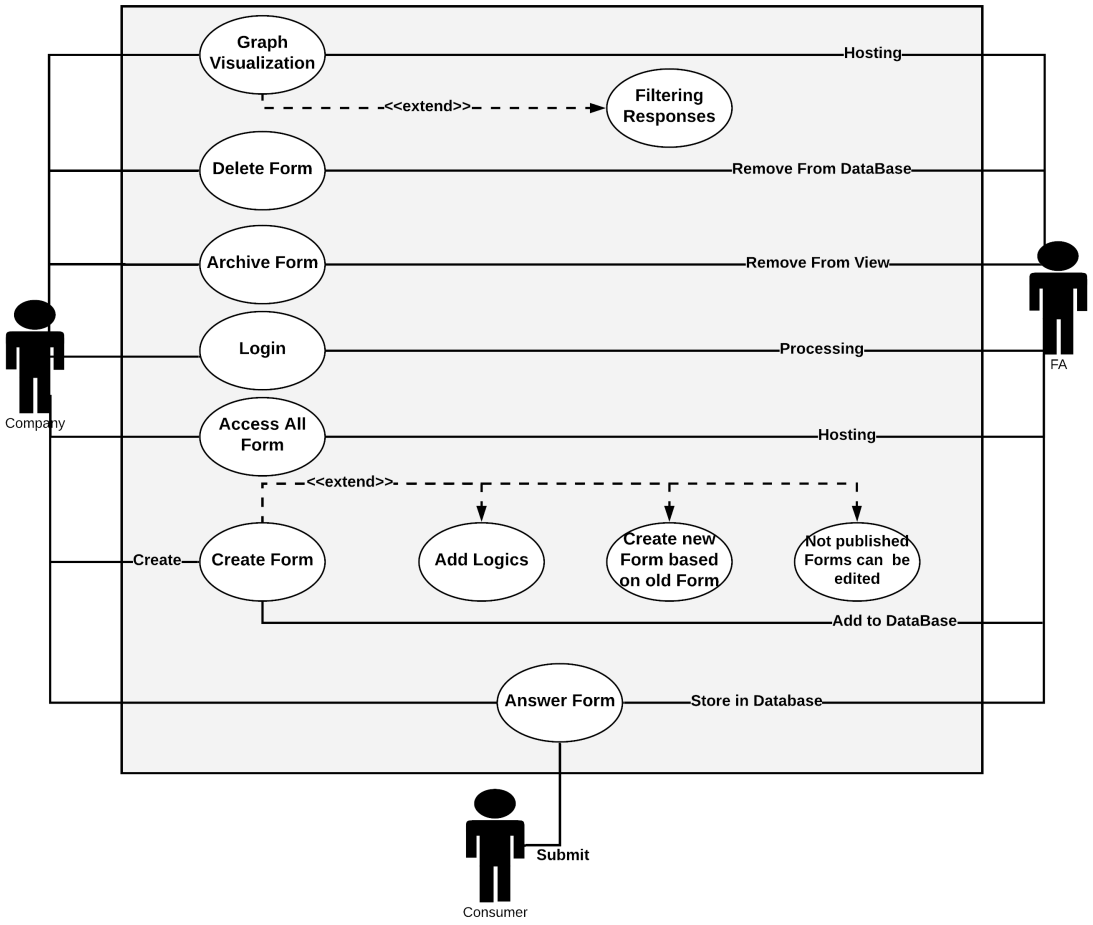

Figure 4.1: Use Case Diagram.

The main user of the web app, is the company, the company makes the forms. The consumer fills in this form which becomes the input for FA to store in the database.

We can split this model into 2 sections: front-end and back-end. Front end handles anything that has to do with what the company sees and works with, e.g. Login pages, Form creation, graph visualization, ect. The back-end is what FA does in the background with the companies input, e.g. server requests , querying the database.

In the next section we will discuss different technologies and explain which ones we chose and why our choices fit the project better.

5

Technology Exploration

When it comes to technologies like frameworks and languages, one could go on endlessly about how many choices there are. In this section we will be going over the current state of the art in technologies for web development. However, we will only discuss the choices we considered the most, that is, what believe to be best fit for this type of project.

5.1. Web development: React vs. Angular

Angular

"Angular is a TypeScript based open source web application, it is a platform that makes it easy to build dynamic applications for the web "[1]. "Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges" [4]. "Angular empowers developers to build applications that live on the web, mobile, or the desktop. With angular, developers wont need to start any development from scratch, everything is transpiled down to JavaScript 5 from higher level languages" [3].

React

React is a JavaScript library for building user interfaces [2]. React allows developers to create large web applications which can change data, without reloading the page. React aims to be fast, and simple. It can be used in combination with JavaScript libraries or frameworks such as Angular [13].

It is not so easy to compare Angular with React, and the reason is that in certain places they overlap with each other in term of what they can do for programmers (Framework and Library). In the following section we try to compare these two, together, using some important key features of these languages.

The differences

Library vs. Framework Angular and React have similarities , but they differ in some key ways. Namely, that Angular is a full-fledged framework as opposed to React which is only a JavaScript library [7]. This makes setting up a new project a very "plug-and-play" process.

Angular as our Choice We decided to work with Angular, because it is a framework. This allowed us to jump right into development instead of spending a lot of time setting up the front-end environment. Because of this reason and many more we decided that Angular in our case is the best choice for this project.

5.2. UI libraries

For development of the front end two libraries where chosen Angular material and material design Bootstrap. These two libraries offer predefined elements which can be easily implemented in angular, for example buttons, drop down menus and charts. This will make development easier and gives the opportunity to focus on other parts of the project. The reasons why both libraries were chosen and what they have to offer are as follows.

Angular Material
Angular material is a library for angular which offers many predefined elements consisting of form controls, navigation, layouts, buttons, modals and data tables. These components are already tested among all the modern browsers. In addition these predefined elements can be styled to our liking giving a great flexibility for creating a front-end. Furthermore, angular material is specially made for angular, so that any predefined Angular Material element can easily be integrated in angular components. But above all, the style and look of the predefined elements are our liking and Angular Materials is therefore chosen as the main UI library for the front-end of this project.

Material Design Bootstrap
Material design bootstrap offers more predefined elements than Angular Material, but the looks are less to our liking for this project. However Material Design Bootstrap offers charts while Angular Material does not and therefore Material Design Bootstrap will also be used, but only to implement charts which are used in this project, such as the piechart, linechart and horizontal barchart.

5.3. Database: SQL vs. NoSQL
In this section we will go over two main types of database structures, SQL and NoSQL, and why we chose to use SQL (MYSQL) for our project.

5.3.1. SQL
SQL databases are tried and true know to be powerful: SQL to this day the most used database structure used, this makes it a logical choice for database implementation and and great for for complex queries. On the other hand, it can be limiting. SQL needs the designer to have enough knowledge of databases to start building a database, you need to already have a n idea or plan for a schema to successfully making the structure of your data before implementation. Once this structure is designed, the data that follows/enter this database must be the same structure or it will not work. Because of this requirement, any changes to the design or the structure of the database can be difficult or troublesome for the database. Preparation is needed.

MySQL Advantages
- Maturity: MySQL is an long time well established database, plenty of support and extensive testing history this also makes it very stable.
- Compatibility: MySQL is support on all operating systems and has plugins to many languages including Node.js which we use in our project.
- Open source: its free and open source.
- Replicable and Efficient: The MySQL database can be replicated across multiple nodes, meaning that the workload can be reduced and the scalability and availability of the application can be increased.

5.3.2. NoSQL
NoSQL databases are made of off dynamic schema. This mean that data can be store in different way other than just tables (columns and rows). This simply means it it very flexible in term of structuring and planning. You can create documents without having to first design exactly how the structure should be. This also means that since there is no set structure, the dynamic nature of NoSQL allows one to use different syntax in the same database[14].

NOSQL advantages
- Dynamic schema: extremely flexible nature to change the data schema without modifying any odata already stored.
- Scalability: NOSQL is horizontally scalable, which helps reduce the workload and scale the database in case of big increases of data.
- Manageability: no as restrictive as MySQL regarding admin rights, it can be used by both developers and administrators.
- Flexibility: addition of new columns or fields without affecting existing rows.

5.3.3. MySQL as our choice

For our database we wanted to have a predefined robust structure. We also put an importance on consistency. We have a clear vision of what each form should contain and also we want to assure that after updating any form data, that queries immediately show the updated timestamps of these changes. Since data analysis is the most important aspect of the system, it would be absolutely unacceptable to continue showing e.g. outdated dates or answers any time after something has been modified.

The database has to guarantee to the utmost degree that the data stored is up to date and correct, not ambiguous in form or way. Taking scalability into account, any future addition to the database are expect to be solely column based, or relation to the existing structure already in place. This si why we found SQL to be a better fit for our system.

5.4. NodeJS - Environment for javascript in the back-end

5.4.1. The main idea behind Node.js

The main idea behind Node.js is to perform data-intensive tasks in an efficient manner and to handle a large amount of concurrent connections in real-time. Therefore, "The main idea of Node.js is to use non-blocking, event-driven I/O to remain lightweight and efficient in the face of data-intensive real-time applications that run across distributed devices."[8]

5.4.2. Node.js in our project

Our project preforms many data-intensive tasks when companies create, edit and save forms and when consumers answer the forms. Node.js allows many consumers to concurrently answer forms.

The Node.js Web API acts as the middleman between our relational database and the front-side of our project. This plays well with the advantages of Node.js because the API only does light manipulations of the data coming from the front-end and the database.

5.4.3. Common Node.js pitfalls

Node.js is not good for 2 operations. Blocking operations and CPU-Intensive operations and using Node,js for these purposes would completely negate all of its advantages.

Node.js runs on a single thread and doesn't have the option to multi-thread for certain operations, which means that if a certain operation is synchronous and takes a long time, then it will keep all the other operations from being completed.

5.5. JWT vs Sessions - How They Work

In this section we will discuss JWT and Session Cookies, their advantages and disadvantages and why we chose to use JWT.

5.5.1. What is JWT

"JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted." [12]

5.5.2. JWT in our project

JWT contains a payload that can be verified by the server/resource that receives the JWT. This means that company data, like their username, their permissions etc. can be retrieved from the database once and stored inside a JWT and there will be a certain guarantee that the data inside of the JWT is unaltered. So, by uing JWT to authorize companies, we can have them use certain resources and immediatly know what they are allowed to do, no call to the database necessary. This lowers the strain on the database. Our JWTs also have a short "alive" time in order to prevent abuse by people who might somehow get a hold of the JWT.

5.5.3. Problems with JWT

Logging out with JWT is not difficult, because one can just delete the JWT from the front-end, but that means that the JWT is still valid and could be used by hackers to abuse our system. We think that this scenario is

unlikely, but in the future we plan to add a JWT blacklist to prevent unexpired JWTs from being used.

5.6. Password storage

For this project it was necessary that users have login functionality in order to protect resources that only certain users have access to and to make sure that only users with the correct access rights have the right to certain actions. Therefore security is an important part of our project and especially password security seemed important. In this chapter we will discuss our choice of password hashing algorithm.

5.6.1. Enter BCrypt - A hashing module that scales with computing power

NPM(Node Package Manager) is a manager for Node.js that allows developers to install code snippets that serve a specific function. One of the snippets/packages that we decided to use in our project is Bcrypt - a password hashing algorithm. Bcrypt is based on the Blowfish algorithm and was introduced in 1999 at USENIX. [5]

"Blowfish is a symmetric-key block cipher, designed in 1993 by Bruce Schneier and included in a large number of cipher suites and encryption products. Blowfish provides a good encryption rate in software and no effective cryptanalysis of it has been found to date." [6]

The blowfish algotithm is known for its expensive key setup. It uses some subkeys in their normal state to create a hash and then uses this hash to replace some of the subkeys. this process repeats until all the normal subkeys have been replaced. [5]

what sets Bcrypt apart is that it also uses salt in the hashing process and that it can perform multiple salt rounds. How this works is simple. The blowfish algorithm alternates between the real password/string given and the salt string generated by the Bcryt module. The salt is generated by taking some random data(e.g some info from the system kernel etc) and creating a random salt, this is to prevent rainbow table attacks. Furthermore, the salt rounds specified force the algorithm to hash exponentially longer per added round, which prevents brute force algorithms from easily getting the original password back.

5.6.2. Bcrypt as our choice

In the end we first hash passwords using Bcrypt and then we store the hashed passwords in the database. In the case of a password leak the attackers will be forced to brute force all the passwords one by one giving us enough time to warn users to change their password.

5.7. Summary of Choices

To reiterate our choices:

- Web development: Angular for its flexibly and out-of-the-box libraries. It can fasten the web development process significantly.
- Database: SQL for its wide support, robust structure and reliability/stability.
- Javascript environment: Node.js for its wide support, documentation and scalability in WEB development.
- Authentication: JWT for the ease in how resources can be managed and lower strain on the database.
- Security: Bcrypt because of its ability to scale with computation power.

6

Process

6.1. Planning

Planning For the first two weeks of our project, our team focused solely on researching the given problem. E.g. It's definition, possible tools, and possible solutions. The former is especially important as we needed a crystal clear understanding of our deliverables before we could properly start development. Our target was to have weekly meetings with our coach, but due to scheduling constraints we ended up keeping it to bi-weekly meetings. We took an Agile approach to our development method, using SCRUM with the help of the SCRUM management tool Jira. (See our sprints in Appendix A.)

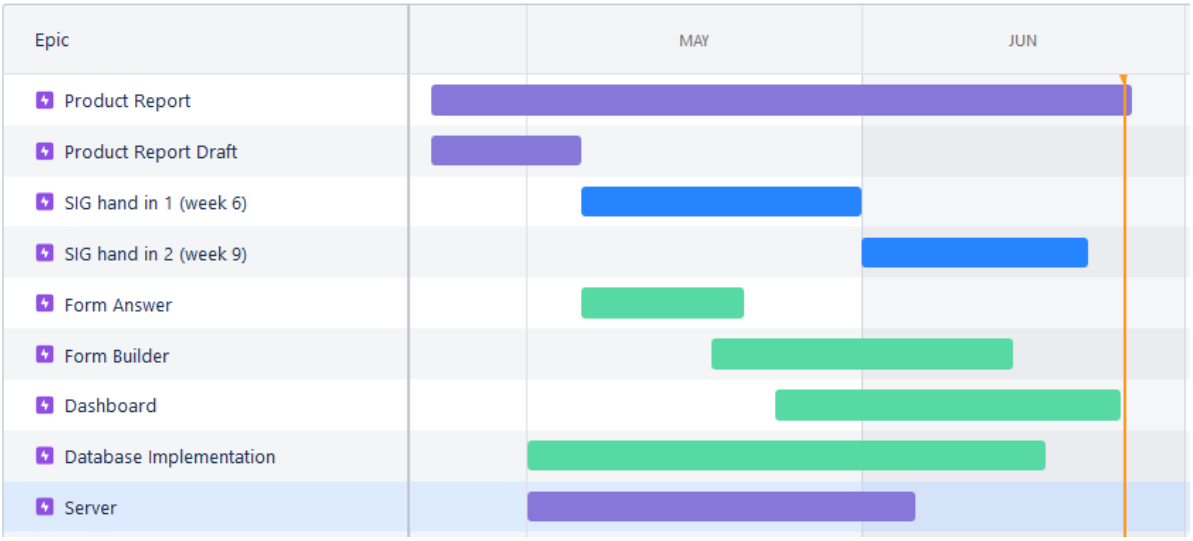

Figure 6.1: Project Roadmap.

Roadmap In Figure 6.1 it's clear that out main focus early on was to split the work in two parts, back-end and front-end. Having the back-end and front-end being worked on concurrently ensured that back-end developers could communicate with front-end developers to ensure compatibility with their code and requirements. This way we could mitigate any backpedaling in cases where the front-end and back-end didn't line up. (E.g. needing certain data to be stored / fetched which wasn't.)

Office space The company (Feedback-Analytics) provided us with an office workspace, as well as workstations so we could all work together in one place. This really boosted the intra-team communication. Because of this provided setup we planned to meet up a minimum of 4 days a week at the office from 10am to 6pm.

6.2. Testing

Jasmine & Karma On the front-end we used Jasmine & Karma for testing. We chose them because they are the default testing setup that Angular uses.

1. **Jasmine** The testing framework used.
2. **Karma** The test runner used to let us test components in the DOM.

Types of tests The front-end has both unit tests and integration tests. Thanks to Karma we could test whether our Angular components actually worked with our code or not. We do not have automated end-to-end tests. With the back-end being under constant construction meant all our tests would be completely flaky so we decided they weer more trouble than they're worth.

7

Design & Implementation

In this chapter we'll discuss implementation as well as design specifics for the final delivered product. In particular this means the server side (VPS, database, server API, authentication/authorization) as well as the client side (form builder, form management, submission visualization, form answering)

7.1. Subsystem Decomposition

In this section we will give a global oversight of the entire system's components. Firstly, in the "Definitions" section we will define the different components and their uses. Secondly, we describe the inter-component interactions that the system will require in the "Interactions" section.

7.1.1. Definitions

Questions A **question** consists of 2 main parts, the actual question being asked, and the logic behind it. There are multiple types of **questions**. Depending on the type, a **question** will take different kinds of user input. The logic decides what the following **question** in a **form** is going to be based on the input to none, one, or multiple **questions**.

The different **question** types and their inputs:

- **Multiple Choice**: a list of choices of the predefined text options.
- **Star Rating**: a strictly positive integer representing the amount of stars.
- **Number**: a real number.
- **Yes/No Choice**: yes or no.
- **Text**: plain text.
- **Phone Number**: a valid phone number.
- **Email**: a valid email address.

An example **question** might be:
Q1: Where you satisfied with the service?
Answer: yes / no
Logic:

if $answer = yes$ **then**
  $jump$ to $Q2$
end if

(See Section 7.3.3 for more information on the logic)

Forms A **form** is list of questions, which acts like a directed graph. The vertices represent **questions** and the edges represent possible follow-up questions. Each **form** has exactly one starting **question**, but can have multiple end **questions**.

Consumer-side Interface The **consumer-side interface** is the front-end web interface used for filling in a **form** and submitting it. This is the only interface a consumer interacts with.

Company-side Interface The **company-side interface** is the front-end web interface for managing existing **forms**, creating new **forms**, and viewing/analyzing **form** responses.

Web API The **Web API** is how the front-end interacts with the database. This could be updating / deleting / creating forms, fetching **form** responses, creating / deleting / updating users in the database.

Database The **database** contains **client** accounts, the **forms** they own, and answers to those **forms**, it also processes queries for data.

7.1.2. Interactions

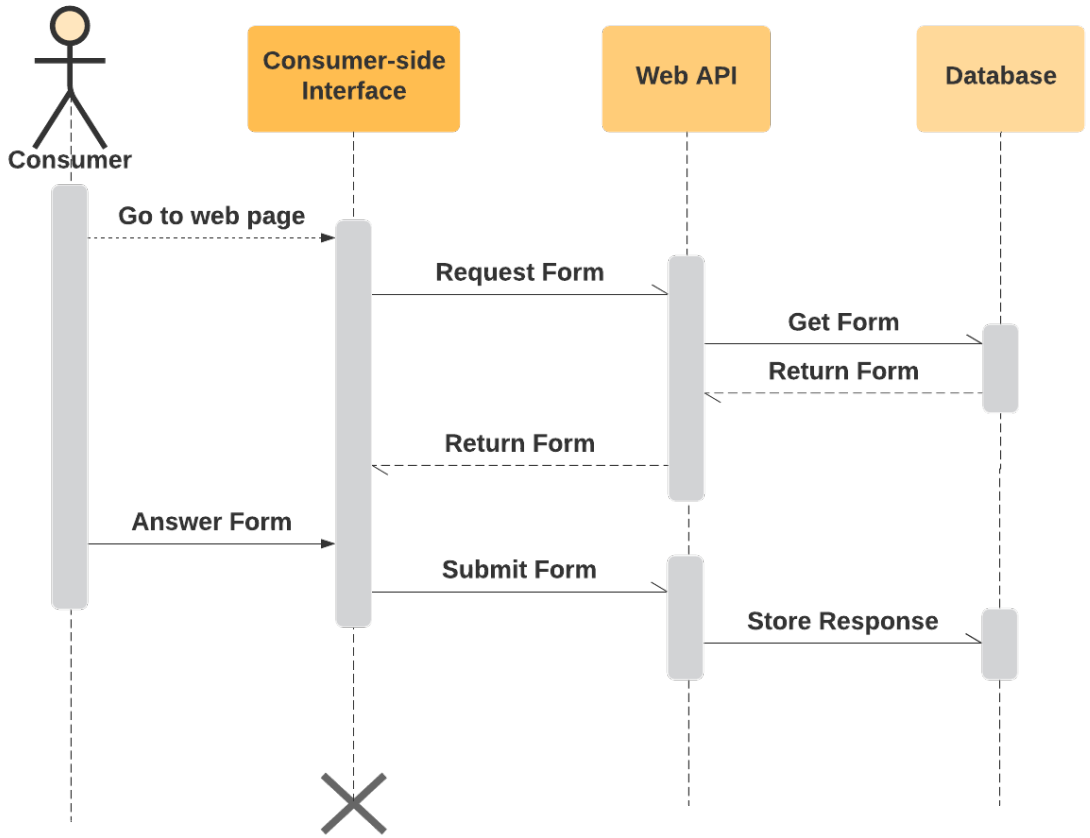

Figure 7.1: A consumer answering a form.

Answering Forms In Figure 7.1 a consumer navigates to our web page to answer a form, they arrive at the consumer-side interface, the interface fetches the form, this gets returned, the consumer fills it in, and lastly the consumer's response gets submitted.

Managing Forms In Figure 7.2 a company navigates to our web page to manage their forms, they arrive at the company-side interface, the interface fetches a list of all their forms, this gets returned, the company can then choose to modify a their list of forms (e.g. creating, renaming, deleting, archiving, publishing, forms), after each modification the form list gets re-fetched from the database so it stays up to date.

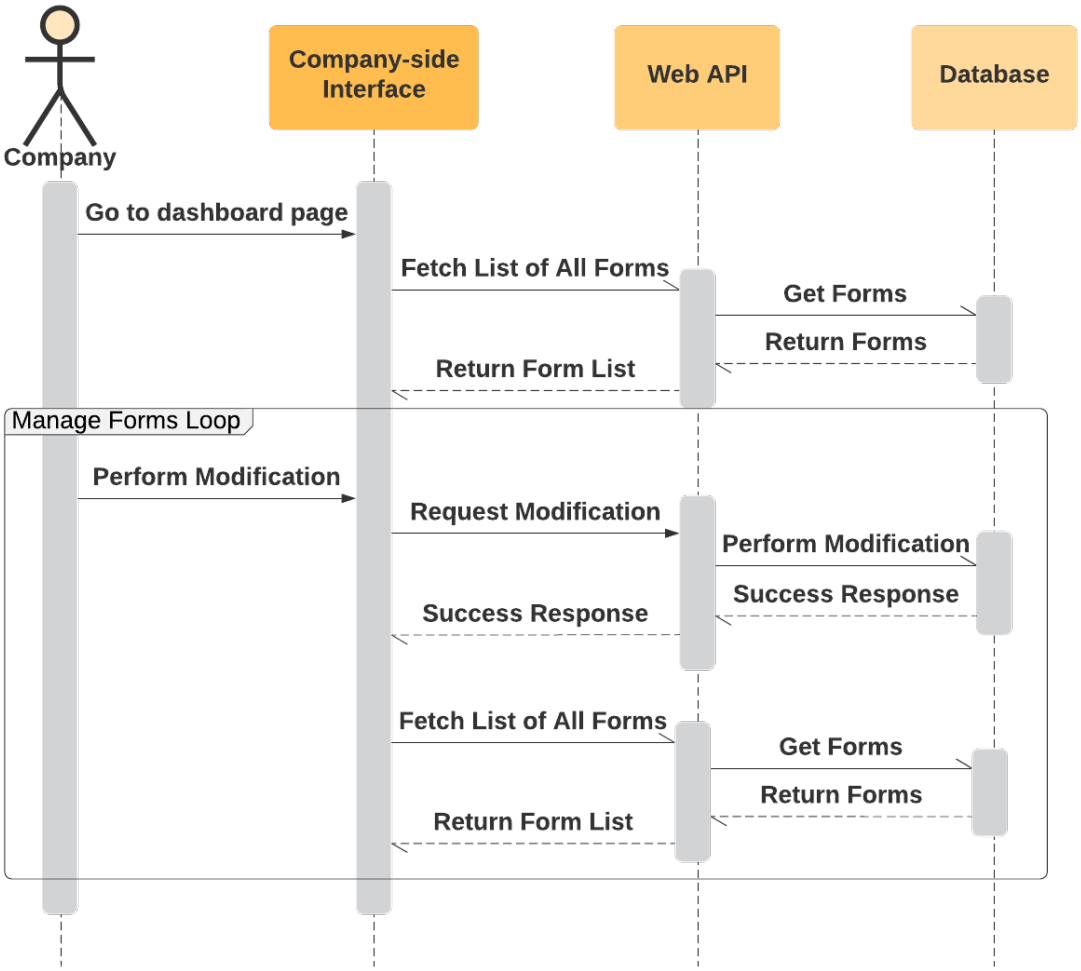

Figure 7.2: A company managing its form.

Building Forms In Figure 7.3 a company navigates to our web page to build a pre-created form. The form gets fetched, the company makes changes, and potentially makes saves. If a save is made the request gets sent to the web API which sends the request on to the database.

7.2. Back-End

The back-end contains all the resources necessary for the application to function. The back-end consists of the VPS, Nginx, the Node.js Wep API and the Database.

7.2.1. VPS - Virtual Private Server

A VPS(Virtual Private Server) is a server that runs on another server. How this works is that a server of approx 40+ cores is split up, using software, into multiple encapsulated servers that run independently of one another. The VPS was bought from TransIP and they have a specific development terminal that is able to connect to the VPS trough their internal network.

installation

We installed Ubuntu 18.04 Bionic Beaver on our VPS. Ubuntu is the most common operating system used to host server applications and that's the primary reason we chose it. Ubuntu 19.04(Disco Dingo) was already released but we still chose to use version 18.04 since there are more tutorials and guides for this version of Ubuntu.

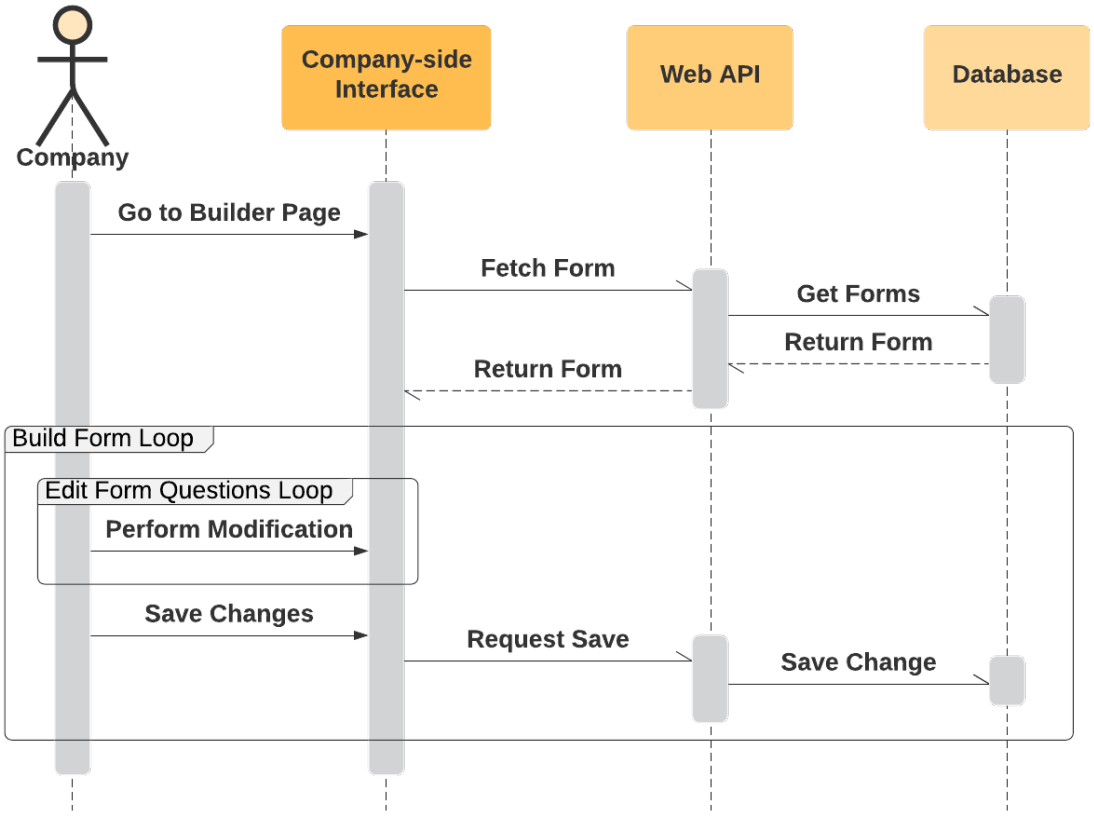

Figure 7.3: A company building a form.

OpenSSH

After installing Ubuntu we enabled OpenSSH on our system. This allows our team to connect to the server from our own terminals on our personal computers. To increase the security of the VPS we obfuscated the SSH port, so we no longer use the default port, and we use SSH keys instead of passwords to connect with the VPS. This prevents hackers from sniffing out the unencrypted password and forces the VPS to only allow developers with the correct private key to connect with the server.

ufw

Now we focused on protecting our system from malicious attackers. We turned on the ufw(uncomplicated firewall) and closed the all the ports(including the default SSH port). The only ports that we left open are the ones necessary for an SSH connection(the obfuscated port) and the port for HTTPS traffic(port 443). We closed port 80 because we do not intend on creating a unsecured website,

Admin(root) Rights

SSH using key verification has the benefit that traffic between the developer PC and the VPS is encrypted. Therefore, one can safely use sudo on the server(sudo allows a normal account, that is in the sudo group, the rights of the administrator(root)). We do not have a root account(an account that anyone can access once and that allows them admin privileges until they log out). We only allow normal accounts that have to specifically ask for root access(this prevents users/devs from changing critical system info on accident and disallows a hacker that gets access to a dev account any critical access since the hacker most likely doesn't know the sudo password.

7.2.2. Nginx - allowing flexibility everywhere

"NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more. It started out as a web server designed for maximum performance and stability. In addition to its

HTTP server capabilities, NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and a reverse proxy and load balancer for HTTP, TCP, and UDP servers." [10]

serving static files with NGINX

We decided to use NGINX in our project because of its ability to serve static files and allow is to reverse proxy to our Web API without needing to give the API its own open port trough the firewall. We use NGINX to serve our static javascript, HTML and css files to the client.

Future Usage

We use NGINX to redirect dynamic content (and requests for dynamic content) to the Web API. Having NGINX in our project allows us to easily add sub-domains to our VPS in the future that can also point to this(or other) Web APIs and this gives us a lot of flexibility to easily improve our service in many different directions.

7.2.3. Node.js Web API

The Node.js Web API was created to handle complex request that could not be handled by simply serving files. These request include authentication and authorization requests, form request and company requests.

Authentication and Authorization

First we need to make a clear distinction between the words authentication and authorization. Authentication confirms a persons identity so that they are allowed access to the system while authorization verifies if a person is allowed to access a resource in that system.

We use passport.js, a very popular authentication middleware for node, in order to authenticate our companies. Companies can simply log in using their email and password, after which the API will hash the password and compare that to the companies hash in the database. If the hashes match then the company is authorized and the front-end receives a JWT [12] that they have to place in the HTTP authorization Header as Bearer Token [11]

We also use passport.js in combination with the jwt-passport module in order to verify incoming JWT. The data contained in the JWT is passed onto the next function in the middleware chain so that it can be decided whether that company has the correct rights etc. to access the requested resource.

Form Request

Our website allows companies to build their own forms and collect the answers to the forms. The API creates a link between the website and the database in order to allow the storage, retrieval, modification and deletion of forms and answers. We use sequelize, a Promise-based ORM, in order to efficiently store and retrieve Data from the database.

Company Requests

In our system, companies will be the clients that pay for the service. So, companies need to be able to add and remove employees and edit the employees permissions. The API allows these changes trough the company routes which are authenticated using passport.js

7.2.4. Database

The design of the database schema was made using MySQL workbench. firstly, to get a feel of how we wanted to store and manipulate the data, which keys and data were constant, and how we could normalize the database for better organization.

Changes The database saw many changes over the course of the project, as we got a good feel of what data was important to query for. whilst designing the data we wanted each user of the system to belong to a company. User belonging to a company would have different right, either admin or read only right. we populated the country and city tables with a complete list of cities from Holland, the project giver is planning only to release in Holland, of course it is possible to populate the country and cities table with a complete list of cities and country's form all over the world but we limited it to National cities for the purpose of this project.

Optimization following 3NF we, worked with primary keys, per table all non-prime attributes depend on the primary key only. we wanted to avoid cases for which table might be populated with null values, e.g. in the answers table. since we have t types f answers tied to a specific from, we split this into two types, int and strings. instead of just having one table for all answers. every answer is then tied to a form by the corresponding primary key. At this stage the database is not and does not need to be to intricate. The structure follow as conservative pattern of keys. Answers tied to forms, forms tied to companies. and user tied to companies.

Implementation The main aim was to implement a database using MySQL from an installed MySQL local server running as a localhost on the VPS server. But we noticed that when the necessity arose to make any changes in queries or parameters to the database, it would mean that we had to manually manage this whit-in the VPS itself. this of cousrse it not a viable option.

Sequelize Conveniently for us, we are working with node.js and thus we have a wide variety of tools to our disposal. We can use an ORM (Object-relational mapping) tool called sequelize, this allowed us to define and create a database by just running typescript. Since we already know with structure our database has we can have the script make each table and they primary key relation automatically right form the code onto the VPS server. This automatized the progress and gave us big flexibility regarding changes to the database.

In the next figure you can find a representation scheme of our current database design, as represent in mysql workbench.

7.3. Front-End

7.3.1. Routing

Modularity We take full advantage of the "lazy loading" capabilities of Angular. Splitting our web-app into modules allows us to keep bandwidth usage down as the user only needs to fetch the needed modules from the host. Another feature of these "lazy loaded modules" is that their child routes are relative. If you were to change the route to a specific module, you wouldn't need to worry about fixing all the internal routing.

Authorization Guarding yada yada we block certain routes from ppl if they aren't authorized to view them.

7.3.2. Forms

Answering When answering a form we decided that making the user feel like they're filling in a traditional linear form is important. This way the entire form structure can't confuse or distract them. After each filled in question, said question's logic gets evaluated, the next question (if there is one) is determined, and the new question is loaded in.

Modification Of course, it is possible to modify a previous question's answer. However, as a precaution we discard all answers to subsequent questions as soon as we detect modification. This issue could've been solved in a more user friendly manner. (E.g. re-evaluate the current input value every time a change is detected and compare that with the previously evaluated follow-up question.) But, because of the short-term nature of the project we left this as a feature for a future build.

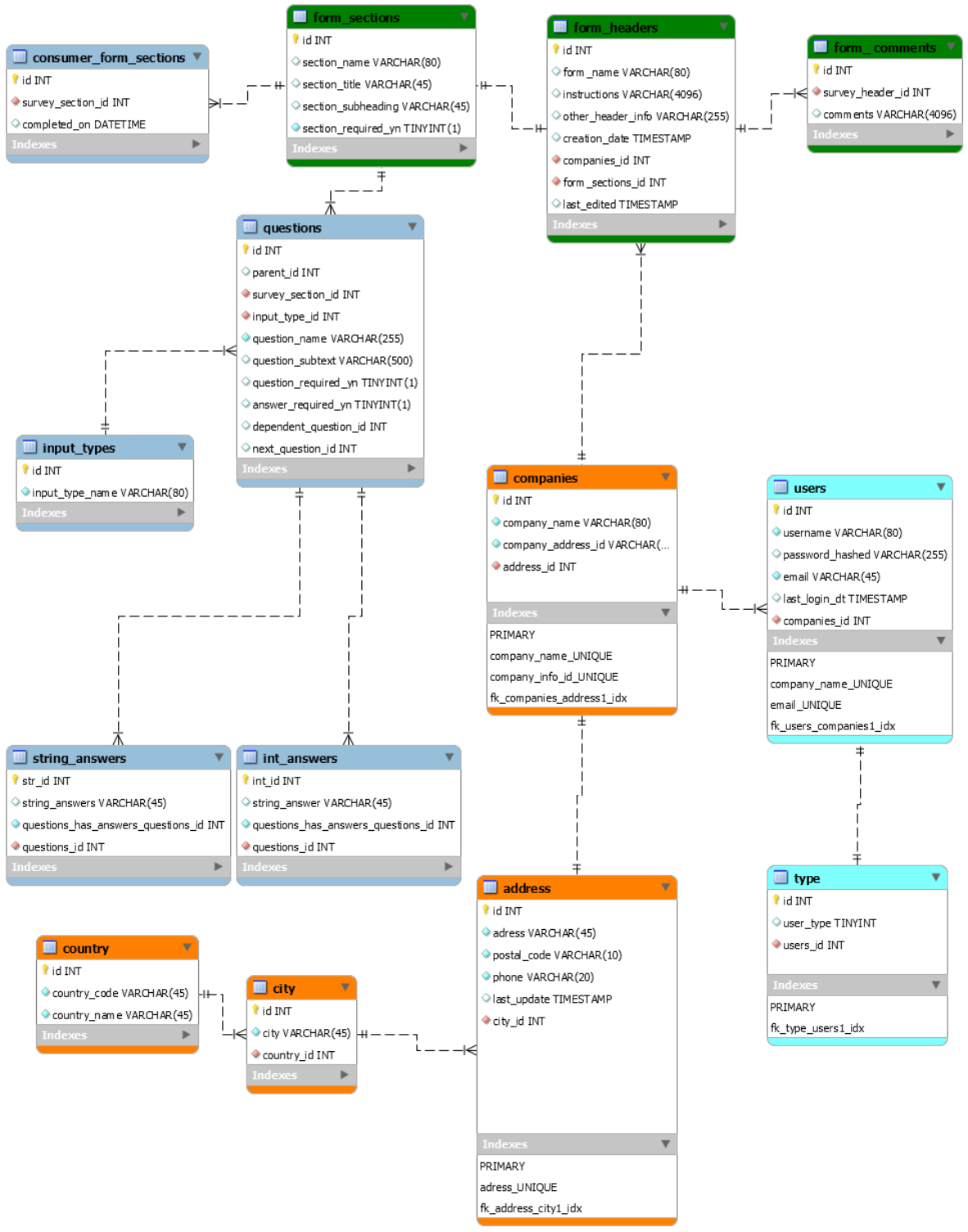

Figure 7.4: database schema

Question Types in general it is fairly straightforward to implement new question types as there is only 1 class used for questions. This class has a config attribute, which itself has a type attribute which holds that question's type. Furthermore, all type specific question attributes are optional in this config. (Type specific constructor functions ensure the proper attributes are set.) This leaves the config open and completely extensible to any future attributes we may need. All that's left to do then is to make a corresponding Angular

Component for this new question type, and make it possible for the builder to select this question type while building.

7.3.3. Form Logic

User Interface To make designing form logic as accessible to the layman as to any coder it had to be simple enough that it could easily be translated to the written language while still being powerful enough to support almost any use case a user might have. This was challenging as it meant we needed to design an interpreted language which could very easily be displayed in a user-friendly graphical interface while still keeping it lightweight qua size of the data and qua visualizing the code.

Elements Because of the restrictions placed upon the language we decided to look at the main action a user wants to achieve with our logic, this being the "jump". A builder wants the form to "jump" to a different question based on the input to a question or multiple questions. To allow the user to specify multiple "jumps" but still keep UI simple, every question contains an ordered list of **Jump** elements. Each **Jump** element contains a list of **Check** elements, which can be evaluated to true/false.

Jumps The list of **Jumps** is essentially just a merging of the traditional if statement and switch statement. In fact the only difference between our **Jump** and a switch case is that **Jumps** exclusively evaluate to the boolean true/false (just like an if statement). On evaluation to true the "switch" breaks and loads in the prespecified question or the end of the form if applicable. In the case that every **Jump** evaluates to false, the default case of going to the direct next question in the list is achieved.

Checks Each **Check** takes the input of a specific question and compares it to some specified value. The available comparators are dependent on the type of the input being compared (e.g. for numerical inputs options are $=, <, \leq, >, \geq$.) Lastly, when the list of **Checks** gets evaluated it gets run through a reduce function which combines the output of each **Check** with the next in the list using the boolean and/or operator. In this way the list allows us to provide rudimentary support for binary boolean operators.

Encoding Our language is encoded and stored as JSON. Because both our Jumps and Checks are in memory as objects anyway we determined this option was the simplest and least time consuming to implement. However, we are not under any impression that it is the most effective form of storage. JSON is a very general notation and if we designed our own format it would definitely be more space efficient. Regardless, we decided getting this feature production ready ASAP was worth much more to us than any space efficiency we could've gained from designing, and implementing our own in-house encoding.

7.3.4. Form Builder

User Interface Something as complex and feature filled as a form builder can be very daunting for a user to use. If a user gets overwhelmed with all the possibilities it could hinder their ability to build forms. It is for this reason that we went for a very focused design. This mainly means that a user can only look at a single question at a time. Our UI is also littered with collapsible UI elements so as much stuff as possible can be hidden at a time.

7.3.5. Analytics

View analytics To make it as clear as possible for the user to view the desired analytics, we have decided to create a to create that shows an overview of all the analytics for each question. More specifically if the user clicks on a created form, he will than be routed to the analytics page of the specific form, This page will than show every question and its specific analytics, which gives the user a clear overview. Moreover in this page the user can choose for each question how they want to visualize the data. But the choices how to view the data has a constraint depending on the question type. So ratings questions can only be visualized by a horizontal bar chart and a line chart, multiple choice and yes/no choice questions by a pie chart and line chart, number questions with a list with all the answer numbers and the calculated average of these numbers, phone numbers, text and email questions should show only a list with the answers.

Charts The user should be able to manipulate the charts in a way so that it can get the information which it desires. Therefore a start and end date is provided for each chart to see which answers were giving in a

certain time frame, this gives the user the opportunity to zoom in on a certain time frame. In addition the user is able to emit a legend for the pie chart, line chart, and horizontal bar chart, the reason being that in the case that the chart is to cluttered the user has to ability to view a chart that is more clear. Furthermore the user is able to decide how big the steps should be on the x-axis in a certain time frame for the line chart, these steps are weeks, months and years. Lastly the line chart has the function to show the data by simply counting the answers in a time frame, getting percentages or a cumulative average. These are implemented because from each view different information can be taken from the line chart.

8

Evaluation & Reflection

In this chapter we will go over the requirements or goals we set out for this project and evaluate what has been achieved. We will explore any ethical issues that this project might bring to front and lastly we will provide a reflection on uor experiences during the project.

8.1. Deliverables

8.1.1. Must Have

We managed to achieve all of our functional as well as technical must haves.

8.1.2. Should Have

At the current time of writing the report, we are still missing the functionality to visualize form response data. Our plan is to have it done by the demo. Sadly, companies will not be able to query form response data to help in analyzing form data. This, however, will is a strong recommendation for future work. Companies can manage their employees in the software and change their permissions. Copying a form is not yet possible but will be done by the demo. Sending forms to the consumers emails using the web-app is not yet possible but can be future work. Our front-end has Continuous Deployment that test our code and automatically puts the current build that is on the master on the VPS. The code quality score from the SIG(Software Improvement Group) hand-in was sufficient.

8.1.3. Could Have

False Form Alert was not accomplished, and neither were automated emails. Automated emails turned out to be more complicated than we initially anticipated.

8.2. Ethics

8.2.1. Consumer Manipulation

A company can build their form in such a way that any unhappy consumers reach a part of the form where they ask them if they would like compensation of a bad service, in effect bribing them. Furthermore, happy consumers can be lend down a path which asks them to leave a review at the end, while not showing unhappy consumers this path. By nudging happy consumers in the direction of reviews and unhappy consumers in the direction where they don't leave reviews, a company can effectively skew their ratings in a positive way even if they don't offer a good product or service.

8.2.2. Ignoring Problems

By covering up any potentially negative reviews a company has less incentive to improve themselves and their product. Which is to the detriment of the consumer because this can leave them without better options, and not even know it.

8.2.3. Automation

A company using an automated feedback system would understandably need less manpower to handle consumer feedback. For example all a company would need is someone to create the form, and someone to analyze it's responses to find weaknesses in the company. This could take away of job opportunities from actual people.

8.3. Self Reflection

8.3.1. Carlos Brunal

When starting this project I expected it to be a deeper, but not much more different, version of the context project. However I quickly learned that there was greater level of freedom that comes with the BEP. I was completely responsible for every decision I took. I enjoyed applying practices that I've covered trough-out the years in the Computer Science Bachelor in a real world situation.

The project took place in a real working environment further making me aware of the level of responsibility towards the end result, not only for my grade but also for the project provider and in turn also a sense of responsibility to represent the TU Delft in a good light. I set out to learn as much as I could about implementing a web-based system of this scale from the ground up. Programming in a controlled environment such as the context project or lab assignments is not comparable. I learned to make compromises with my shortcomings and focus on teamwork in order to get things done in a realistic time frame. My main tasks were mostly focused to database design and implementation, a subject I felt I was fairly experienced in. However, theory and practice differed greatly. I learned quite a big deal on the difficulty of database implementation on a live server. I do feel like I am more comfortable in my own skin that I was before the project when it comes this field. Overall it has been a stress-full yet very positive experience. In the future I think this experience will help me greatly in future projects. I learned how to communicate better with project givers considering that he is not so technically inclined and come to a common ground of understanding of what he wants. On the database I think I would have no issues deploying a similar structure in the future.

8.3.2. Daryll Lobman

I didn't really know what to expect going into this project. I only knew the work ethic of 1 out of the 5 group members and the project didn't seem to be all that complicated. However, I soon realised that this project would bring more challenges than I previously thought.

I coded the back-end, excluding the database, which means that a large part of the project was on my shoulders. Being responsible of the back-end I needed to manage both my own progress and that of my teammate who was working on the database, but I often found myself working on my own aspect of the code and not managing our progress properly, which eventually caused the back-end to lag behind because there was no available database when we needed it.

I learned a lot about my own skills and improved my coding skills a lot in this project. Renzo was very helpful in managing the team (taking up the role of the leader) by making sure to put pressure on everyone to finish their work. However, I still find myself lacking when it comes to management of others and coding speed, code quality and quickly figuring out to solve coding issues.

In the future, I would like to work on my blind typing and my typing speed in order to code faster. I also want to work more with websites(by using React) and learn GO(to write back-end servers).

8.3.3. Saeed Najafi

For me the most exciting aspect of this project was that, there was no foundation to be built upon, but rather we had to start with every part of the project on our own. This gave all of us the chance to improve our skills in many different levels, from front-end to back-end, from design to development and research. This fact was more specifically useful for me as this would help me enormously in my future career. By doing this project I learned various important lessons that is directly applicable in my life and future work opportunities, things such as: Teamwork, how to do proper research, decision making, high-level programming and many more, also thanks to a hardworking team and the helpful coach that we had. That is exactly why I took everything about this project seriously from the very beginning and devoted myself to learn and grow personally as well as making sure we have a great end result for the project.

To make sure by the end of the project we all meet the client's requirements and hopefully even exceed that, we decided to first divide the tasks and have one person assigned to one specific part of the project and make sure that parts excels, but besides that be there for your teammates and help them in their parts when

needed. I mainly focused on front-end web development during the project. I created parts of the dashboard and the splash pages and made sure the routing between them works correctly. However, that was surely not the only part that I worked on. The other parts of the project that I contributed to, among other things, I helped one of my colleagues with some parts of the data-visualizations and helped another one with the login and registrations system.

Now that we are at the end of the project and have shown the end-product to our client, and see that he is happy with the end results, I can say that I am really proud of this accomplishment that we made in the matter of 2 months. And I am honored to hear that the client is offering us to even continue working on this product after this project ended. This shows that personally for me, that all the hard work and efforts that I put into this project has paid off.

8.3.4. Sander Oostmeyer

At the start of this project I was excited to work for a company and apply the skills that I have obtained from my bachelor and to further develop myself. I mostly worked on the front end, and this is something which I had not much skill in as I mostly worked on the back end on most of the other projects I ever did. In addition I never worked on web applications excluding the lab assignments for the web and databases course, and therefore I have learned a lot from this project concerning the development of web applications. Furthermore I also liked the freedom that we had in a project for the first time, but with the fact that we were not making this project only for a good grade to finish the bachelor but also because the company was depending on us gave a lot of pressure at times. I worked mostly on the front end in the entire project and I learned a lot from this especially on front end development and design. But in retrospective there are some things in the project I could had done better. Firstly, in the beginning of the project I did not commit my code frequently, but later on I picked up this and improved myself. Secondly I underestimated some parts of my work, while working on the analytics I thought I would be done in a week but in the end it took longer than that, the reasons being that I ran into problems with the charts not working and bugs that emerged which I had not foreseen to take up this much time. What I learned from this is that I should improve on the estimation of work that I am given, as I think this is a important part to schedule the work effectively in the project. In the future I will do more research on the problem that I am given before I give any estimations. But most importantly I learned overall to work in a team in a professional environment, how to communicate with the supervisor from the company, to really give the product that he desires. These are all skills which I feel are very important in the future as a software developer, and these lessons I will take with me for the rest of my life. Overall I'm satisfied with the project and the teamwork from the group.

8.3.5. Renzo Russel

As someone who usually works on system level code, I explicitly wanted to challenge myself by working on the front-end. Web development has always been a weakness of mine. And I'm proud to say that I've learned a ton doing almost exclusively front-end web development. But most importantly I learned a lot about managing a development project. From learning the importance of task division to learning how to better understand the needs of the client. Most importantly my shortcomings during the project have inspired me to act more quickly in the future when I notice a project is going off course. Lastly, I can now with certainty say I'm not really fond of development and I'm going to stick to lower level stuff from now on.

9 Conclusion

In conclusion we have made a web application for the company Feedback-Analytics, which had to be build from the ground up both for the front-end and back-end. The teamwork with the company and within the project group went well. At the moment of writing, however, not all requirements are entirely finished. Requirements such as the analytics and form fetching are not yet ready for production, however, this is expected to change before the presentation deadline. As a result, we will have delivered a product with the basic features Feedback-Analytics expected from the project such as a login functionality, form building, form answering and data analytics. The product will then be able to service potential customers and can be further built upon with more features to the liking of Feedback-Analytics. In conclusion, Feedback-Analytics has a fully functioning product and it can be extended by the company in the future.

A

Weekly Sprints

Sprint 1 - April 22nd

Tasks	Assigned to
As a developer we want to have a good work environment / Install Visual Studio/ Git	All
As a developer we want to have starting knowledge of the languages and tools | Research Angularjs	All
As a developer we want to have our code synchronized with code history	All
As developers we want to have a clear definition of what we consider our complete product to be	All
Analysis of existing alternatives	All

Sprint 2

Tasks	Assigned to
As developers we want to have a clear and concise product vision	Saeed
As developers we want to have clear product plan	Renzo
As a developer we want to have a clear plan for our architecture design	Renzo
As a developer we want to have access to a CI-server, resulting in reliable and clear tests	Daryll
Write Introduction	Sander
Write System Design	Renzo
Write Problem Definition	Carlos
Write Software Implementation section in repport	Saeed
Write the form definition section on the repport	Renzo
Defining features	Saeed
Research Database implementation options	Carlos
Proof read draft	Daryll
Training knowledge of the languages and tools | Research Angularjs	All

Sprint 3

Tasks	Assigned to
Expand CI	Daryll
Check and correct the report draft	Daryll
Make the main page	Saeed
Make basic website routing	Saeed
Create form viewing page	Renzo
Add form answering functionality to the page	Renzo
Create form datastructure	Renzo
Create form answer datastructure	Renzo
Design the SQL relational database	Carlos
Set up the SQL relational database in Mysqlworkbench	Carlos
Set up the Server	Daryll
Create all the routes	Daryll
basic data storage & retrieval (Server DB combo)	Daryll
Create dashboard view	Sander
Create settings page for company accounts	Sander

Figure A.1

Sprint 4

Tasks	Assigned to
Debug form viewing page	Renzo
Debug form answer datastructure	Renzo
Set up the SQL relational database on the VPS	Carlos
Finalize settings page for company accounts	Sander
REPORT: update visualization scheme	Carlos
REPORT: add sequence diagram to Software Design	Renzo
REPORT: add use cases+ diagrams	Saeed
Deploy the server (live)	Daryll
Handle server requests	Daryll
Create server service	Daryll
Create database queries	Carlos
Create dashboard page	Saeed

Sprint 5

Tasks	Assigned to
Debug the design of the SQL relational database	Carlos
Create update user info page	Sander
Create analytics page	Sander
Write description for use case diagram	Saeed
Work further on splash page/ dashboard	Saeed
test the splash page+dashboard page components	Saeed
Create formbuilder UI	Renzo
formbuilder settings UI for changing the general settings	Renzo
Add question to form functionality	Renzo

Sprint 6

Tasks	Assigned to
Update SQL Database Scheme to 3NF form	Carlos
Create login function	Saeed
Create user registration page for Freek!	Saeed
Add notes, scheme's and proof read Project Report	Carlos
Server hardening and Security checking	Daryll
Debugging and preparation for First SIG hand in.	All
Code quality control, and Coverage testing for SIG	All

Figure A.2

Sprint 7

Tasks	Assigned to
Address JS Vunerabilities in the Code	Daryll
Update Database keys and dependencies according to new changes	Carlos
Implement mdBootstrap for chart building and design	Sander
Add different types of charts to be used for analytics	Sander
Navigation Bar toggle, finalize main page dashboard graphics design	Saeed
Implement check components for logic builder	Renzo
Migrate databse to VPS server	Daryll
Create server controller to connect fron and back end	Carlos
create databse scheme for better querying overview	Carlos
Debug Logic jump portiong in front end	Renzo

Sprint 8

Tasks	Assigned to
Setup Sequelize ORM for Serverside Database Creation	Daryll
Adapt existing database design to sequelize	Carlos
Passport atuthorization implementation	Daryll
Add automated deployment or repository code on server side	Daryll
Create API service for server requests	Renzo
Further Debugging of form builder	Renzo
Finalize front page with content and all key components	Saeed
Further work on the Analytics side of the front end	Sander

Sprint 9

Tasks	Assigned to
Debugging and checktyles fixes to the code in preparation to sig	All
Finalize Sequelize ORM	Carlos
Implementation of the form API to connect back and front end database	Carlos
Debug and fix vaious issues wit authorization	Daryll
Apply SIG feedback	All
Debugging Analytics side of front end for SIG	Sander
Datavisualization in dashbord	Saeed

Sprint 10

Tasks	Assigned to
Finish infosheet	Carlos
Create demo video	Renzo
Preapare live demo	Renzo
Report Process	Carlos
Report Ethics	Saeed
Report Research	Daryll
Finalization of Report	All
Project presentation	All
Report Implementation	All

Figure A.3

B

Git Statistics

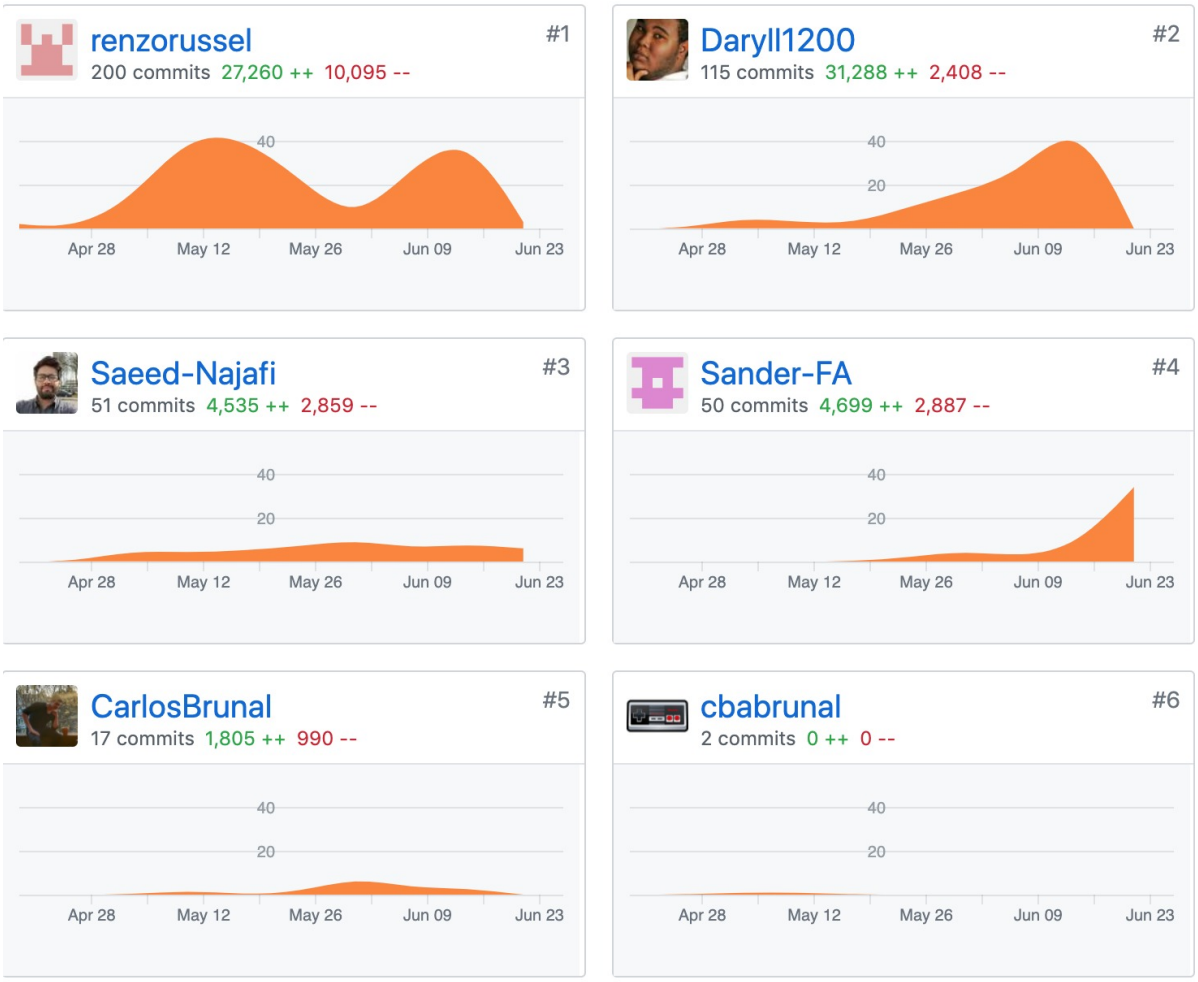

Figure B.1: renzorussel: Renzo Russel, Daryll1200: Daryll Lobman, Saeed-Najafi: Saeed Najafi, Sander-FA: Sander Oostmeyer, Carlos-Brunal & cbabrunal: Carlos Brunal

Bibliography

[1] Angular. URL https://angular.io/.

[2] React – a javascript library for building user interfaces. URL https://reactjs.org/.

[3] How to build a news application with angular 6 and material design, Oct 2018. URL https://www.smashingmagazine.com/2018/10/news-application-with-angular-and-material-design/.

[4] New angular features you didn't know existed, Jun 2019. URL https://blog.logrocket.com/new-angular-features-you-didnt-know-existed-7f292a6e7afc/.

[5] Bcrypt, Apr 2019. URL https://en.wikipedia.org/wiki/Bcrypt.

[6] Blowfish (cipher), Jan 2019. URL https://en.wikipedia.org/wiki/Blowfish_(cipher).

[7] React vs. angular: The complete comparison, Mar 2019. URL https://programmingwithmosh.com/react/react-vs-angular/.

[8] Tomislav Capan. Why the hell would i use node.js? a case-by-case tutorial, Aug 2013. URL https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js.

[9] Avhishek Chatterjee, Lei Ying, and Sriram Vishwanath. Revenue and reputation: A stochastic control approach to profit maximization. In *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, Oct 2012. doi: 10.1109/allerton.2012.6483275. URL http://dx.doi.org/10.1109/allerton.2012.6483275.

[10] NGINX Inc. What is nginx?, 2019. URL https://www.nginx.com/resources/glossary/nginx/.

[11] D. Hardt M. Jones. The oauth 2.0 authorization framework: Bearer token usage. RFC 6750, RFC Editor, October 2012. URL https://www.rfc-editor.org/info/rfc6750.

[12] N. Sakimura M. Jones, J. Bradley. Json web token (jwt). RFC 7519, RFC Editor, May 2015. URL https://www.rfc-editor.org/rfc/rfc7519.txt.

[13] Nitin Pandit. What is reactjs and why should we use it? URL https://www.c-sharpcorner.com/article/what-and-why-reactjs/.

[14] Ayush Sharma. Difference between sql and nosql. (2018, october 26), 2018. URL https://www.geeksforgeeks.org/difference-between-sql-and-nosql.