

# Student Research Project

Documentation

## Knackpunkt BRK

Semester: Autumn 2022/23



Version: 1.0

Date: 2023-03-01 20:53:44Z

Git Version: Not available

**Project Team:** Carlo Del Rossi  
Claudio Knaus

**Project Advisor:** Mirko Stocker

School of Computer Science  
OST Eastern Switzerland University of Applied Sciences

# Abstract

The UN CRPD (United Nations Convention on the Rights for People with Disabilities) is a human rights instrument, aiming to change attitudes and approaches toward persons with disabilities.<sup>1</sup>

The ZEN (Center for Ethics and Sustainability) of the OST (Eastern Switzerland University of Applied Sciences) has created a printable survey form, that focuses on several aspects of the UN CRPD. This survey form allows institutions and homes for people with disabilities to evaluate themselves.

The goal of this project is to digitize the survey form created by the ZEN to simplify the entire process of distributing survey forms and tallying scores.

We ended up with a survey application that allows institutions to start surveys that are accessible via an online link. The survey can then be filled out by respondents who received that link. Once the survey is completed, institutions can view the results and receive recommendations on how to improve their implementation of the UN CRPD. During the creation of the application, we placed great emphasis on accessibility, to make sure it can be filled out by almost everyone. During the duration of the project we managed to implement a large part of the desired functionality, however, there are still some features missing. Despite this, we have managed to create a usable product that provides value for institutions wishing to evaluate themselves.

---

<sup>1</sup>*Convention on the Rights of Persons with Disabilities (CRPD)*. URL: <https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities.html>. (accessed: 09.12.2022).

# Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>1</b>	<b>Management Summary</b>	<b>2</b>
1.1	Baseline . . . . .	2
1.2	Achieved Results . . . . .	5
<b>II</b>	<b>Analysis</b>	<b>6</b>
<b>2</b>	<b>Domain Analysis</b>	<b>7</b>
2.1	Problem . . . . .	7
2.2	Domain Specific Language . . . . .	8
2.3	Domain Model . . . . .	8
2.4	Domain Model Assumptions . . . . .	9
<b>3</b>	<b>Requirements</b>	<b>10</b>
3.1	Use Case Diagram . . . . .	10
3.2	Functional Requirements . . . . .	11
3.3	Non-Functional Requirements . . . . .	12
<b>III</b>	<b>Design</b>	<b>13</b>
<b>4</b>	<b>Architecture</b>	<b>14</b>
4.1	Overview . . . . .	14
4.2	Architectural Decisions . . . . .	16
4.2.1	Frontend . . . . .	16
4.2.2	Backend . . . . .	16
4.2.3	Database . . . . .	17
4.2.4	Authentication . . . . .	17
4.3	C4 Diagrams . . . . .	17
4.3.1	Context Diagram . . . . .	18
4.3.2	Container Diagram . . . . .	18
4.3.3	Component Diagram . . . . .	19

<b>5</b>	<b>Database</b>	<b>21</b>
5.1	Database Tables . . . . .	22
5.1.1	Survey Template . . . . .	22
5.1.2	Institution and Admin Account . . . . .	22
5.1.3	Credential . . . . .	22
5.1.4	Survey . . . . .	22
5.1.5	Question . . . . .	23
5.1.6	Definition and Scored Answer Option . . . . .	23
5.1.7	Comment and Answer . . . . .	23
<b>IV</b>	<b>Implementation</b>	<b>24</b>
<b>6</b>	<b>Frontend</b>	<b>25</b>
6.1	Application Structure . . . . .	25
6.2	Access Token Flow . . . . .	26
<b>7</b>	<b>Backend</b>	<b>27</b>
7.1	File Structure . . . . .	27
7.2	Interesting Code Snippets . . . . .	28
7.2.1	Injecting Dependencies . . . . .	28
7.2.2	Testing Functions that use the Notifier . . . . .	30
7.2.3	Testing Exceptions . . . . .	31
<b>8</b>	<b>Proxy</b>	<b>34</b>
8.1	Configuration . . . . .	34
<b>9</b>	<b>Quality Measures</b>	<b>36</b>
9.1	Coding Guidelines . . . . .	36
9.2	Test Concept . . . . .	37
9.2.1	Unit Testing . . . . .	37
9.2.2	Integration Testing . . . . .	37
9.2.3	Frontend . . . . .	38
9.2.4	Backend . . . . .	38
9.3	Workflow . . . . .	39
<b>V</b>	<b>Discussion</b>	<b>40</b>
<b>10</b>	<b>Results</b>	<b>41</b>
10.1	Functional Requirements . . . . .	41
10.2	Non-Functional Requirements . . . . .	42
10.3	Screenshots . . . . .	43

<b>11 Conclusion &amp; Next Steps</b>	<b>48</b>
11.1 Functional Requirements	48
11.1.1 Non-Functional Requirements	49
11.2 Reflection Frontend	49
11.2.1 axe DevTools Testing Limitations	50
11.2.2 Maintainability and further Testing	50
11.2.3 Evaluation on Technology	51
11.3 Reflection Backend	51
11.4 Next Steps	52
<b>VI References</b>	<b>54</b>
<b>VII Appendix</b>	<b>57</b>
11.4.1 Responsibilities	58
<b>12 Operational Notes</b>	<b>59</b>
12.1 CI/CD	59
12.1.1 Automated Backend Test Action	59
12.1.2 Automated Docker Image Build Actions	60
12.1.3 Automated Documentation Building	61
12.2 Installation instructions	62
12.2.1 Docker Compose File	63
12.2.2 Environment File	65
12.3 Updating the Survey Questions	66
12.4 Test-logs	70
12.4.1 Backend	70
12.4.2 Frontend	71

**Part I**

**Introduction**

# Chapter 1

## Management Summary

















The ZEN (Center for Ethics and Sustainability) research team commissioned the OST University of Applied Sciences to conduct an online survey as part of a software engineering project. The online survey developed by the project team is primarily aimed at institutions with disabled people who want self-evaluation.

### 1.1 Baseline

The ZEN has developed a questionnaire to evaluate institutions with disabled individuals against the UN CRPD (United Nations Convention on the Rights of Persons with Disabilities). The questionnaire will be completed by residents, professionals of the institution, or the resident's relatives. The survey focuses on the wishes of disabled people regarding sexuality, partnership and the desire to have children.

The survey results are used to create recommendations for different groups of respondents that institutions have insight into. Anonymity among respondents and institutions is also a requirement. The questionnaire is available on paper, and respondents can enter their answers into an evaluation system. The questionnaire also includes explanations of certain words in easy language and space for comments.

The project was divided into five phases: elaboration, requirements, building, transition, and documentation. In the elaboration phase, the long-term project plan and initial tooling were established. The requirement phase focused on developing drafts and defining the MVP (Minimum Viable Product). The building phase involved the actual creation of the product using vanilla TypeScript for the frontend and Node.js with TypeScript for the backend. The transition phase included a dockerized deployment on a test server, used by the ZEN to re-evaluate and incorporate new requirements. Finally, the documentation phase will involve the creation of necessary documentation.

F	Sexuelle Gewalt	 Ja	 Manchmal Teilweise	 Nein	 Kann ich nicht beantworten
40	Hat das Wohnheim Ihnen erklärt, was <b>sexuelle Gewalt</b> ist?  <div style="border: 1px solid blue; padding: 5px;"> <b>Sexuelle Gewalt:</b>             Jemand belästigt Sie sexuell.            Zum Beispiel: jemand berührt Sie an einer intimen Stelle.            Zum Beispiel: am Busen oder Po.            Du willst das aber nicht.            Oder jemand zwingt Sie zum Sex. Das ist verboten.         </div>				
41	Hat das Wohnheim Ihnen erklärt, was Sie machen müssen, wenn Sie sexuelle Gewalt erleben?				
42	Gibt es im Wohnheim eine Beschwerde-Stelle bei sexueller Gewalt?				

(a) Initial Print version from the ZEN

## Bildung

Gibt es im Wohnheim Ansprechpersonen zum Thema Sexualität?

Bitte wählen Sie eine Antwort aus.

Ja 

Nein 

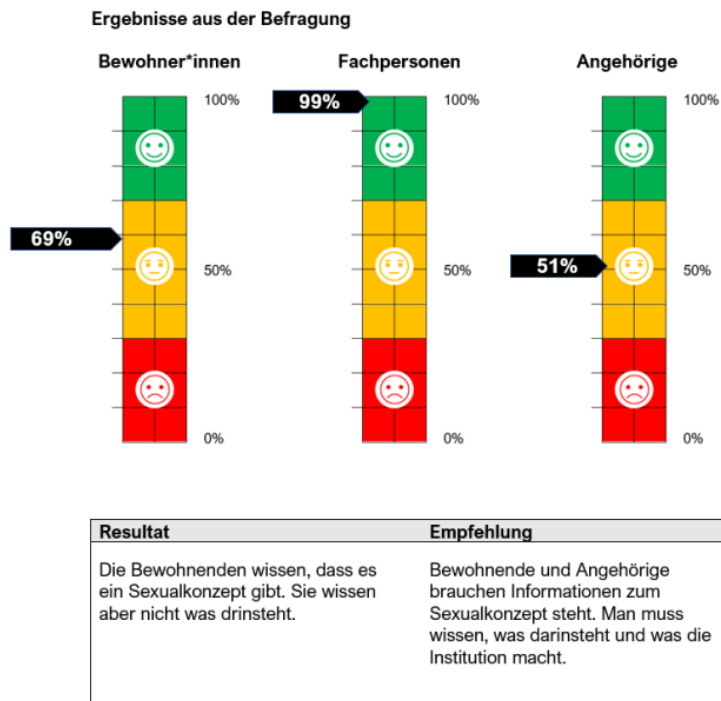
Teilweise / Manchmal 

Kann ich nicht beantworten ✕

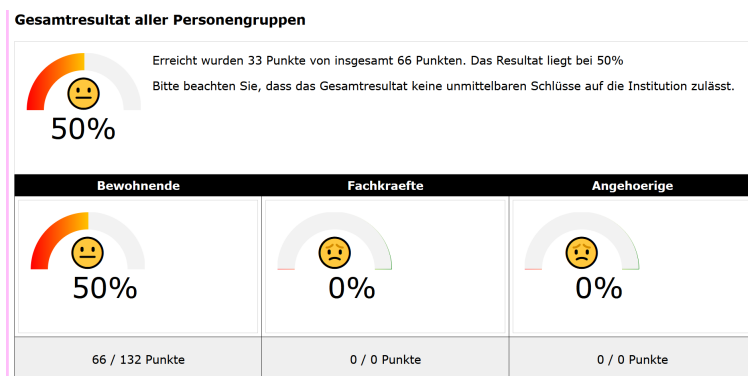
(b) Resulting Product

Figure 1.1: Original Print Version and Result Side by Side

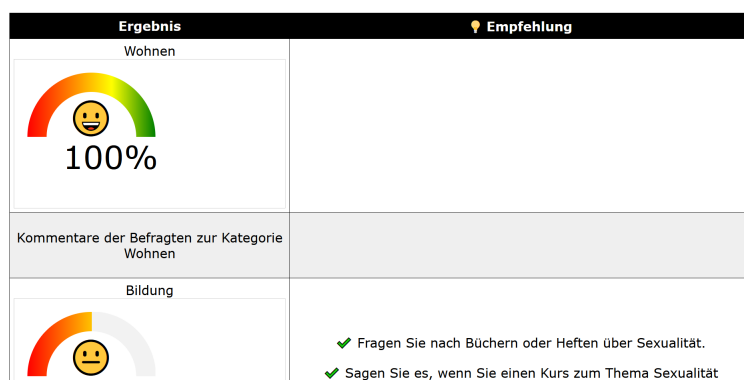




(a) Original Print Version of Results and Recommendations from the ZEN



(b) Resulting Product of Results



(c) Resulting Product of Results & Recommendations from OST

Figure 1.2: Original Print Version and Result Side by Side

## **1.2 Achieved Results**

The authors found that the explicitly required goals were met, allowing institutions to create and delete surveys. Some implications of the specified goals were further measures, such as creating a user account or changing the email address and password. However, some requirements resulting from implications of the main requirements were not met, including the ability for the ZEN to edit the questions after the project had been submitted. The explicit requirements have been outlined in the Requirements chapter, and further details on non-compliance are in the Conclusions chapter.

**Part II**  
**Analysis**

## Chapter 2

# Domain Analysis

### 2.1 Problem

To analyze the domain we have discussed the problem with the ZEN. The goal is to create an application for the evaluation of the implementation of the United Nations Convention of the Rights of Persons with Disabilities (UN-CRDP). From the discussion we were able to take away some key points:

- The institutions should be able to start self-evaluations on the implementation of the UN-CRDP.
- Those self-evaluations should be made in the form of a survey.
- The survey is filled out by employees and residents of the institution, as well as relatives of the residents.
- The survey questions are managed by the ZEN.
- There are up to four answer options per question: 'Yes', 'No', 'Sometimes/Partially' and 'Cannot Answer'.
- Each option has a certain score.
- Each question belongs to a certain category.
- Once a survey is over, the scores can be tallied up and grouped by target group and category.
- The results should be displayed on a scale from 0 to 100.
- The scales should be color-coded. (red: below 33.3%, yellow: between 33.3% and 66.6% and green: more than 66.6%)
- If the total is below 66.6%, some recommendations are displayed that can help the institution with the implementation of the UN-CRDP.

## 2.2 Domain Specific Language

**Respondent:** A person that fills out the survey received from the institution.

**Employee:** A person that works at the institution. A subgroup of respondents.

**Resident:** A person that lives in the institution. A subgroup of respondents.

**Relative:** A relative of a resident. Subgroup of respondents.

**Survey Administrator:** A person that has control over the questions that are part of the survey and is capable of adding/removing questions and managing institutions.

**Institution Representative:** A person that creates and maintains the account for the corresponding institution.

## 2.3 Domain Model

From the previous textual description we have created the following domain model:

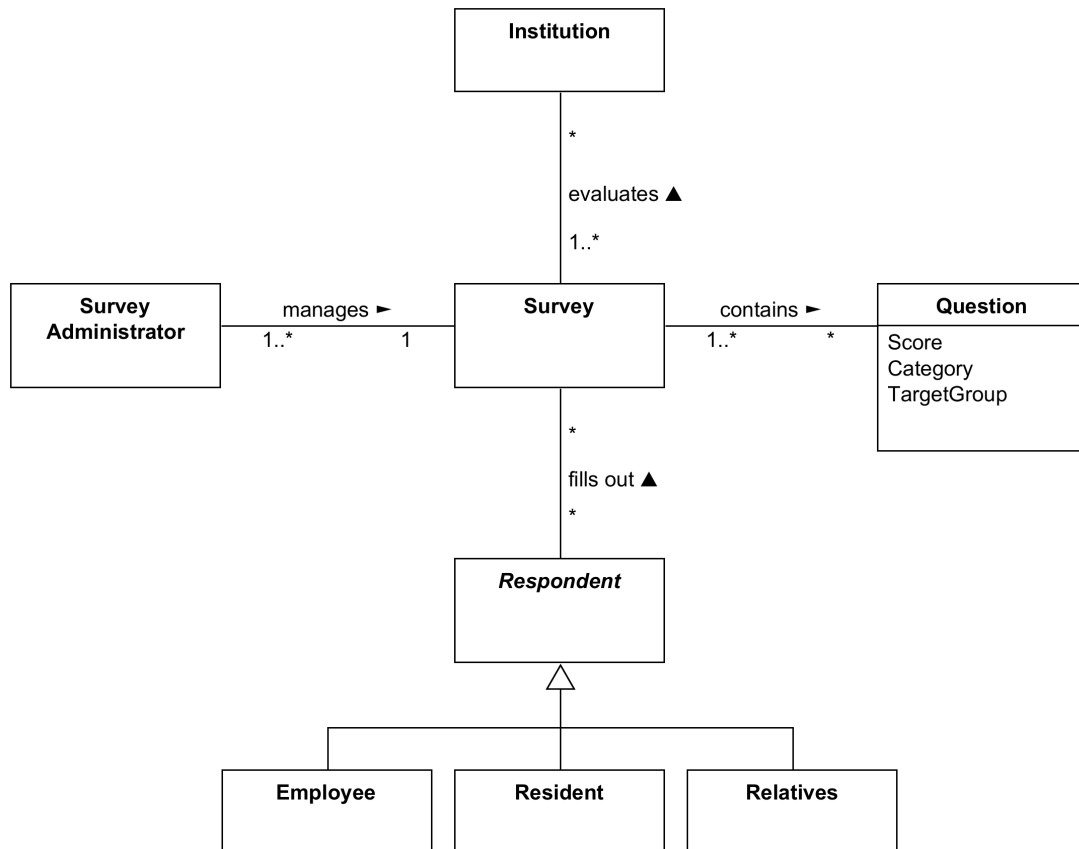


Figure 2.1: Domain Model

## 2.4 Domain Model Assumptions

The following assumptions were made concerning the Domain Model:

- The surveys can be edited. This means Questions can be added and removed. For past surveys to make sense, removed questions must be kept. If a question is not removed, it will be part of multiple iterations of the survey. That is the reason for the 1..\* cardinality in the Survey-Question relationship.
- An institution can be evaluated multiple times, to see the progress. That is the reason for the 1..\* cardinality of the institution-survey relationship.
- There has to exist at least one maintainer of the survey that keeps the survey up to date by adding or removing questions. This corresponds to the Survey Admin.

# Chapter 3

## Requirements

### 3.1 Use Case Diagram

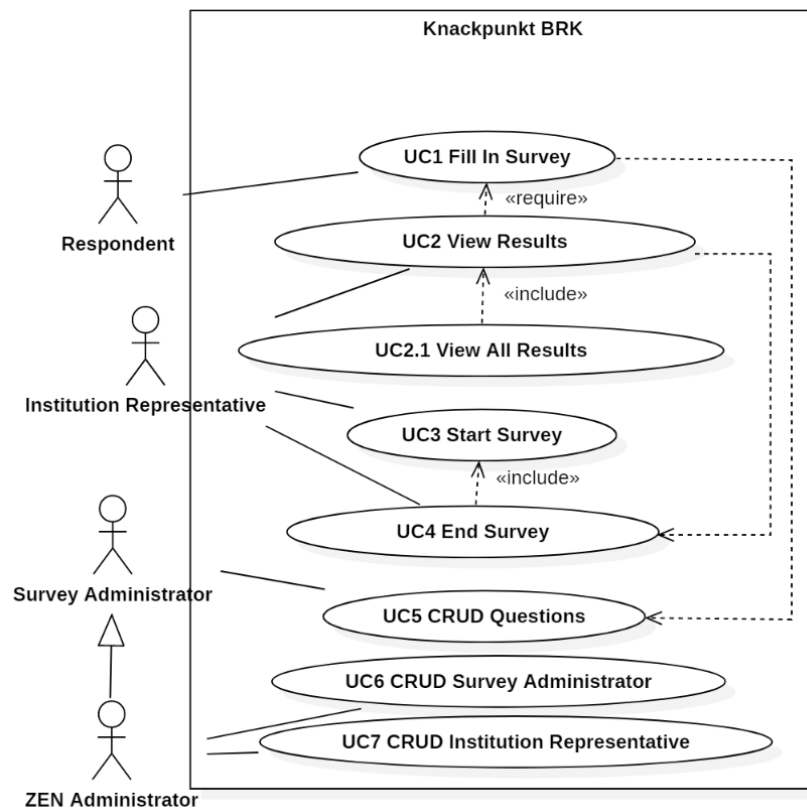


Figure 3.1: Use Case Diagram

The use case diagram summarizes the functional requirements (FRs) as a graphic depiction of interactions among different roles within the system, which shows the ex-

pected system in its final form. Additional requirements that have not been prioritized are not explicitly listed in the use case diagram. This included the displaying of recommendations for the disabled people at the end of a survey, as well as the possibility for institution representatives to anonymously share their results with the ZEN. In addition, the customer's requirement to print out a survey became obsolete.

## 3.2 Functional Requirements

<b>Use Case</b>	<b>Description</b>
<b>UC1 Fill In Survey</b>	Respondents should be able to fill in a survey based on their role: 'Resident', 'Employee', 'Relative'.
<b>UC2 View Results</b>	Institution Representative should be able to view results and recommendations of a single survey that has ended.
<b>UC2.1 View All Results</b>	Institution Representative should be able to view results and recommendations for each ended survey.
<b>UC3 Start Survey</b>	Institution Representative should be able to start a new survey.
<b>UC4 End Survey</b>	Institution Representative should be able to set an end date for a survey to make corresponding results visible for the account.
<b>UC5 CRUD Questions</b>	Survey Administrator should be able to add and remove survey questions.
<b>UC6 CRUD Survey Administrator</b>	ZEN Administrators should be able to manage account rights for their staff members with Survey Administrator accounts.
<b>UC7 CRUD Institution Representative</b>	ZEN Administrators should be able to manage account rights for Institution Representative accounts.



### 3.3 Non-Functional Requirements

The following non-functional requirements (NFRs) were derived from the requirements on accessibility, useability, privacy, domain specifications and engineering specifications.

Nr.	Description	Measurement	Prio
NFR1	People with cognitive impairments can fill out surveys easily.	Acceptance Tests	5
NFR1.1	The survey follows the WCAG 2.1 guidelines with a score of AA, thus minimum standard plus additional features.	Acceptance Tests, Linter Verification	5
NFR1.2	WCAG 2.1 Guideline, Operable - Navigation: Survey can be filled out by keyboard, mouse, touch screen or screen reader only.	Acceptance Tests, partial Linter Verification	5
NFR1.3	WCAG 2.1 Guideline, Understandable - Readability: Content is readable and easy to understand. Definitions should be enclosed by the abbr-tag.	Acceptance Tests, partial Linter Verification	5
NFR1.4	WCAG 2.1 Guideline, Robustness - Compatibility: Semantically correct HTML must be used wherever possible and they should be enriched with ARIA tags.	Acceptance Tests, partial Linter Verification	5
NFR2	The survey must be anonymous. A questioned person must never be asked to provide personally identifying information.	Acceptance Tests	5
NFR3	Code must stay maintainable.	At least 80% test coverage, so that refactoring can be applied as confidently as possible.	4
NFR4	For easier deployment, all the components must be dockerized.		5
NFR5	To prevent secrets from accidentally being uploaded, they must be saved in environment variables.		3
NFR6	A linter must be used to prevent bad practices.		4
NFR7	Quick response time when starting a survey.	After clicking the 'Start Survey' button, the user must start seeing questions after no longer than 3s.	4
NFR8	Quick response time when inspecting survey results.	After clicking the 'See Survey Results' button, the user must see the results within 3s.	3
NFR9	Quick response time for submitting results.	After clicking the 'Submit' button, the user must see the 'Survey Finished' screen after no longer than 3s.	4

**Part III**  
**Design**

# Chapter 4

## Architecture

### 4.1 Overview

For our project, we decided to go with a classic three-tier design. We split up the application into the following tiers: Frontend (Presentation & Dialog Control Layer), Backend (Application Kernel & Database Access Layer) and Database (Persistence Layer). Thus, we applied the following Client/Server Cuts: Remote User Interface and Remote Database.<sup>1</sup>

We have decided to go for this approach as it allows us to separate the application into self-contained parts. This allows us to define clear error boundaries and easily collaborate on the project independently. Since each of those tiers runs on a separate docker container and the backend is stateless, this would also allow us to duplicate the frontend and backend services, to be able to scale horizontally. The database could also be scaled horizontally, but it would require some extra work to ensure consistency between the different instances.

As seen in figure 4.1, we decided to follow the Clean Architecture approach.<sup>2</sup> By doing this, we mainly tried to achieve high testability and independence of the framework, as well as the independence of the database.

At the center of our architecture are the entities. They are at the core of the business logic. It is those entities that are persisted in the database. They contain all the data needed to make the application work.

A layer further out we can find the business logic, security components and repositories.

The business logic is responsible for all the use cases. It is the business logic that

---

<sup>1</sup>Klaus Renzel and Wolfgang Keller. “Client/Server Architectures for Business Information Systems A Pattern Language”. In: 1997. URL: <https://hillside.net/plop/plop97/Proceedings/renzel.pdf>. (accessed: 09.12.2022).

<sup>2</sup>Robert C. Martin. *The Clean Architecture*. URL: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. (accessed: 09.12.2022).

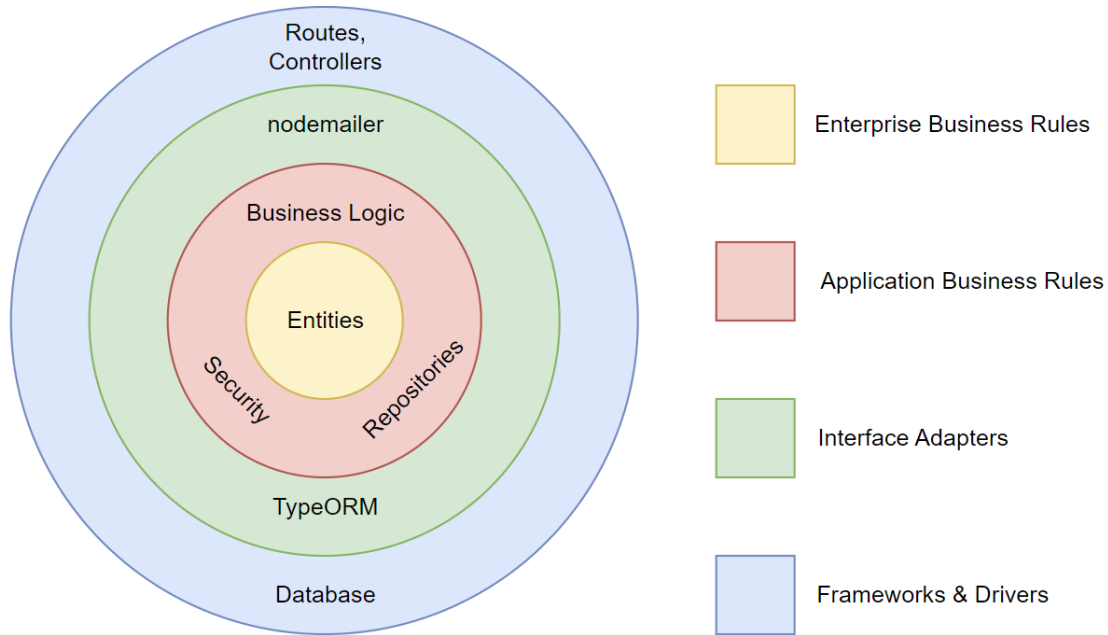


Figure 4.1: Architecture Diagram

performs all the necessary operations for the application to work.

The security components are supporting the business logic. For example, the hashing module provides all the necessary logic to hash the users' credentials so that the login logic can function properly. And the JWT module provides the necessary logic to create JWTs, which is also crucial for the business logic to be able to function.

The repositories provide the necessary logic to read from and write to the database. Using dependency inversion we were able to decouple the business logic from the TypeORM logic. For this, we created different repository interfaces, that contain the necessary methods for the business logic to operate properly. The repository implementation is injected at the startup of the application. To swap out the TypeORM library one simply has to rewrite the implementations of the repositories, the business logic is not affected by this change.

Yet another layer further out we can find the libraries used. These libraries include nodemailer for the sending of emails to users, and the TypeORM library responsible for the persisting and loading of database entries. Both of these libraries are wrapped in a class and injected into the business logic via dependency injection. This allows these components to be exchanged easily. For example, TypeORM could be exchanged with node-Postgres just by rewriting the repository implementations or providing new ones. The same goes for the nodemailer library which is wrapped inside the GmailNotifier class.

On the outermost layer, we can find the express framework which consists of routes and controllers. The routes are responsible for the routing of HTTP requests to the right controller, which in turn is responsible for executing the correct business logic functions and providing HTTP responses for the HTTP requests.

## **4.2 Architectural Decisions**

### **4.2.1 Frontend**

#### **Framework**

In the context of the frontend framework, facing the short amount of time available for our project, we decided to use plain HTML/CSS/TypeScript and neglected frameworks like VUE to avoid having to familiarize ourselves with a framework, accepting that we have to do more work by ourselves.

#### **Communication with Backend**

In the context of communication between frontend and backend facing the need to quickly and easily answer requests from the frontend we decided to use an HTTP-web-API and neglected other solutions like WebSockets to achieve a unidirectional API for the frontend to make requests. As the API only needed to be unidirectional, sockets would not have made a lot of sense, as they bring more complexity.

### **4.2.2 Backend**

#### **Server-Side Technology**

In the context of server-side technology, facing the wish to use a technology we were already familiar with, we decided to use Node.js and neglected other technologies like Spring Boot to minimize the risk of using a technology that we're not familiar with.

#### **Programming Language**

In the context of server programming language, facing the wish to use a typed language we decided to use TypeScript and neglected JavaScript accepting that we would have to get used to a slightly different syntax.

#### **Framework**

In the context of the backend framework, facing the need to easily answer requests from the frontend, we decided to use the express framework and neglected other frameworks like nest.js etc. to achieve a simple solution using frameworks that we were already familiar with accepting that we miss out on extra functionality, other frameworks could have provided.

## Database Connection

In the context of communication between backend and database facing the need to get/save entities to/from the database in an easy way, we decided to use TypeORM and neglected libraries like node-Postgres to achieve easy database management so we don't have to manage connections etc by ourselves and so we don't have to create the database model by ourselves accepting certain limitations of the object-relational mapping (ORM).

### 4.2.3 Database

#### Database Paradigm

In the context of the database, facing the need to persist data with as little redundancy as possible and not having to learn to use a new paradigm from the start, we decided to use the relational paradigm and neglected the graph- and document-oriented paradigm to achieve a solution without the extra effort of learning a new paradigm and a low degree of redundancy, accepting that there is more redundancy than if we used a graph-based paradigm and that there is a bit more complex than if we used a document-based paradigm.

#### Database Type

In the context of the database, facing the need to use a database that we were already familiar with, we decided to use a PostgreSQL database and neglected more lightweight databases like SQLite to achieve a solution without the extra effort of becoming familiar with the details of other databases accepting that our database is a bit more heavyweight.

### 4.2.4 Authentication

In the context of authentication, facing the wish to not be forced to use pre-existing accounts like Google or Facebook accounts we decided to implement our authentication using JSON Web Tokens (JWTs). and neglected existing frameworks like OAuth 2.0 in combination with the Google API accepting that we would have to put in more work to realize authentication.

## 4.3 C4 Diagrams

To get a quick overview of our backend architecture, in this section we show a couple of C4 Model diagrams of our architecture. We have decided to skip the Code Diagram as it is far too detailed, and we don't think that this level of detail is required here.<sup>3</sup>

---

<sup>3</sup>Simon Brown. *The C4 model for visualising software architecture. Context, Containers, Components, and Code*. URL: <https://c4model.com/>. (accessed: 09.12.2022).

### 4.3.1 Context Diagram

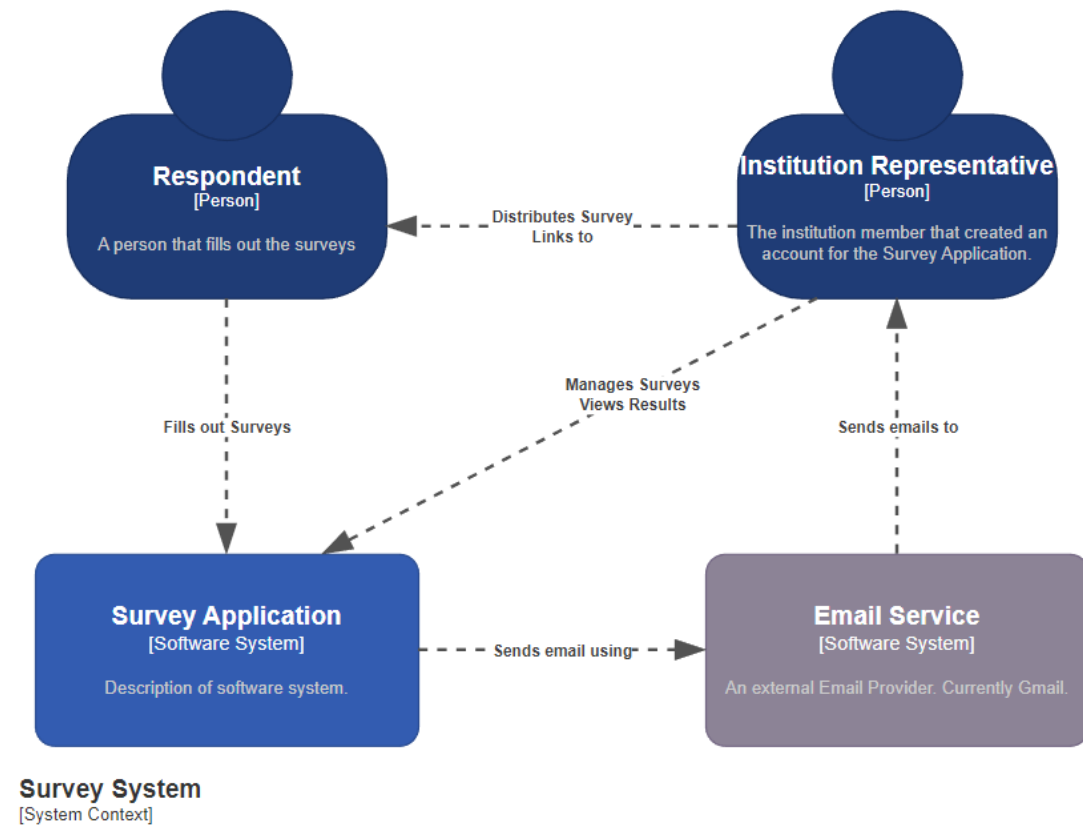


Figure 4.2: Context Diagram

Our application has two main actors. The institution representatives manage surveys of their respective institutions. This includes the creation of surveys as well as the deletion of surveys. In addition to that, they are responsible for distributing the survey links to the respondents that will fill out those surveys. Once a survey is over, the institution representatives can view the results.

The respondents receive those links and can then fill out the surveys.

Our survey system makes use of Gmail as an external service to send various emails to the institution representatives.

### 4.3.2 Container Diagram

Our application consists of the following containers:

At the entry point, we have put up a proxy that is responsible for redirecting requests to the respective endpoint. The proxy is also responsible for HTTPS encryption, making sure that the messages exchanged are confidential. Besides HTTPS encryption the proxy

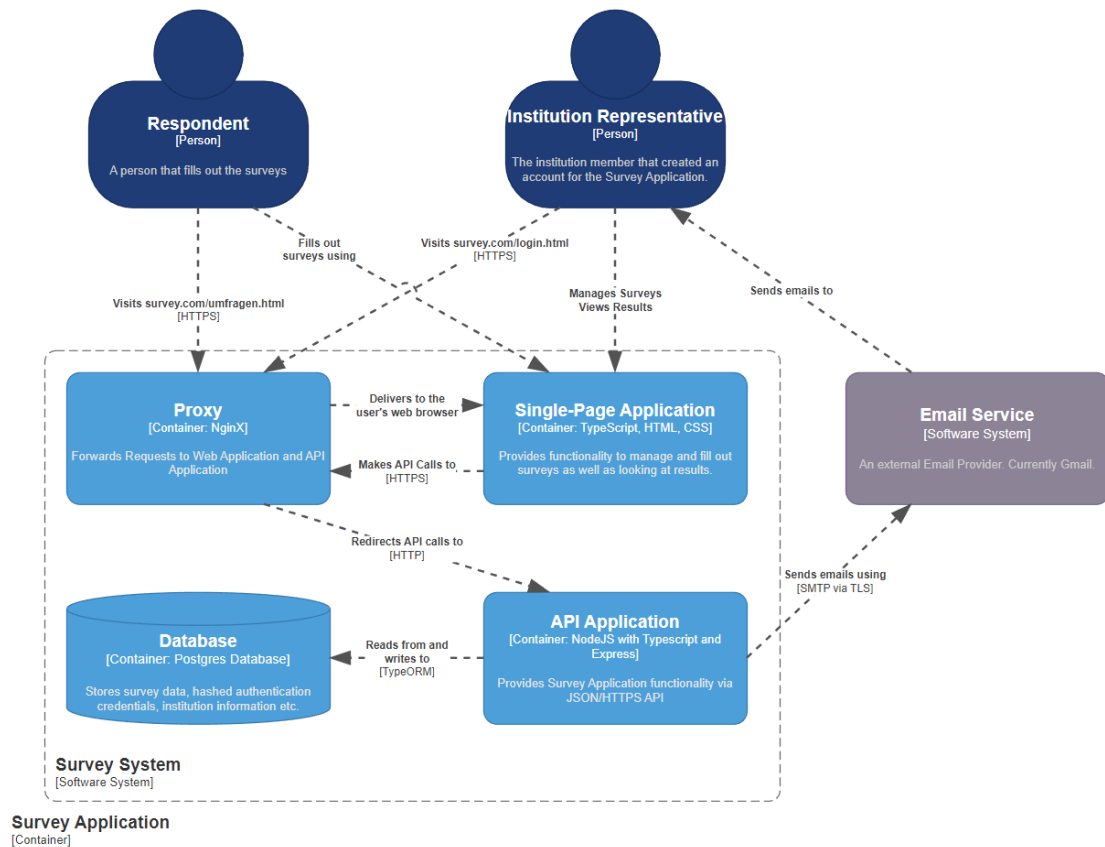


Figure 4.3: Container Diagram

serves another purpose, which is to make it look like all resources are located in the same place. This removes the need for Cross-Origin-Resource-Sharing (CORS), which also increases security.

To be able to deliver our Single-Page Application (SPA) to the user, we use an NGINX server. Non-API requests are automatically redirected to the NGINX server. Requests made by the SPA are redirected to the API Server.

The API itself is located behind the proxy as well. It runs on a NodeJS docker container.

The database is accessed by the API Server. It runs on a Postgres docker container.

### 4.3.3 Component Diagram

In the API Application, the requests made by the SPA are delivered to the correct controllers via express routers. The controllers are responsible for executing the business logic and returning the responses to the user. During the startup process, the dependencies are injected into the controllers, which pass them to the business logic. Due to this, the business logic can be decoupled from its dependencies, like repositories for database



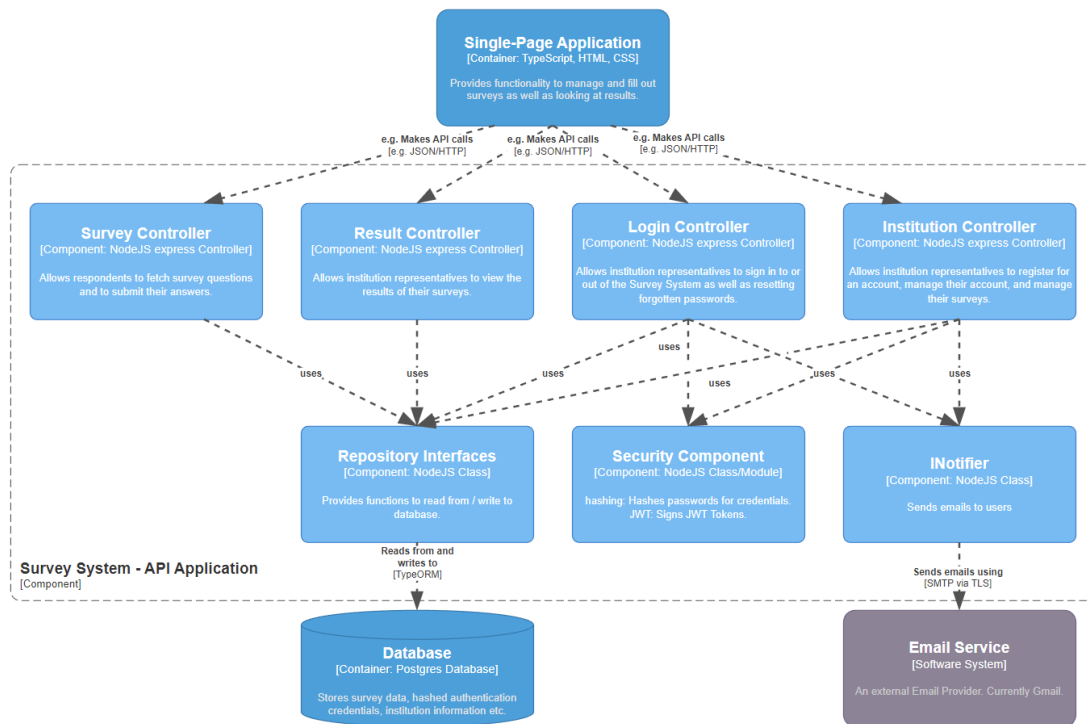


Figure 4.4: Component Diagram

access and notifiers to send emails to the users. The controllers are also responsible for catching any errors that might have been thrown by the business logic. The creation of a custom error class allows different error messages and HTTP error codes to be sent while keeping the controller logic simple.

The business logic is called by the controllers. Each business logic module makes use of at least one repository to load or persist data. As already mentioned before, the business logic only depends on an interface, which means the repository implementations can be exchanged freely.

Some business logic modules also make use of security components, which are the hashing module and the JWT module. As the name suggests, the hashing module is responsible for the hashing of passwords. Thus it is used in the login part of the business logic, where the hashing module is also used to check passwords for login attempts. The JWT module is responsible for the creation of JWTs. Again, the login business logic module makes use of this component to create access tokens, so that the users can log in.

# Chapter 5

## Database

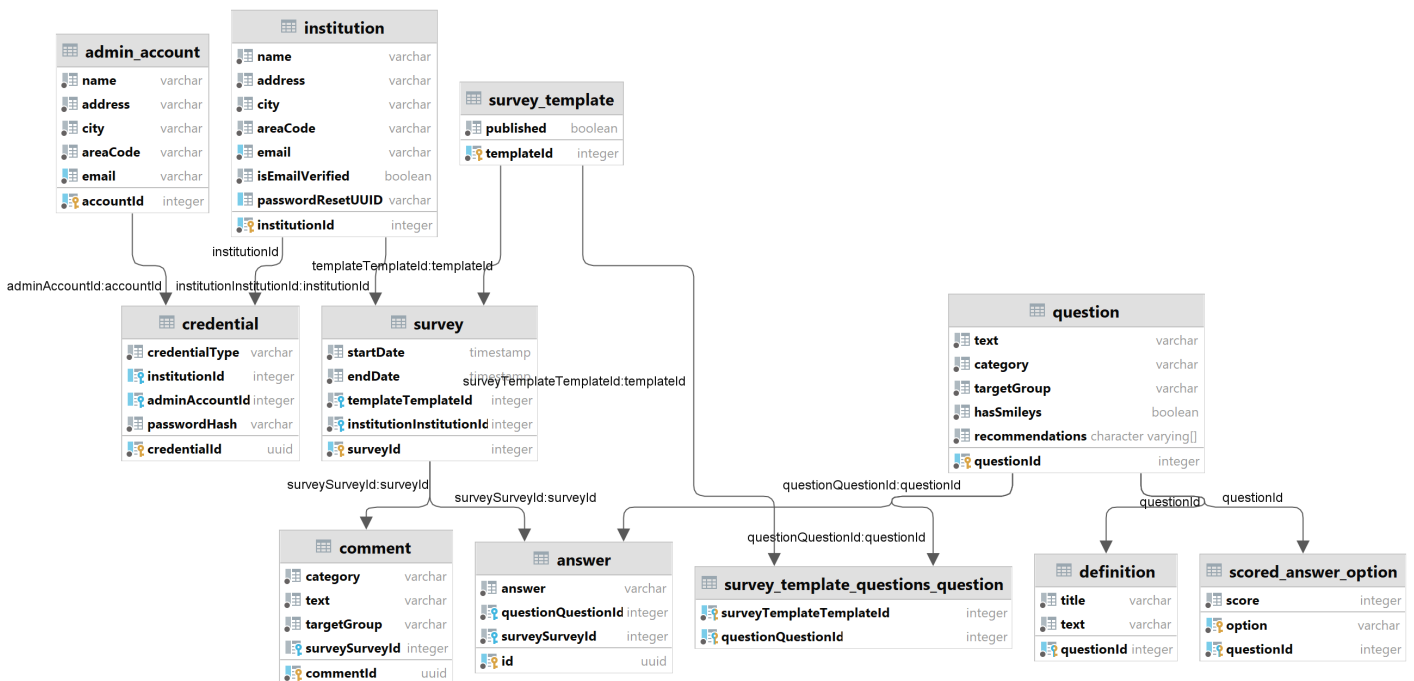


Figure 5.1: Database Diagram

The database was designed with the intention to reduce the amount of overhead in the surveys, while at the same time allowing the Survey Administrators to add and remove questions, while making sure that backwards compatibility is guaranteed.

## 5.1 Database Tables

### 5.1.1 Survey Template

The survey template is the core of survey management. It keeps track of the questions belonging to a survey.

If a question is deleted, it cannot just be removed from the database. It has to be preserved for past surveys, in case an institution representative wants to look at the results. This restriction led to the idea of survey templates that would keep track of all the questions that together form a survey. Each time a question is added or removed, a new survey template is created. Thanks to that, older surveys can still remember which questions were answered, which is important for the calculation of points and derivation of recommendations.

If a question is removed by the Survey Administrator, it does not get deleted from the database. Instead, a new survey template is created, and all the references to the questions in the old survey template are copied. Only the deleted question is not referenced in the new template. The overall number of questions in the database remains unchanged. And if an institution wants to look at its older results, it can still see the results from the question that has been removed, as it is still available in the old template.

To make this possible the `published` column has been added to the survey templates. As long as a survey template is not published it can be edited by the Survey Administrators. Once it is published, however, it becomes immutable, to make sure that everyone with the same survey link gets the same questions.

### 5.1.2 Institution and Admin Account

Both `admin account` and `institution` are tables that represent persons. They share many attributes and reference the credential table. The main difference is that an institution can have surveys, while an admin account can't. They are used to manage personal and login information.

### 5.1.3 Credential

The `credential` table is used to manage login information. It holds the password hash of a user. It either references an admin account or an institution. The credential type is set to either 'Institution' or 'AdminAccount', to be able to tell which of the two columns 'institutionId' or 'adminAccountId' is not null.

### 5.1.4 Survey

A survey is a concrete instance of a survey template. It references `survey template` to be able to tell which questions belong to that survey. As a survey always has to belong to an institution, this table also references the `institution` table.

### 5.1.5 Question

A question contains a `text` column, which is the question itself. Every question belongs to exactly one category. Since there are different surveys for different target groups, each question must also keep track of the target group it is intended for. Valid values for target groups are 'Angehoerige', 'Fachkraefte', and 'Bewohnende'. In addition to this, a question has an array of recommendations, that are displayed in the results if the question is not answered sufficiently well.

### 5.1.6 Definition and Scored Answer Option

Each question has three to four Scored Answer Options (SAO). Every SAO contains the score of this particular answer option, the answer option itself ('Ja', 'Nein', 'Manchmal', 'NichtBeurteilbar'), and references a question. Using these two pieces of information, the score of a single answer can be determined.

Each question can also have an array of definitions, which contain complicated words and their description.

### 5.1.7 Comment and Answer

Both the `comment` and `answer` tables are used to persist the information filled out by the user. Each comment belongs to a certain category and a target group. This information is needed for the correct display of the results.

Each answer references a question. The answer column of the table contains either the string 'Ja', 'Nein', 'NichtBeurteilbar' or 'Manchmal'. With those two pieces of information, the score of a single answer can be determined.

**Part IV**  
**Implementation**

# Chapter 6

# Frontend

This chapter describes various implementation aspects of the frontend.

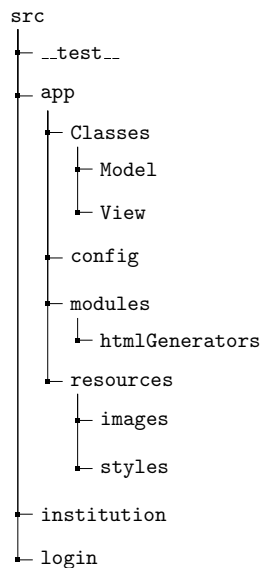
## 6.1 Application Structure

The file structure of the frontend intertwines the application structure with the underlying HTML pages. The MVC (Model View Controller) pattern structures the application in distributed directories. As a variant, the MVC pattern follows a semantic cut, so that the respective HTML page holds the logic of the JS files together.

The folder `app/Classes/Model` contains the classes for the business logic

The folder `app/Classes/Views` is mainly limited to the view of the MVC pattern.

The folders `institution` and `login` correspond to the relative path of the site:



## 6.2 Access Token Flow

When logging in via GET `/api/login`, the browser saves an access token in the browser's Session Storage, which is only valid until the time limit has been reached or until the browser tab has been closed. Persistency is achieved by a key-value pair by a setter or getter in TypeScript as part of the standard library. This access token is base64 encoded and can be used for various API calls to allow the user to make appropriate requests to the backend via HTTP verbs GET, DELETE, POST, PATCH, and UPDATE.

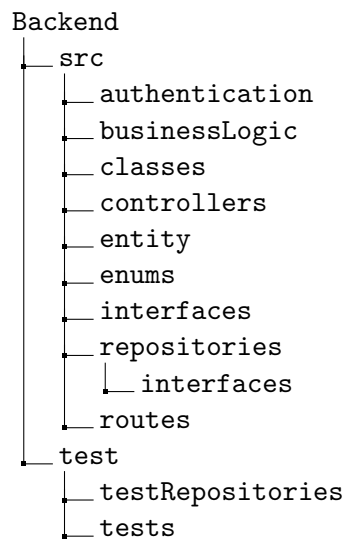
For example, starting a new survey: `/api/institutions/institutionId/surveys/start`. The token checks its validity at regular intervals, by issuing a renewal request. If the renewal of the token was successful, the login remains persistent by replacing the old token with a new one, otherwise, the user ends up redirected to the login page.

# Chapter 7

## Backend

This chapter covers the implementation of the backend a bit more in-depth.

### 7.1 File Structure



**Authentication:** The `authentication` folder contains all classes and modules necessary to create and verify JWTs as well as hashing and checking passwords.

**Business Logic:** The `businessLogic` contains most of the functionality of the application.

**Classes:** The `classes` folder contains all the classes that didn't belong in any other folder.



**Controllers:** The `controllers` folder contains the controllers, which are responsible for answering requests. They extract the necessary information from the requests, call the corresponding business logic functions, and then send back the responses.

**Entities:** The `entity` folder contains all the entity classes. Created entities are stored in the database.

**Enums:** The `enums` folder contains all enums.

**Interfaces:** The `interfaces` folder miscellaneous interfaces.

**Repositories:** The `repositories` folder contains all repositories. The repositories are used to access the database.

**Repositories/Interfaces:** The `repositories/interfaces` folder contains all interfaces that the repositories must implement.

**Routes:** The `routes` folder contains all the routes, which define the mapping of requests to controller functions. The mapping is based on the URL and request method.

**Test-Repositories:** The `testRepositories` folder contains all the repository classes that are used for testing. Each test repository must implement an interface defined in the `repositories/interfaces` folder.

**Tests:** The `test` folder contains all the unit tests that are run during test execution.

## 7.2 Interesting Code Snippets

This section contains interesting code snippets from the backend. These code snippets show interesting patterns like dependency injection or interesting test concepts.

### 7.2.1 Injecting Dependencies

To avoid dependency cycles, dependency injection was used to inject the repository and notifier classes into the controllers. Each controller has a function that sets the repository (and optionally the notifier), so they don't have to import the data source manager from the `index.ts` file.

```

1  # resultController.ts
2
3  let resultRepository: IResultRepository;
4  let institutionRepository: IIInstitutionRepository;
5
6  export function setResultRepository(repositories: {
7      resultRepository: IResultRepository,
8      institutionRepository: IIInstitutionRepository,
9  }) {
10     resultRepository = repositories.resultRepository;
11     institutionRepository = repositories.institutionRepository;
12     return (req: Request, res: Response, next?: NextFunction) => {
13         next();
14     };
15 }

```

The function `setResultRepository` is meant to be called as an express middleware from the `index.ts` file. Each controller has one function similar to this one, where the dependencies are set. By doing this, the dependencies can be instantiated once and passed to each controller. There is no singleton needed, and by programming against interfaces, the business logic can be decoupled from its dependencies.

```

16 # index.ts
17
18 const repositories = {
19     resultRepository: new ResultRepository(AppDataSource.manager),
20     questionRepository: new QuestionRepository(AppDataSource.manager),
21     answerRepository: new AnswerRepository(AppDataSource.manager),
22     surveyRepository: new SurveyRepository(AppDataSource.manager),
23     commentRepository: new CommentRepository(AppDataSource.manager),
24     institutionRepository: new InstitutionRepository(AppDataSource.manager),
25     surveyTemplateRepository: new SurveyTemplateRepository(
26         AppDataSource.manager
27     ),
28 };
29
30 app.use('/api/resultate', resultRouter, setResultRepository(repositories));

```

In the `index.ts` file, all repositories are instantiated once. The entity manager is passed as an argument. While mapping the route `/api/resultate` to the `resultRouter` module, the `setResultRepository` function is called as an middleware, setting the repositories in the `resultController`.

## 7.2.2 Testing Functions that use the Notifier

Classes implementing the `INotifier` interface are responsible for sending messages via email (or potentially other means of communication). The sending of emails can potentially take a big amount of time, which is why the `sendMessage` function is not awaited in the business logic. This, however, means that the business logic can't check if the function was executed properly.

To still be able to test the behavior of the business logic in the unit tests, a fake of the `INotifier` interface is created, that saves the relevant information in its properties. Those properties can then be checked to make sure that the business logic calls the right functions at the right time. The `INotifier` fake also contains a property called `shouldFail`, which indicates if the `sendMessage` function should fail and throw an error. This can be used in case the `sendMessage` function is awaited, to be able to test the error handling of the calling business logic.

```
1  # TestHelper.ts
2
3  const testNotifier = {
4    shouldFail: false,
5    sendMessage: (messageInfo: {
6      message: string, subject: string, recipient: string
7    }) => {
8      testNotifier.recipient = messageInfo.recipient;
9      testNotifier.message = messageInfo.message;
10     testNotifier.subject = messageInfo.subject;
11     if (testNotifier.shouldFail) {
12       return Promise.reject(
13         new Error('Sending message failed! (This is a mock error)')
14       );
15     }
16     return Promise.resolve();
17   },
18   recipient: <string> undefined,
19   message: <string> undefined,
20   subject: <string> undefined,
21 };
```

```

22 # InstitutionLogicTest
23
24 const notifierMock = TestHelper.createTestNotifier();
25 const notifier = notifierMock as unknown as INotifier;
26
27 const institutionInfo = {
28     name: 'Inst.',
29     address: 'Addr.',
30     city: 'City',
31     areaCode: '4564',
32     email: 'inst@mail.com',
33 };
34 const { email } = institutionInfo;
35
36 await institutionLogic.createNewInstitution(
37     institutionInfo,
38     institutionRepository,
39     notifier
40 );
41
42 const expectedToken = JWT.generateEmailConfirmationToken(institutionInfo);
43
44 notifierMock.recipient.should.equal(email);
45 notifierMock.subject.should.equal('Registration');
46 notifierMock.message.should.match(/.*emailBestaetigung\.html\?token=.*/);
47 notifierMock.message.should.contain(expectedToken);

```

The test case shown above first creates a new test notifier fake using the `TestHelper`. The fake must be "casted" to `INotifier`, so it can be passed to the `createNewInstitution` function of the `institutionLogic`. Once the `createNewInstitution` function is done, the fake notifier's properties can be checked to make sure that the `sendMessage` function has been called correctly.

### 7.2.3 Testing Exceptions

As it turned out during the writing of the tests using `chai` and `mocha`, it was not very easy to test if the functions would throw the correct exceptions. This is partially due to `chai` not checking the messages of the errors. In addition, we have defined a custom error class called `ApiError`, which also includes the HTTP error code that should be returned, to simplify error handling. This however made the testing of errors even harder, since the functions provided by `chai` would never check those error codes. Also checking for the correct error type caused some problems.

To circumvent all of the problems mentioned above, the `TestHelper` class offers four functions. Two functions to check the errors thrown by non-async functions, one for

checking for ApiErrors, and one for checking any other kind of Error. And an additional two functions to check errors thrown by async functions, again, one for ApiErrors and one for all other Errors.

In this section, one of these functions is shown in detail, the remaining three are very similar and thus not shown.

```
1  async function assertThrowsApiErrorAsync(  
2    f: () => Promise<unknown>,  
3    message: string,  
4    errorCode: number,  
5  ) {  
6    try {  
7      await f();  
8      chai.assert.fail('No error was thrown!');  
9    } catch (e) {  
10     assertApiError(e, message, errorCode);  
11    }  
12  }  
  
13  function assertApiError(error: Error, message: string, errorCode: number) {  
14    if (error instanceof ApiError) {  
15      if (  
16        message === error.message  
17        && errorCode === error.getErrorCode()  
18      ) {  
19        return;  
20      }  
21      chai.assert.fail('Actual: {  
22        message: ${error.message},  
23        code: ${error.getErrorCode()}  
24      } is not the same as expected: {  
25        message: ${message},  
26        code: ${errorCode}  
27      }.');  
28    }  
29    chai.assert.fail(`  
30    Actual: {  
31      message: ${error.message}  
32    } is not the same as expected: {  
33      message: ${message}, code: ${errorCode}  
34    }.`);  
35  }
```

The `assertthrowsApiErrorAsync` function takes an async function, the expected error message and the expected error code as parameters. It calls the provided functions,

and if it throws no error, it causes the test to fail. If it throws an error, the error is checked by the `assertApiError` function. This function checks the error type, message and code of the thrown exception, and if there is a mismatch, it causes the test to fail.

Using these custom error-check functions eliminated all the previously mentioned problems that we had with the testing library.

The functions can then be used like this:

```
36 # LoginLogicTest.ts
37
38 await TestHelper.assertThrowsApiErrorAsync(
39     () => loginLogic.resetPassword(
40         passwordResetToken,
41         newPassword,
42         institutionRepo,
43     ),
44     'Dieser Link ist ungültig!',
45     400,
46 );
```

# Chapter 8

## Proxy

This chapter highlights the configuration that we used to implement the NGINX proxy.

### 8.1 Configuration

The configuration file is made up of two parts.

The first part is the HTTP endpoint (lines 3 through 14). The purpose of this part is to redirect all HTTP requests to the HTTPS endpoint (lines 11 through 13). The only requests that are not redirected are the requests to `/.well-known/acme-challenge/`, which are used to obtain a TLS certificate (lines 8 through 10).

The second part is the HTTPS endpoint (lines 16 through 29). To enable TLS lines 21 through 22 specify the required files. Requests starting with `/api` are forwarded to the backend server (lines 24 through 36), while the remaining requests are forwarded to the frontend server (lines 27 to 29).

To generate the TLS certificate we use the `certbot`<sup>1</sup> docker image, which helps to easily generate certificates via Let's Encrypt<sup>2</sup>

---

<sup>1</sup>*certbot*. URL: <https://certbot.eff.org/>. (accessed: 14.12.2022).

<sup>2</sup>*Let's Encrypt*. URL: <https://letsencrypt.org/>. (accessed: 14.12.2022).

```
1 # default.conf
2
3 server {
4     listen 80;
5     listen [::]:80;
6     server_name <hostname>;
7
8     location /.well-known/acme-challenge/ {
9         root /var/www/certbot;
10    }
11    location / {
12        return 301 https://<hostname>$request_uri;
13    }
14 }
15
16 server {
17     listen 443 default_server ssl http2;
18     listen [::]:443 ssl http2;
19     server_name <hostname>;
20
21     ssl_certificate /etc/nginx/ssl/live/<hostname>/fullchain.pem;
22     ssl_certificate_key /etc/nginx/ssl/live/<hostname>/privkey.pem;
23
24     location /api {
25         proxy_pass http://backend:8000/api;
26     }
27     location / {
28         proxy_pass http://frontend:80/;
29     }
30 }
```



## Chapter 9

# Quality Measures

This chapter describes the various quality measures we have put in place to make sure that our code is as clean as possible and working as expected.

### 9.1 Coding Guidelines

- General
  - Code needs to be committed frequently with descriptive commit messages.
  - Side effects should be avoided whenever possible.
  - Comments may only be used if they are helpful.
  - Global variables should be avoided if possible.
  - Exceptions are preferred over error codes.
  - Nesting should not be deeper than 2 levels.
  - No duplicate code. (DRY)
  - Functions should be kept short (less than 11 lines).
  - Create feature branches, before every merge, tests must be run.
- Formatting
  - Line length should not exceed 120 characters.
  - Style guide: ESLint + AirBnB
  - Code must be auto-formatted before being committed.
  - Code must pass all tests before being committed.
- Naming
  - Function names should be verbs.
  - Class names should be nouns.

- Longer, more descriptive names are preferred over short ones.
- No abbreviations in names.
- No 1-character names.

## 9.2 Test Concept

This section describes the test concept for the project. The concept should be seen as a consecutive description for developers, beginning from Unit Tests up to Usability Tests in corresponding order. The concept outlines the verification of requirements on the system.

### 9.2.1 Unit Testing

Unit Tests should often be carried out under the AAA paradigm, which stands for Arrange, Act, Assert. This approach includes the following:

- Arrange: Setting up tests for a particular piece of code such as methods, variables and classes.
- Act: Performing the actual testing after the code has changed.
- Assert: Observing the resulting behavior and checking whether expectations were met. The code should be changed to achieve a positive test result.
- Each operation returns the correct result in case the operation is applied correctly.
- Each operation throws the correct exception in case the operation is applied incorrectly.

### 9.2.2 Integration Testing

The Docker containers should be tested for inter-communication via API calls. Integration tests verify intra-connectivity between the frontend, backend and database containers. In general, the procedure includes verification on:

- Correctly called operations.
- The combination of individually called operations.
- Asynchronous timeout requests and responses.
- On database queries.

## 9.2.3 Frontend

### Usability

Usability Tests are the main derivation of functional requirements. As such, they are a quality measuring tool for the entire project. NFRs 1.0 - 1.4 fall at least in one of the following two categories:

- User experience of the main application (survey & results) for disabled people.
- Accessibility of the main application (survey & results) for disabled people.

Usability tests are partially automated, and thus remain mostly manually tested in this concept due to their complexity. The effectiveness of time and resources for our usability testability is maximized with the following tools:

- axe as VSC Linter plugin<sup>1</sup> for accessibility.
- axe DevTools as Chrome Plugin<sup>2</sup> verifies in-browser accessibility.

Both tools provide insights into the previously mentioned user experience and accessibility under the WCAG 2.1 standard.<sup>3</sup> With these tools, technical requirements were tested and verified.

Some NFRs require manual testing since only a human being can judge how good or bad a user experience is. The reason is, that "semantic correctness" is hard or impossible to test with a tool, that is not capable of thinking like a human being. To verify the early requirements (FRs & NFRs) acceptance tests were used in real-world situations within a dockerized test environment. Furthermore, the ZEN was asked to test against the requirements of the target audience (disabled people) to meet the expectations.

## 9.2.4 Backend

On the backend, unit tests are used to make sure everything is working properly. To allow the testing of the complete business logic, dependency injection is used. Instead of the real repositories, for example, repository fakes are passed as parameters that eliminate the need to include the database in the tests. Besides performance improvements, this also makes tests independent of each other, as they don't make changes on a common database.

---

<sup>1</sup>*axe Linter for Visual Studio Code.* URL: <https://www.deque.com/blog/shift-further-left-with-deques-axe-linter-for-vs-code/>. (accessed: 12.12.2022).

<sup>2</sup>*axe DevTools web accessibility.* URL: <https://chrome.google.com/webstore/detail/axe-axe%20DevTools-web-accessib/lhdoppojpmngadmndnejejpokejbdd>. (accessed: 12.12.2022).

<sup>3</sup>*Web Content Accessibility Guidelines (WCAG) 2.1.* URL: <https://www.w3.org/TR/WCAG21/>. (accessed: 21.12.2022).

### **9.3 Workflow**

The workflow is also part of our quality assurance. On the backend repository, we created a pipeline that runs the tests automatically when creating a merge request. This adds another layer of quality assurance, making sure that the tests are not only passing on the local machine. Only once those tests pass, are we allowed to merge the feature branch into the main branch.

**Part V**  
**Discussion**

# Chapter 10

## Results

This chapter describes the achieved and not achieved goals, which are closely related to the FR and NFRs.

### 10.1 Functional Requirements

UC1 - UC4 partially satisfied the MVP requirements according to acceptance tests. UC5 to UC7 were not implemented and thus didn't satisfy all the requirements of a MVP. As a result, the MVP was only partially satisfied.

UC#	Description	Satisfied
1	Respondents are able to fill in surveys.	Yes
2	Institutions are able to view the results and recommendations of filled-in surveys.	Yes
3	Institutions are able to start surveys.	Yes
4	Institutions are able to end surveys.	Yes
5	Survey Administrators aren't able to manage questions.	No
6	ZEN Administrators aren't able to manage Survey Administrators.	No
7	ZEN Administrators aren't able to manage Institution Representatives.	No

## 10.2 Non-Functional Requirements

NFR#	Description	Satisfied
1	People with cognitive impairments can fill out surveys easily.	Yes
1.1	The acceptance tests, automated tests and linter showed, that the survey met the WCAG 2.1 AA guidelines.	Yes
1.2	Automated tests showed that the survey can be completed and navigated using a keyboard.	Yes
1.2	An acceptance test by an institution showed that the survey was not displayed optimally on a mobile device.	No
1.2	Acceptance tests on screen readers were not performed.	No
1.2	The acceptance tests and automated tests showed that the survey can be filled out using a mouse.	Yes
1.3	The acceptance tests showed that the content is readable and easy to understand.	Yes
1.4	The acceptance tests, automated tests and linter showed, that the survey met the robustness requirements by following semantically correct HTML.	Yes
1.4	Automated tests and linter showed, that ARIA tags were not necessary for the project scope.	Yes
2	The acceptance tests showed that the survey is anonymous thus a respondent was not asked to provide personally identifying information.	No
3	Test results showed that the backend met the requirements.	Yes
3	The frontend didn't meet the requirements due to the lack of tests.	No
4	Manual acceptance tests showed that components are dockerized.	Yes
5	Manual acceptance tests showed that environment variables were kept secret.	Yes
6	Manual acceptance tests showed that a linter prevented bad practices.	Yes
7	Manual acceptance tests showed that the page response time on average is no longer than 3s after clicking the 'Start Survey' button.	Yes
8	Manual acceptance tests showed that the page response time on average is no longer than 3s after clicking the 'See Survey Results' button.	Yes
9	Manual acceptance tests showed that the page response time on average is no longer than 3s after clicking the 'Survey Finished' button.	Yes

## 10.3 Screenshots

**Umfrage zur Sexualität in Institutionen**  
Basierend auf der UNO Behindertenrechtskonvention (BRK)

Wählen Sie hier, wer Sie sind

Ich bin

Klicken Sie hier, um die Umfrage zu starten ✓

Nachdem die Umfrage gestartet wurde, haben Sie eine Liste mit verschiedenen Fragen vor sich. Bitte kreuzen Sie die Antwort an, die für sie passt.

Figure 10.1: Extract of the Survey Entry Page

**Wohnen**

Haben Sie ein Einzelzimmer?

Bitte wählen Sie eine Antwort aus.

Ja 😊

Nein 😞

Kann ich nicht beantworten ✕

✕ Umfrage abbrechen.

Figure 10.2: Extract of the Survey Page

Haben Sie in diesem Wohnheim schon einmal sexuelle Gewalt erlebt?

Bitte wählen Sie eine Antwort aus.

Ja 😞

Nein 😊

Teilweise / Manchmal 😞

Figure 10.3: Smileys and Colors can be inverted, in case 'Yes' is the bad Answer.



Befürworten Sie Doppelzimmer für Paare?

Bitte wählen Sie eine Antwort aus.

Ja

Nein

Kann ich nicht beantworten ✕

Figure 10.4: If the Question is just about the Opinion, Smileys and Colors can be removed.

**Wohnen**

Etwas mitteilen:

Kommentare zum Thema «Wohnen» können Sier hier hinschreiben.

zurück zur letzten Frage weiter zur nächsten Frage

Figure 10.5: Field for Comments during the Survey.

**Gesundheit und Pflege**

Werden Ihre Bedürfnisse bei der Intim-Pflege berücksichtigt?

was bedeutet **Intim-Pflege**?

Bitte wählen Sie eine Antwort aus.

Figure 10.6: Button to display Definitions.

The slide features a dark background with a white 'X' icon in the top right corner. On the left, there is a blue square icon containing a white figure of a person. To the right of the icon, the title 'Intim-Pflege' is written in white. Below the title, a paragraph of white text provides a definition: 'Intimpflege ist die Körper-pflege im Bereich der Geschlechts-Organen. Bei der Intimpflege werden die Geschlechts-Organen gewaschen.'

Figure 10.7: Definition of a Hard Word.

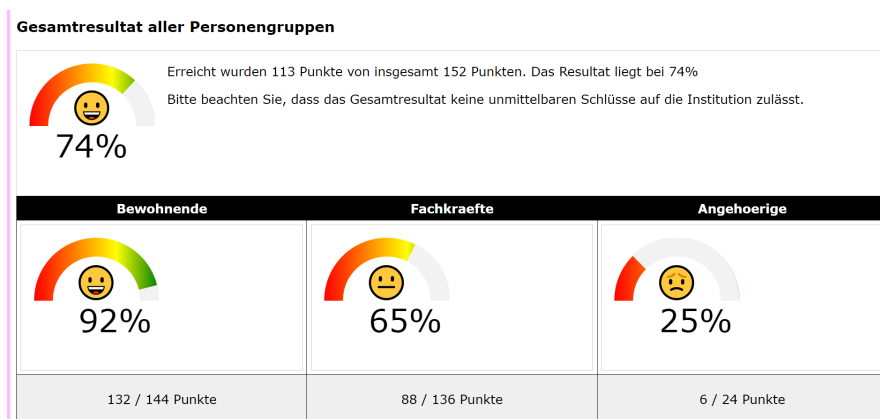


Figure 10.8: Extract of the Results Page for Institution Representatives

## Registrierung für Institutionen

**Institutionsname**

**Adresse**

**Ort**

**Postleitzahl**

**Email**

Registrieren

Figure 10.9: Registration Input for Institutions.

## Login für Institutionen

**Email**

**Passwort**

Error: ✘ falscher Benutzername und / oder falsches Passwort.

Login

[Passwort vergessen?](#)  
[Registrieren](#)

Figure 10.10: Login Screen for Institutions.

## Umfrageübersicht

Erstellen Sie eine Umfrage.

Identifikation	Optionen	Start	Ende
1	<input type="button" value="Löschen"/>	2022-11-27T21:39:58.081Z	2022-11-27T21:39:58.081Z

### Umfrage erstellen

Umfragedauer in Tagen:

Figure 10.11: Overview of all the Institution's Surveys.

## Informationen zur Institution

<b>Institutionsname</b>
Lindenhof
<b>Adresse</b>
Sosostrasse 12
<b>Ort</b>
Dahausen
<b>Postleitzahl</b>
0000
institution@mail.com
aktualisieren

Figure 10.12: First Half of the Institution Management Screen to update Information.

## Passwort erneuern

<b>Altes Passwort</b>
Passwort eingeben
<b>Neues Passwort</b>
Passwort eingeben
aktualisieren

## Email ändern

<b>Email</b>
Email eingeben
aktualisieren

Figure 10.13: Second Half of the Institution Management Screen to update Information.

# Chapter 11

## Conclusion & Next Steps

In this chapter, the project team reflects on and evaluates the results achieved. Therefore, goal achievement is measured by comparing it to the task and goal of the work.

The following sections discuss the facts surrounding the project concerning FRs & NFRs and illuminate the frontend and the backend from different perspectives. At the same time, attempts are made to find new solutions in the discussion.

### 11.1 Functional Requirements

According to the test results, UC1 to UC4 were implemented. Other requirements added by the ZEN that have not been met are discussed in the frontend and backend sections.

#### **Additional Requirements**

In the course of the transition phase, new requirements came into the project. Some requirements were prioritized by the ZEN due to the limited time in the project, therefore some requirements could no longer be met.

Among other things, it was not possible to introduce personal recommendations for disabled people based on their survey results, as proposed after the first project presentation.

In addition, another additional requirement could not be met, which should allow the anonymous sharing of survey results of institutions with the ZEN.

A trailing requirement in the final week of construction addressed optimization for mobile devices. Since building time has already passed and the pending issues still have to be dealt with, this requirement could not be met as well.

One requirement was removed from the project by the ZEN. This concerned the im-

plementation of a printable survey for respondents.

## **Basic Requirements**

UC5 to UC7, which particularly affected the CRUD operations for the ZEN, could not be completed due to various reasons. The reasons will be discussed in the section Reflection Frontend.

### **11.1.1 Non-Functional Requirements**

The main intention and thus the core criterion of the ZEN was that institutions could evaluate themselves based on the responses of respondents (NFR 1). With the help of non-representative acceptance tests conducted by the ZEN, it could be shown that people with cognitive impairments were able to complete the survey. Subsequent acceptance tests thus theoretically ensured a barrier-free user experience for disabled people.

The ZEN stipulated that the survey had to be usable with a mouse, keyboard, touch screen and screen reader (NFR 1.2). The project team did not establish direct contact with the test subjects, so it was not possible to finally explain which device inputs the test subjects had used to carry out the test.

A further requirement concerned the simple comprehensibility of the content of the survey text (NFR 1.3). It was necessary to ensure an implementation so that definitions of complicated words could be made accessible and explained to disabled people.

The backend has stable test results. The code ins the frontend is only partially maintainable with the current implementation (NFR 3). Tests were always part of the initial requirements of the project.

The linter was previously used to set framework conditions (NFR 6). Some linter suggestions were ignored because they weren't practical in the context. However, the majority of the linter suggestions resulted in better code quality.

Manual tests have shown that the response time to start a survey Displaying results and submitting results was less than 3s (NFR 7-9). These test results were not logged.

## **11.2 Reflection Frontend**

The tight schedule is due to the time that was spent on change requests for UC1, as mentioned in the previous subsection on Basic Requirements. In the course of the main requirements, however, it can be argued that these requirements had a very high value, in which the time was well invested. This was possible due to the design decisions made by the team in cooperation with the ZEN and disabled people re-evaluating the changing requirements. On the technical side, time was invested in the semantic correctness of

the presentation so that this requirement could be met. The presentation and content of the entire survey were planned with the greatest possible care and implemented under the WCAG2.1 guidelines (NFR 1.1, NFR 1.4).

Unfortunately, a considerable part of the available time was invested in UC2 (View Results) in terms of accessibility and usability, so that overall there was less time for the development of the additional requirements from UC5 to UC7. The situation described was subject to a change in requirements, although it was initially considered that the results should be visible to all respondents including disabled people. Afterward, however, only institution representatives should be able to see the results, according to the ZEN. Had this requirement been clear from the outset, at least the semantic correctness with all implementation details for accessibility & usability could have been dispensed with.

### **11.2.1 axe DevTools Testing Limitations**

Concerning the accessibility tester axe DevTools, the limitation of partial testability is worth mentioning. In general, this testing tool was subject to certain restrictions that did not allow any insight into the user guidance. The evaluation of a tool intended for this purpose might have yielded further insights.

Additional limitations regarding axe DevTools are the erroneously made flags that should have violated the semantic correctness of HTML. For example, empty a-tags were classified as "critical", although the empty tags were only used to control the flow of pop-up definitions. We decided not to change this code to ensure a good user experience regardless of failed test results.

Another hurdle was the evaluation of the test results at the end of the project because the test results in the appendix corresponded to obsolete test results. This is because the evaluation period for the chargeable test environment axe DevTools had ended at the time of the construction phase of the project. As a result, the test results could not be renewed, although appropriate countermeasures were taken after the tests. The additional limitations in the last section mentioned, however, were also included in the test logs.

### **11.2.2 Maintainability and further Testing**

Regarding the frontend maintainability, only a few integration tests are present for the frontend (NFR 3). Testability was dispensed with in favor of functionality in terms of the changing requirements on the part of the client. The implementation followed the MVC (Model View Controller) pattern, which makes the testability of the frontend more difficult. The lack of structuring in the code generally required architectural decisions, which would have simplified the implementation.

Therefore, for multi-class projects like this, following an MVVM (Model View View-Model) pattern approach might be more appropriate. MVVM separates the development

of user interfaces from business logic and behavior. Declarative data bindings allow UI and development work to occur separately within the same codebase,

Moreover, testing was made more difficult because the use of global variables had a fixed place in the code, so that testability could no longer be guaranteed as a requirement. All of the problems mentioned meant that this requirement was not met.

### 11.2.3 Evaluation on Technology

In the evaluation phase at the beginning of the project, the vue.js framework was deliberately avoided. A further evaluation phase with React was not subsequently carried out. In retrospect, the evaluation of the React Framework would probably have been better, so that the complexity could have been mitigated.

## 11.3 Reflection Backend

While not all use cases were implemented yet and the MVP has not been reached yet, we managed to provide most of the important functionality to make the result usable. All the self-evaluation functionality for the institutions has been implemented, which is the most used feature.

The bar for the MVP was probably set a bit too high to be realistic, especially considering the quality measures, which required a big amount of the code to be tested.

There were many things that we have done for the first time in this project, like for example the deployment including proxy and TLS encryption, as well as the deployment itself, which we had never done before. Working on all these things has certainly helped to expand our toolkit for future projects and will probably be a valuable experience that will help with the implementation of similar upcoming projects.

This being said, there is still some more work that could be done on the currently implemented functionality. For starters, we would rethink the database design a slightly bit.

As of right now, there are two types of accounts, the **Institution** and the **AdminAccount**. Those two entities share many common properties. The only property that is not shared is the **surveys** property, as only institutions can create and start surveys. Thus, it might be a good idea to either create a common base entity and extend that with inheritance, or to put the two entities together to create one single **Account** entity.

This problem resulted from thinking ahead too far when creating the entity that stores credentials. It was a quick fix, that wasn't really needed but was still performed. It would probably be a good idea to refactor those two entities before continuing work, to make account management and login easier.

Another weak point is the JWT module with the different tokens that it generates. It



might be better to create a class per token type, to make the handling a bit easier, as right now the JWT module is responsible to create all different token types. These token types include refresh tokens, access tokens, and the so-called event tokens. The event tokens are tokens that are created to send email links to the users, which allow them to reset their passwords, delete their accounts permanently and register new accounts. So with those five different token types, it might be a good idea to separate them, which could be done by separating them into different classes.

This problem is a result of the organic growth of the JWT module and insufficient refactoring. It would surely be worth checking this out before continuing to implement new features.

There are also still some problems with the TypeORM entities is that some relations were not properly defined for lazy loading. The type of the relation should have been a promise of a type, instead of just the type. Because of this, the `ResultRepository`, has to do a complex query to be able to get all results for a certain survey. This change would probably take a bit of a bigger refactoring, but it is certainly something worth doing for the continuation of the project.

The formatting of the emails sent to the users is not too good either. The text could be improved a bit, right now it just contains the most important information and not much more. An HTML version of the emails could be created too, which would make the formatting of the email easier. This has been neglected though, as other things needed to be prioritized.

The application is also still missing some (error) logging capabilities. For development, the errors were logged using `console.log`, but for production, a feature like this is missing. This is due to the limited available time. The logging was not prioritized.

Something that I would probably do differently for a future project is to use another backend technology. While I have enjoyed working with TypeScript, especially compared to regular JavaScript, I felt like the express framework made it a bit difficult to work with dependency injection. I had to pass all the instances through many functions, to be able to keep the business logic decoupled. Compared to Spring Boot for example, which handles dependency injection 'automagically', this was a bit inconvenient.

It might also be worth checking out other frameworks than express, as these might help with this problem as well.

## 11.4 Next Steps

The project team felt the need to meet the client's MVP. The team designed & developed a valid and stable product. All roles except ZEN Administrator and Survey Administrator are fully functional and ready to go into production. At the same time, the user interface and thus the roles of ZEN Administrator and Survey Administrator are not

implemented on the frontend. Furthermore, all tests would have to be implemented later by the frontend. In general, the maintainability of the frontend would have to be ensured by making structural changes.

Unanswered questions arise about the gained accessibility and usability. Is a representative accessibility test as positive as the feedback received? Is the site accessible for screen readers and more exotic input devices?

**Part VI**  
**References**

# List of Figures

1.1	Original Print Version and Result Side by Side . . . . .	3
1.2	Original Print Version and Result Side by Side . . . . .	4
2.1	Domain Model . . . . .	8
3.1	Use Case Diagram . . . . .	10
4.1	Architecture Diagram . . . . .	15
4.2	Context Diagram . . . . .	18
4.3	Container Diagram . . . . .	19
4.4	Component Diagram . . . . .	20
5.1	Database Diagram . . . . .	21
10.1	Extract of the Survey Entry Page . . . . .	43
10.2	Extract of the Survey Page . . . . .	43
10.3	Smileys and Colors can be inverted, in case 'Yes' is the bad Answer. . . . .	43
10.4	If the Question is just about the Opinion, Smileys and Colors can be removed. . . . .	44
10.5	Field for Comments during the Survey. . . . .	44
10.6	Button to display Definitions. . . . .	44
10.7	Definition of a Hard Word. . . . .	45
10.8	Extract of the Results Page for Institution Representatives . . . . .	45
10.9	Registration Input for Institutions. . . . .	46
10.10	Login Screen for Institutions. . . . .	46
10.11	Overview of all the Institution's Surveys. . . . .	46
10.12	First Half of the Institution Management Screen to update Information. . . . .	47
10.13	Second Half of the Institution Management Screen to update Information. . . . .	47
12.1	Enable 2FA . . . . .	66
12.2	Generate App Password . . . . .	67
12.3	Test Output Backend . . . . .	70
12.4	Frontend Integration Tests . . . . .	71

# References

- axe DevTools web accessibility*. URL: <https://chrome.google.com/webstore/detail/axe-axe%20DevTools-web-accessib/lhdoppojpmngadmndnejejpokejbdd>. (accessed: 12.12.2022).
- axe Linter for Visual Studio Code*. URL: <https://www.deque.com/blog/shift-further-left-with-deques-axe-linter-for-vs-code/>. (accessed: 12.12.2022).
- Brown, Simon. *The C4 model for visualising software architecture. Context, Containers, Components, and Code*. URL: <https://c4model.com/>. (accessed: 09.12.2022).
- certbot*. URL: <https://certbot.eff.org/>. (accessed: 14.12.2022).
- Convention on the Rights of Persons with Disabilities (CRPD)*. URL: <https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities.html>. (accessed: 09.12.2022).
- Creating a personal access token*. URL: <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>. (accessed: 10.12.2022).
- Let's Encrypt*. URL: <https://letsencrypt.org/>. (accessed: 14.12.2022).
- Martin, Robert C. *The Clean Architecture*. URL: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. (accessed: 09.12.2022).
- Renzel, Klaus and Wolfgang Keller. "Client/Server Architectures for Business Information Systems A Pattern Language". In: 1997. URL: <https://hillside.net/plop/plop97/Proceedings/renzel.pdf>. (accessed: 09.12.2022).
- Web Content Accessibility Guidelines (WCAG) 2.1*. URL: <https://www.w3.org/TR/WCAG21/>. (accessed: 21.12.2022).

**Part VII**  
**Appendix**

The following list is a brief overview of our agreed cooperation for the project scope.

- **Collaboration method:** Scrum+, which is a combination of Scrum and RUP (Rational Unified Process).
- **Roles:** were not specified due to the team size of two people.
- **Planned meetings:** half-weekly scrums were typically held on Mondays, Wednesdays and Fridays. Sprints were held on demand.
- **Long Term Plan:** see section Project Plan.
- **Short Term Plan:** available in [Jira](#)
- **Risks:** Most of the technology is known to the team members. However, a risk of sickness initially remained such that a buffer of two weeks would be enough time to finish the MVP in week 12. This is also visible in the long-term plan.
- **Time tracking:** available in [Jira](#).

#### 11.4.1 Responsibilities

Carlo Del Rossi was responsible for the backend, CI/CD, test server deployment and general deployment. Claudio Knaus was responsible for the frontend.

# Chapter 12

## Operational Notes

This chapter describes the used tools for frontend and backend, CI/CD, installation instructions and test-logs.

### 12.1 CI/CD

For our project, we have created several GitHub Actions on our repositories to build docker images, test our code etc. In this section, we will show them and explain what they do.

#### 12.1.1 Automated Backend Test Action

This GitHub action runs all tests in the backend repository.

As seen in lines 3 through 6, the tests are run every time a pull request merging into the main branch is created, as well as every time a push-on merge is done.

As seen on line 18, the command 'npm test' is executed. This command is defined in the package.json file of the backend repository and contains the run configuration for the tests:

```
"test": "ts-mocha -p tsconfig.json test/tests/*.ts"
```



```
1 name: Backend Tests
2 on:
3 push:
4   branches: [ "main" ]
5 pull_request:
6   branches: [ "main" ]
7 jobs:
8   build:
9     runs-on: ubuntu-latest
10    steps:
11    - uses: actions/checkout@v3
12    - name: Use Node.js
13      uses: actions/setup-node@v3
14      with:
15        node-version: 14.x
16        cache: 'npm'
17    - run: npm install
18    - run: npm test
```

### 12.1.2 Automated Docker Image Build Actions

This GitHub action builds docker images from the code on the repositories and pushes them to the container registry when done. Overall we have five actions in the same style as this, each one responsible to build one of the components of our system (database, proxy, secure proxy, frontend, backend). Each action contains a line similar to lines 16 and 17 in this action, that builds the docker containers from the Dockerfile. Lines 18 through 22 are almost the same in all the actions. They are responsible for logging in to the GHCR (GitHub Container Registry) and for pushing the build image to the GHCR.

```

1 name: Docker Image CI
2 on:
3   push:
4     branches: [ "main" ]
5 jobs:
6   build:
7     runs-on: ubuntu-latest
8     steps:
9     - uses: actions/checkout@v3
10    - name: Build the Docker image
11      env:
12        DB_HOST: ${ secrets.DB_HOST }
13        DB_USER: ${ secrets.DB_USER }
14        DB_PASSWD: ${ secrets.DB_PASSWD }
15        DB_NAME: ${ secrets.DB_NAME }
16      run: docker build . --file Dockerfile
17          -t ghcr.io/knackpunktbrk/backend:latest
18    - name: login to registry
19      run: echo ${ secrets.CONTAINER_REGISTRY_TOKEN }
20          | docker login ghcr.io -u <username> --password-stdin
21    - name: push to registry
22      run: docker push ghcr.io/knackpunktbrk/backend:latest

```

### 12.1.3 Automated Documentation Building

The last type of action that we created is an action that automatically builds a PDF from our  $\text{\LaTeX}$ -files. This ensures that the documentation can be easily accessed and viewed without requiring a  $\text{\LaTeX}$ -environment. The first step (starting on line 11) compiles the document and creates a PDF. The second step (starting on line 16) uploads the PDF so that it can be downloaded from the GitHub repository.

```
1 name: Build LaTeX
2 on:
3   push:
4     branches: [ "main" ]
5     workflow_dispatch:
6 jobs:
7   build:
8     runs-on: ubuntu-latest
9     steps:
10    - uses: actions/checkout@v3
11    - name: Compile Document
12      uses: xu-cheng/latex-action@v2
13      with:
14        working_directory: ./Documentation/src
15        root_file: main.tex
16    - name: Upload PDF
17      uses: actions/upload-artifact@v3
18      with:
19        name: PDF
20        path: ./Documentation/src/main.pdf
```

## 12.2 Installation instructions

In order to deploy the application, two files are required, namely the docker-compose.yml file and the .env file. Both these files should be in the same folder. The following subsections describe those two files in more detail.

## 12.2.1 Docker Compose File

```
1     version: "3.9"
2
3     services:
4         db:
5             image: ghcr.io/knackpunktbrk/database:latest
6             restart: always
7             ports:
8                 - "${DB_PORT}:${DB_PORT}"
9             expose:
10                - ${DB_PORT}
11            environment:
12                POSTGRES_DB: ${DB_NAME}
13                POSTGRES_USER: ${DB_USER}
14                POSTGRES_PASSWORD: ${DB_PASSWD}
15            volumes:
16                - ./postgres-data:/var/lib/postgresql/data
17
18        backend:
19            image: ghcr.io/knackpunktbrk/backend:latest
20            ports:
21                - "8000:${API_PORT}"
22            restart: always
23            depends_on:
24                - db
25            environment:
26                DB_HOST: db
27                DB_USER: ${DB_USER}
28                DB_PASSWD: ${DB_PASSWD}
29                DB_NAME: ${DB_NAME}
30                JWT_TOKEN_SECRET: ${JWT_TOKEN_SECRET}
31
32        frontend:
33            image: ghcr.io/knackpunktbrk/frontend:latest
34            ports:
35                - "${FE_PORT}:80"
36            restart: always
37            depends_on:
38                - backend
39                - db
40            environment:
41                FE_PORT: ${FE_PORT}
42                API_HOST: ${RP_HOST}
43                API_PORT: ${RP_PORT}
```

```

42     rps:
43         image: ghcr.io/knackpunktbrk/secure-proxy:latest
44         restart: always
45         ports:
46             - "80:80"
47             - "443:443"
48         expose:
49             - "80"
50             - "443"
51         volumes:
52             - ./certbot/www:/var/www/certbot/:ro
53             - ./certbot/conf:/etc/nginx/ssl/:ro
54         depends_on:
55             - backend
56             - frontend
57
58     certbot:
59         image: certbot/certbot:latest
60         volumes:
61             - ./certbot/www:/var/www/certbot/:rw
62             - ./certbot/conf:/etc/letsencrypt/:rw

```

On lines 5, 18, 31 and 43 the required docker images are pulled from the GHCR. To be able to pull the images from the private repositories one has to log in first. Similar to the GitHub actions mentioned earlier, the login happens with the following shell command:

```
echo <password> | docker login ghcr.io -u <username> --password-stdin
```

When the login has succeeded, the application can be started using the following command:

```
docker-compose up
```

This will download the docker images if they are present on the server yet and then start them up. To update the application in case there are newer images on the GHCR, the images can be downloaded with the following command:

```
docker pull ghcr.io/knackpunktbrk/<repository-name>
```

The containers can then be restarted with the following two commands:

```
docker-compose down
docker-compose up
```

To be able to log in a personal access token needs to be created on GitHub first. For more details consider.<sup>1</sup>

---

<sup>1</sup>Creating a personal access token. URL: <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>. (accessed: 10.12.2022).

## certbot

The certbot image is used to easily receive a TLS certificate from Let's Encrypt. The certificate is valid for 90 days. Thus certbot needs to be run manually every once in a while to refresh the certificate, as this task has not been automated yet.

To generate a certificate for the first time, the following command can be used:

```
docker-compose run --rm certbot certonly --webroot
--webroot-path /var/www/certbot/
-d example.org
```

To renew an existing certificate, the following command needs to be run:

```
docker-compose run --rm certbot renew
```

### 12.2.2 Environment File

In addition to the docker-compose.yml file, a file containing the required environment variables is necessary. Usually, the file is named '.env'. The .env file should look like this:

```
1 # API
2 API_PORT=8000
3 API_HOST=<hostname of the website>
4 JWT_TOKEN_SECRET=<secret for the generation of JSON Web Tokens>
5
6 # Database
7 DB_TYPE=postgres
8 DB_HOST=db
9 DB_PORT=5432
10 DB_USER=<username>
11 DB_PASSWD=<password>
12 DB_NAME=<db name>
13
14 # Frontend
15 FE_PORT=8080
16
17 # Reverse Proxy
18 RP_PORT=80
19
20 # Mail Service
21 MAIL_SERVICE_ADDRESS=<gmail address>
22 MAIL_SERVICE_PASSWD=<password>
23
24 # Github
25 CONTAINER_REGISTRY_TOKEN=<personal access token from GitHub>
```

## Registry Token

The `CONTAINER_REGISTRY_TOKEN` is not mandatory, as it is only used to log in to the GHCR, in order to pull the newest images.

## Mail Service Credentials

As of the writing of this documentation, Gmail is used to send emails to users. The `MAIL_SERVICE_PASSWD` is not the standard password used with the Gmail account. Instead, it is a password that can be generated on the google account management website. 2 Factor Authentication (2FA) needs to be enabled in order to be able to generate such a password. Under Security, there is a section called 'Signing in to Google'. In this section, 2FA can be enabled.

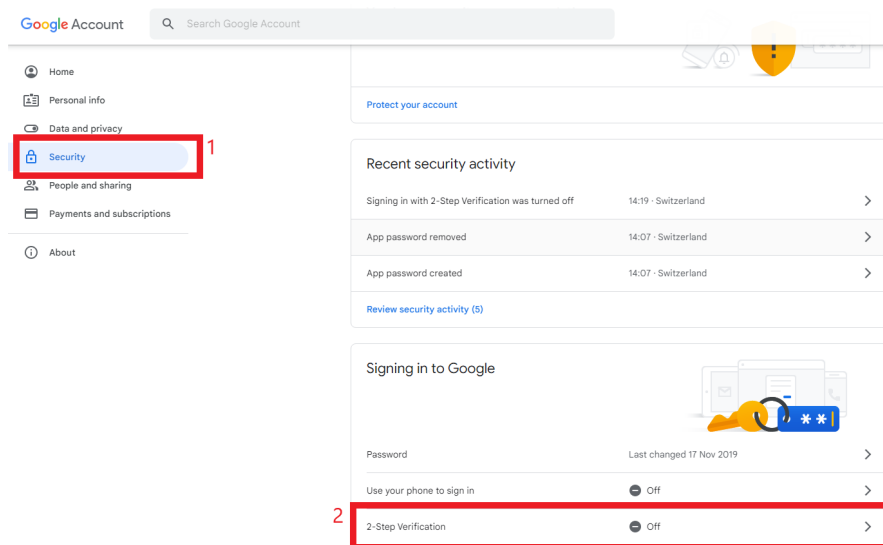


Figure 12.1: Enable 2FA

Once that is completed, 'App passwords' can be generated. In the app password creation dialogue, 'Other' can be selected as app and then a name can be entered. That name does not have a significance other than keeping track of what app passwords are active. Once the password is generated, it can be only looked up once, after that it will not be possible to see the password anymore. The generated password can now be used in the `.env` file to be able to send emails.

## 12.3 Updating the Survey Questions

As the application does not support question creation or deletion, this has to be done manually using the init scripts provided in the database repository.

Before executing any scripts, the existing scripts should be adjusted, to reflect the

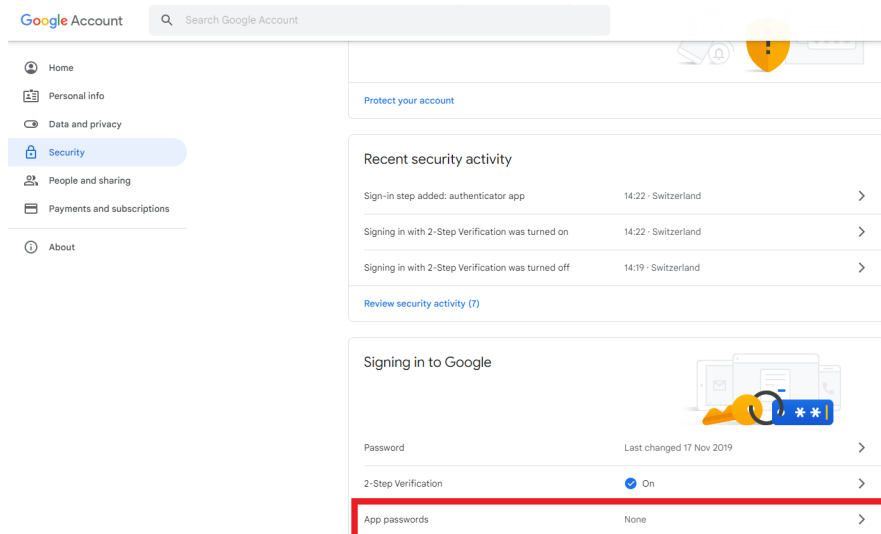


Figure 12.2: Generate App Password

desired changes to the questions. To edit the questions, the files starting with 20 or higher need to be adjusted accordingly. The files are named after the following scheme:

`<number in sequence>_<target group>_<category>.sql`  
 f.ex: `20_fachkraefte_wohnen.sql`

Each of the scripts with a sequence number greater or equal to 20 contains the necessary SQL statements to insert the questions. Those statements are arranged in blocks containing three to four individual statements.

The first statement (starting at line 1) creates the question itself. The question text, category, target group and recommendations need to be specified.

The second statement (starting at line 18) inserts all the possible answer options for the question. For this statement, the point distribution needs to be specified.

The third statement (line 27) adds a relation between the created question and the newest survey template. Nothing needs to be specified in this statement.

The fourth statement (line 33) is optional and inserts a definition in case a hard word is used in the question text. The title, which corresponds to the hard word, and the text, which corresponds to the description of the hard word, need to be specified.

In the example below, the segments of the statement that can be edited, are underlined. It is not recommended to adjust any other segments of the query, as those changes might break the scripts. It should also be noted that the sequence of the statements matters,



and that they should always be written in those blocks of three to four statements in this order.

```
1  INSERT INTO question (
2    "questionId",
3    text,
4    category,
5    "targetGroup", -- Can be 'Bewohnende', 'Angehoerige', or 'Fachkraefte'
6    recommendations,
7  ) VALUES (
8    default,
9    'Hat das Wohnheim Ihnen erklärt, ABC ist?',
10   'ABC',
11   'Bewohnende',
12   ARRAY [
13     'Es ist wichtig, dass sie wissen, was ABC ist.',
14     'ABC ist sehr wichtig für XYZ.'
15   ]
16  ) RETURNING "questionId" into questionid;
17
18  INSERT INTO scored_answer_option (
19    option, score, "questionId"
20  ) VALUES
21    ('Ja', 6, questionid),
22    ('Nein', 0, questionid),
23    ('Manchmal', 3, questionid),          -- 'Manchmal' is optional,
24    ('NichtBeurteilbar', 0, questionid) -- the other options
25    ;                                       -- must be provided
26
27  INSERT INTO survey_template_questions_question (
28    "surveyTemplateTemplateId", "questionQuestionId"
29  ) VALUES(
30    (SELECT MAX("templateId") from survey_template), questionid
31  );
32
33  INSERT INTO definition ( -- This statement can be omitted,
34    title,                -- if no complicated words are used
35    text,
36    "questionId"
37  ) VALUES (
38    'ABC',
39    'ABC bedeutet, dass....',
40    questionid
41  );
```

Once all the questions have been adjusted, a new survey template needs to be created. In order to do this, it suffices to run the 01\_add\_survey\_template.sql script, found in the initScripts folder of the Database repository.

```
INSERT INTO survey_template ("templateId", published)
VALUES (default, true);
```

Once a new survey template is created, all scripts that add questions need to be run. That includes all scripts that start with 20 or higher. (The first one would be 20\_fachkraefte\_wohnen.sql)

Once all those scripts are run, all questions should be updated.

## 12.4 Test-logs

This section shows the test logs and results for the frontend and backend.

### 12.4.1 Backend

As seen in figure 12.3, an average test coverage of around 88% has been reached on all the relevant files in the backend. The lowest coverage is on the AdminAccount class. The reason for this is that the AdminAccount class is not yet used productively, so the tests have not been fully implemented yet. The only missing relevant files are the Controllers, which are not tested yet. However, most of the controllers just call one single business logic function, and all business logic functions have been tested individually already, which is why the controller tests weren't prioritized a lot.

```
158 passing (2s)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	88.03	76.67	74.57	91.62	
src	100	100	100	100	
ApiError.ts	100	100	100	100	
src/authentication	95.52	83.87	100	95.52	
JWT.ts	94.91	81.48	100	94.91	60,67,78
hashing.ts	100	100	100	100	
src/businessLogic	93.2	80.76	94.23	93	
institutionLogic.ts	98.75	90.9	100	98.71	133
loginLogic.ts	84.21	68.18	84.61	84.21	21,33,40-43,50-53,114
resultLogic.ts	100	100	100	100	
surveyLogic.ts	92.98	87.5	94.44	92.45	38,96,158-159
src/classes	86.9	80	83.33	88.6	
CombinedResult.ts	60.71	33.33	55.55	64	64-75,90
SingleResult.ts	100	100	100	100	
Utils.ts	100	100	100	100	
src/entity	83.16	72.02	60.15	89.94	
AdminAccount.ts	50	15.38	6.66	51.28	57-115
Answer.ts	82.14	65	50	91.66	19,52
Comment.ts	92.59	87.5	71.42	100	33
Credential.ts	83.67	66.66	69.23	83.33	34-35,42-43,56,75,83,99
Definition.ts	85.71	50	33.33	100	25-27
Institution.ts	91.07	85	80.95	96.15	43,138
Question.ts	90.47	92.85	75	94.8	41-42,54-55
ScoredAnswerOption.ts	90.47	64.28	71.42	100	22-24
Survey.ts	80	75	46.66	100	18-59
SurveyTemplate.ts	84	100	60	100	
src/enums	100	100	100	100	
AnswerOption.ts	100	100	100	100	
CredentialType.ts	100	100	100	100	
EventType.ts	100	100	100	100	
QuestionType.ts	100	100	100	100	
TargetGroup.ts	100	100	100	100	
TokenType.ts	100	100	100	100	

Figure 12.3: Test Output Backend

## 12.4.2 Frontend

The following sections contain all relevant frontend logs of the project.

### Integration Test Logs

Integration Tests in the frontend can be executed with the following command:

```
"test": "npx ts-mocha -n loader=ts-node/esm  
-p tsconfig.json 'src/__test__/**/*.ts'",
```

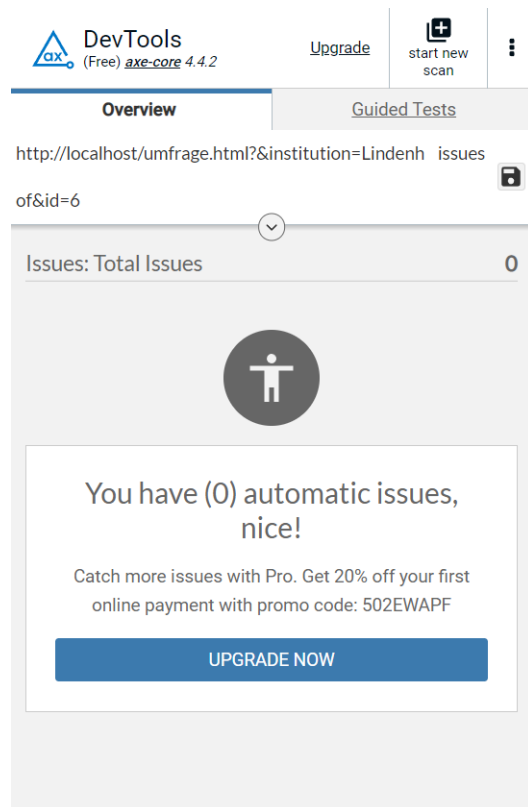
```
GET /api/umfrage/{institutionName}/{surveyIdentifier}  
  ✓ should get the type of wrong answers from the server (253ms)  
GET /api/institutionen/{institutionId}/umfragen/  
  ✓ should get all running and past Surveys of an Institution (74ms)  
PUT /api/institutionen  
  ✓ should return a string after registering an Account.  
POST /api/login  
  ✓ should return a JWT token on successful login (139ms)  
POST /api/institutionen/{institutionId}/umfragen/start  
  ✓ should start a new survey (125ms)  
GET /api/resultate/umfrage/{surveyId}  
  ✓ should fetch survey results (134ms)  
POST /api/umfrage/{institutionName}/{surveyIdentifier}  
  ✓ should establish a connection to the server with status 400 (157ms)  
DELETE /api/institutionen/{institutionId}/umfragen/{surveyId}  
  ✓ should delete a existing survey (99ms)  
PATCH /api/institutionen  
  ✓ Update Institution Information (124ms)  
GET /api/institutionen/{institutionId}  
  ✓ Get Institution Information (73ms)  
PATCH /api/institutionen/passwort  
  ✓ should Update Institution Password (419ms)  
PATCH /api/institutionen/email  
  ✓ should Update Institution Email (104ms)  
POST /api/passwort/vergessen  
  ✓ should connect to server and send email with reset-link to specified email, if email  
is in database  
DELETE /api/institutionen/{institutionId}  
  ✓ should Request Institution Deletion (119ms)  
14 passing (2s)
```

Figure 12.4: Frontend Integration Tests

Manual tests for all other endpoints were not logged.

## axe DevTools Page Scan

The page scan includes automated tests on all components of the survey pages on WCAG 2.1AAA HTML semantics and HTML errors. The test shows no problems on all tested pages and hence has WCAG 2.1AA compliance. Furthermore, the test recommends example solutions for problems, if any problems had occurred. So far, all test results were successful.



## axe DevTools intelligent Guided Test

The guided test on all components of the survey pages includes tests of the keyboard, modal dialog, interactive elements, structure, images and forms. The test shows potential problems as well as WCAG 2.1AAA compliancy. Furthermore, the test recommends example solutions for problems. Please note that these tests are obsolete, as discussed in the Conclusion chapter.

## Issues: Total Issues19

Toggle Highlight

Role: The element's role is missing or incorrect2

Inspect issue

More Info

SHARE ISSUE

- 
- 
- 1 of 2
- Next Issue
- Last Issue

### Issue Description

The element's role is missing or is not appropriate for the element's function.

### Element Location

```
body > div#main-view-19:nth-of-type(14) > main:nth-of-type(1) > section.questions:nth-of-type(1) > p#question19:nth-of-type(1)
```

### Element Source

```
<p tabindex="0" id="question19">
  Hat das Wohnheim Ihnen erklärt, was sexuelle Gewalt ist?
</p>
```

To solve this problem, you need to...

Fix the following:

Add the interactive role "dialog" to the element (using semantic HTML where possible).

• Found:

Manually

• Impact:

critical

- wcag2a
- wcag412b
- Found on:

26.11.2022 at 9:49 PM

Toggle Highlight

Dialog / Alert Dialog: Dialog is missing appropriate role and/or attributes2

Inspect issue

More Info

SHARE ISSUE

- 
- 
- 1 of 2
- Next Issue
- Last Issue

### Issue Description

The element appears and functions like a modal dialog but is missing required ARIA role(s) and/or attribute(s).

## Element Location

`body > div#main-view-19:nth-of-type(14) > main:nth-of-type(1) > section.definition:nth-of-type(2) > a:nth-of-type(1) > section:nth-of-type(1) > p:nth-of-type(1)`

- Found:

Manually

- Impact:

critical

- wcag2a
- wcag412b
- Found on:

26.11.2022 at 10:06 PM

Toggle Highlight

Keyboard focus is not placed on opened modal2

Inspect issue

More Info

SHARE ISSUE

- 
- 
- 1 of 2
- Next Issue
- Last Issue

## Issue Description

When the modal dialog is activated, keyboard focus is not placed on/in it.

## Element Location

`body > div#main-view-19:nth-of-type(14) > main:nth-of-type(1) > section.definition:nth-of-type(2) > a:nth-of-type(1) > section:nth-of-type(1) > p:nth-of-type(1)`

- Found:

Manually

- Impact:

serious

- wcag2a
- wcag243a
- Found on:

26.11.2022 at 10:06 PM

Toggle Highlight

Function cannot be performed by keyboard alone2

Inspect issue

More Info

SHARE ISSUE

- 
-

- 1 of 2
- Next Issue
- Last Issue


## Issue Description

There is no way to perform the function using only the keyboard on the same screen or on a qualifying conforming alternate version.

## Element Location

```
body > div#main-view-10:nth-of-type(2) > main:nth-of-type(1) > section.answers:nth-of-type(2) > form:nth-of-type(1) > fieldset.answer-form:nth-of-type(1) > div#\31 0-Nein.radio-button:nth-of-type(2) > label:nth-of-type(1)
```

## Element Source

```
<label for="10-Nein-answer">Nein </label>
```

- Found:

Manually

- Impact:

critical

- wcag2a
- wcag211a
- Found on:

26.11.2022 at 9:34 PM

### Toggle Highlight

Able to browse outside modal with screen reader2

Inspect issue

More Info

SHARE ISSUE

- 
- 
- 1 of 2
- Next Issue
- Last Issue

## Issue Description

Screen readers can read content outside the modal dialog.

## Element Location

```
body > div#main-view-19:nth-of-type(14) > main:nth-of-type(1) > section.definition:nth-of-type(2) > a:nth-of-type(1) > section:nth-of-type(1) > p:nth-of-type(1)
```

To solve this problem, you need to...

## Fix the following:

Add `aria-hidden="true"` attribute to all wrapping elements within `<body>` that do not contain the modal. You do not need to add `aria-hidden="true"` to `<script>` and `<style>` elements as screen readers do not read content in those tags.

- Found:

Manually



● Impact:

serious

- wcag2a
- wcag132a
- Found on:

26.11.2022 at 10:06 PM

Toggle Highlight

Keyboard focus is not maintained in modal1

Inspect issue

More Info

SHARE ISSUE

- 
- 
- 1 of 1
- 
- 

## Issue Description

Keyboard focus is not maintained within the modal. It is possible to tab out of the modal.

## Element Location

```
body > div#main-view-19:nth-of-type(14) > main:nth-of-type(1) > section.definition:nth-of-type(2) > a:nth-of-type(1) > section:nth-of-type(1) > p:nth-of-type(1)
```

● Found:

Manually

● Impact:

serious

- wcag2a
- wcag243a
- Found on:

26.11.2022 at 10:06 PM

Toggle Highlight

Modal is closed, focus is not returned to trigger1

Inspect issue

More Info

SHARE ISSUE

- 
- 
- 1 of 1
- 
- 

## Issue Description

When the modal dialog or similar element is closed, keyboard focus is not returned to the triggering element.

## Element Location

```
body > div#main-view-19:nth-of-type(14) > main:nth-of-type(1) > section.definition:nth-of-type(2) > a:nth-of-type(1) > section:nth-of-type(1) > p:nth-of-type(1)
```

Found:

Manually

Impact:

serious

- wcag2a
- wcag243a
- Found on:

26.11.2022 at 10:06 PM

Toggle Highlight

All page content should be contained by landmarks1

Inspect issue

More Info

SHARE ISSUE

- 
- 
- 1 of 1
- 
- 

## Issue Description

Ensures all page content is contained by landmarks

## Element Location

```
#intermediate-view-12 > .ask-and-comment > .ask-category
```

## Element Source

```
<section class="ask-category">  
  <p tabindex="0">  
    Wollen Sie noch etwas sagen zum Thema «Wohnen»?  
  </p>  
</section>
```

To solve this problem, you need to...

Fix the following:

Some page content is not contained by landmarks

Found:

Automatically

Impact:

moderate

- cat.keyboard
- best-practice
- Found on:

26.11.2022 at 9:31 PM

Toggle Highlight  
Form elements must have labels1  
Inspect issue  
More Info  
SHARE ISSUE

- 
- 
- 1 of 1
- 
- 

## Issue Description

Ensures every form element has a label

## Element Location

`#comment-category-form12`

## Element Source

```
<input type="text" name="comment-category-form" id="comment-category-form12">
```

To solve this problem, you need to...

Fix at least (1) of the following:

- Form element does not have an implicit (wrapped) `<label>`
- Form element does not have an explicit `<label>`
- `aria-label` attribute does not exist or is empty
- `aria-labelledby` attribute does not exist, references elements that do not exist or references elements that are empty
- Element has no `title` attribute
- Element has no `placeholder` attribute
- Element's default semantics were not overridden with `role="none"` or `role="presentation"`
- Found:

Automatically

- Impact:

critical

- `cat.forms`
- `wcag2a`
- `wcag412`
- `wcag131`
- `section508`
- `section508.22.n`
- `ACT`
- Found on:

26.11.2022 at 9:31 PM

Toggle Highlight  
States/Properties: The element has missing or incorrect states or properties5  
Inspect issue  
More Info  
SHARE ISSUE

- 
-

- 1 of 5
- Next Issue
- Last Issue

## Issue Description

The element has missing or incorrect states or properties that are necessary for screen reader users to interact with or understand the content conveyed by the element.

## Element Location

```
body > div#main-view-19:nth-of-type(14) > main:nth-of-type(1) > section.answers:nth-of-type(3) > form:nth-of-type(1) > fieldset.answer-form:nth-of-type(1) > div#\31 9-Ja.radio-button:nth-of-type(1) > input#\31 9-Ja-answer:nth-of-type(1)
```

## Element Source

```
<input id="19-Ja-answer" for="19-Ja" tabindex="0" type="radio" name="answer" value="19-Ja">
```

To solve this problem, you need to...

Fix the following:

Apply the following state(s) to the element: html "checked" property (or aria-checked="false").

- Found:

Manually

- Impact:

critical

- wcag2a
- wcag412b
- Found on:

26.11.2022 at 9:50 PM

## Very Last Page

## Issue Description

Ensures headings have discernible text

## Element Location

```
#end-view > header > h2
```

## Element Source

```
<h2></h2>
```

To solve this problem, you need to...

Fix at least (1) of the following:

- Element does not have text that is visible to screen readers
- aria-label attribute does not exist or is empty

- aria-labelledby attribute does not exist, references elements that do not exist or references elements that are empty
- Element has no title attribute