

Titel der Diplomarbeit

Tobias Prasser, Gernot Fasching, Max Martin Hierzer

April 15, 2025

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Kurzfassung

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

| | |
|---|-----------|
| Abstract | 1 |
| Kurzfassung | 2 |
| 1 Introduction | 5 |
| 1.1 Short description | 5 |
| 1.2 Description of performed work | 5 |
| 1.3 Methodology of the thesis | 6 |
| 2 Tech Stack | 7 |
| 2.1 PostgreSQL | 7 |
| 2.1.1 Generel | 7 |
| 2.1.2 Scheme of the database | 7 |
| 2.2 node.js | 8 |
| 2.3 Sequelize | 8 |
| 2.4 express | 9 |
| 2.5 React | 9 |
| 2.6 PWA | 9 |
| 3 Features | 10 |
| 3.1 Login | 11 |
| 3.1.1 Frontend | 11 |
| 3.1.2 Backend | 12 |
| 3.1.3 Error Handling and Security | 12 |
| 3.2 Registration | 12 |
| 3.2.1 Function Admin | 13 |
| 3.2.2 Function User | 13 |
| 3.2.3 Email | 14 |
| 3.2.4 Roles | 15 |
| 3.3 Group System | 16 |
| 3.4 Create Polls | 16 |

| | | |
|--------|---------------------------------|----|
| 3.4.1 | Start-/ Endtime | 16 |
| 3.4.2 | Questions | 16 |
| 3.4.3 | Demographic Questions | 16 |
| 3.5 | Edit Polls | 18 |
| 3.6 | Voting | 18 |
| 3.6.1 | Disclosed Voting | 18 |
| 3.6.2 | Anonymous Voting | 18 |
| 3.6.3 | Public Voting | 18 |
| 3.7 | Results | 19 |
| 3.7.1 | CSV-Export | 19 |
| 3.8 | MyPolls | 19 |
| 3.8.1 | Polllink | 19 |
| 3.8.2 | Delete Polls | 19 |
| 3.9 | Accessibility | 21 |
| 3.9.1 | Tooltips | 21 |
| 3.9.2 | Screenreader | 22 |
| 3.10 | Styling | 22 |
| 3.10.1 | Generel styling | 22 |
| 3.10.2 | Responsive design | 22 |
| 3.10.3 | Individual styling | 23 |

| | | |
|----------|----------------|-----------|
| 4 | Summary | 24 |
|----------|----------------|-----------|

Chapter 1

Introduction

Link Gliederung: <https://www.diplomarbeiten-bbs.at/durchfuehrung/gliederung-der-diplomarbeit-und-formale-vorgaben>

1.1 Short description

The topic of this diploma thesis is creating a platform which supports different voting options like single, multiple or weighted choice. Additionally there should be a Login system with different roles to administer and create or delete polls and one where the user can simply vote for the polls he's included in. Furthermore there an option to disclose the results and who voted for which answers. The database should run on a remote server and be accessed by an API.

The reason we chose this topic is because our supervisor is part of the LMP party and they couldn't find an appropriate platform to vote on party intern problems and topics. Hence he approached us and suggested we write our diploma thesis on a voting platform.

1.2 Description of performed work

Our aim is to provide a website where different organizations can create and publish polls for their members. Since our finished work will be open source, everyone who wants to create polls will benefit from our work.

We chose to accept the LMP as our partner, because they brought up that there isn't a platform that supports all the features they need. Moreover can they give us feedback of the real life application so we can adjust the features to a user organization. During the development of our work we had monthly meetings with the LMP to discuss the progress. Because we decided

to develop our software in an agile way the discussions we had with them also helped so we could focus on the more important features first and implement elements of lesser importance later.

1.3 Methodology of the thesis

At first we had to decide on a tech stack. After careful consideration we decided upon a PostgreSQL database, a backend of node.js, sequelize to perform database operations and express to write APIs so we can connect with our frontend. Our frontend is based on React and we also included a PWA. After this decision we began with a simple input and output from front- to backend so ensure we all understood how each part is connected to each other. The next step was implementing the first features. We split the elements in different components so we could work separately and efficiently, e.g the single choice is split in create the poll, display the poll, vote, and show the results. Reasons we chose this tech stack and a thorough description of each function our work has will be in the main part.

Chapter 2

Tech Stack

2.1 PostgreSQL

2.1.1 General

PostgreSQL was chosen as the relational database management system for this project due to its robustness, scalability and strong support for advanced data types and queries. The decision to utilize PostgreSQL was further supported by its user-friendly design, extensive and well-maintained documentation, and its widespread adoption as one of the leading open-source relational databases.

The installation was also simple because of the detailed install guide provided by sites like **w3schools**. Before the installation certain configuration had to be done such as: specify the storage directory, select a password and setting the port the server should listen on. To create the first database the SQL shell of PostgreSQL was used in the beginning. But after that the preferred tool was **pgAdmin 4**, an popular and feature rich Open Source administration and development platform for PostgreSQL, to manipulate data or view schemes of the database.[7]

For this project it was essential to have a stable and robust system in the backend that can handle a bigger amount of data efficiently.

2.1.2 Scheme of the database

The initial database scheme differed significantly from its current structure, primarily due to evolving client requirements. However, it was evident from the outset that the "Polls" and "Users" tables would serve as central com-

ponents of the schema, as the majority of the data entities and relationships would reference or depend on these two core tables. For the Entity Relationship Diagrams tools like **draw.io** and **dbDiagramm** where used to create the visualization directly from SQL. The database consists of 13 tables, with "UserPolls", "UserAnswers", "PollGroups" and "QuestionAnswers" functioning as intermediary tables that facilitate many-to-many relationships between the core entities.

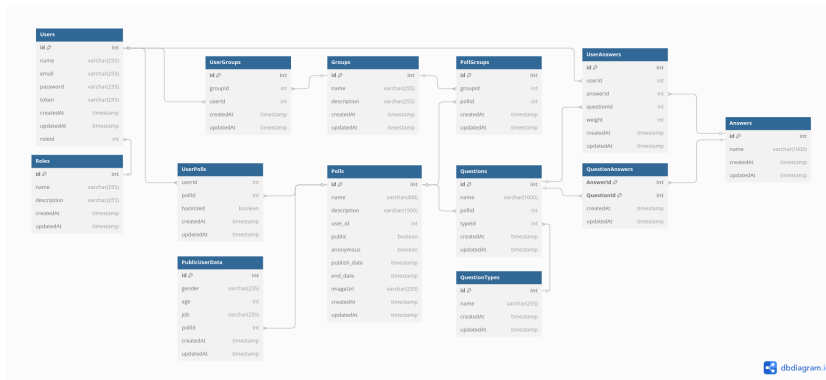


Figure 2.1: New Entity Relationship Diagram

2.2 node.js

2.3 Sequelize

Sequelize was selected as the Object-Relational-Mapping(ORM) tool for this project due to its seamless integration with Node.js and PostgreSQL, offering a robust and declarative approach to database interaction. By abstracting raw SQL queries into model-based definitions, Sequelize enables a more maintainable and readable codebase, especially in project with complex data relationships and evolving schema requirements.

Its support for model associations, transaction management and migrations provides a powerful foundation for implementing scalable and consistent backend logic. Additionally, Sequelize's promised-based architecture aligns well with asynchronous programming paradigms common in modern JavaScript applications, improving performance and responsiveness.[5]

The comprehensive documentation, active community support, and compat-

ibility with various SQL dialects, as in our case PostgreSQL, further contributed to the decision to use Sequelize as the ORM in this project.

As mentioned earlier, the migrations are useful in this case to populate the database to automate testing or at least make setting up the database easier. The Sequelize Command-Line Interface(CLI) was needed to create migrations and seeder. To generate a seeder, the command `npx sequelize-cli seed:generate -name demo-user` is used, which creates a file looking like this: `XXXXXXXXXXXX-demo-user.js`. It follows the **up** / **down** semantics to apply and undo changes. For this project seeders were used to populate the "Roles" and "QuestionTypes" tables for default values, "Users" to have an admin user, "Groups" for a group where admin is a member of and "Polls" for testing purpose.[4]

2.4 express

2.5 React

2.6 PWA

Chapter 3

Features

```
1  useEffect(() => {
2    const linkParam = window.location.search.substring(1);
3    if (linkParam) {
4      const unhashed = atob(decodeURIComponent(linkParam));
5      const params = new URLSearchParams(unhashed);
6      const token = params.get('token');
7      if (token) {
8        setNewUserRegistration(1);
9        setNewUserToken(token);
10     } else {
11       const publicValue = params.get('public');
12       if (publicValue === "true") {
13         setIsPublic(1);
14       } else {
15         setIsPublic(0);
16       }
17     }
18   }
19 }, []);
```

Figure 3.1: Hier ist ein Beispielcode

```

1      useEffect(() => {
2          const linkParam = window.location.search.substring(1);
3          if (linkParam) {
4              const unhashed = atob(decodeURIComponent(linkParam));
5              const params = new URLSearchParams(unhashed);
6              const token = params.get('token');
7              if (token) {
8                  setNewUserRegistration(1);
9                  setNewUserToken(token);
10             } else {
11                 const publicValue = params.get('public');
12                 if (publicValue === "true") {
13                     setIsPublic(1);
14                 } else {
15                     setIsPublic(0);
16                 }
17             }
18         }
19     }, []);

```

Figure 3.2: Hier ist ein Beispielcode

3.1 Login

The login feature is essential for our application, as users need a secure and reliable way to authenticate themselves and gain access to their accounts. It consists of two main functionalities: the login itself and also the logout. Additionally, we have implemented user-friendly error handling to enhance user experience as well as security.

3.1.1 Frontend

The frontend component of the login feature uses React's built-in `useState` hook in order to manage the input username and password and store them in the component's state temporarily. After submitting the login form, the input data is sent to the backend API via a POST request.

The password is sent to the backend in plain text over HTTPS. It is not encrypted on the client side, because HTTPS already provides built-in encryption and also protects against interception or man-in-the-middle attacks.

3.1.2 Backend

The backend receives the login request through a predefined API route that forwards the data to the `handleFetchLogin` function in the user controller. It extracts the username and password from the request body and passes them to the `fetchLogin` function in the user service.

This function first checks if a user with the provided username even exists in the database. If no user is found, it returns an error message stating that either the username or password is invalid. We intentionally chose to not specify which of the two parts is incorrect in order to prevent attackers from being able to determine whether a username exists in the system. This provides protection against user enumeration attacks.

If a user is found, the provided password is then compared to the hashed password stored in the database using bcrypt's comparison method, `bcrypt.compare()`. In case the passwords do not match, the same error message mentioned above is returned to the frontend.

However, if the password is correct, the backend returns a response containing a success indicator, the user's unique ID (`userId`), their username, the assigned role ID (`roleId`), and the name of the role (`roleName`).

3.1.3 Error Handling and Security

Error handling is implemented consistently across both frontend and backend to ensure security and user-friendliness. If a login attempt fails, whether due to an incorrect username, password, or both, the system always returns the same general error message. This prevents attackers from determining whether a specific username exists in the system.

Sensitive data like passwords is handled securely. Passwords are not stored in plain text. Instead they are hashed using bcrypt before being inserted into the database. Furthermore, communication between frontend and backend is encrypted through HTTPS. This ensures that all transmitted data remains private and protected from unauthorized access.

3.2 Registration

The application features a controlled and secure registration. In order for users to be able to join the system, they first have to be invited. The regis-

tration consists of two main functionalities.

3.2.1 Function Admin

Only administrators are allowed to invite new users. They start the process by navigating to the registration tab and entering the user's email as well as their intended role. After submitting, the data gets sent to the "sendEmail" function in the backend through a POST request.

The function checks if a user with the provided email address already exists in the database. If so, it returns a message indicating exactly that. In case the email is not found, an invitation email is sent and a new user entry with the provided email, role ID and a randomly generated token is created. The token is later used for verifying the legitimacy of the registration. If the process is successful the administrator is informed that the email was sent and the user was created successfully.

3.2.2 Function User

The email sent to the user contains a registration link consisting of the URL of the application and the previously mentioned token. By clicking the link, the user is redirected to a registration page on which they can enter a username and password.

The page contains three fields. One for the username, one for the password and one for password validation. This approach was chosen in order to ensure a correct password input. If the contents of the two password fields do not match, an error message telling them about the mismatch is displayed. Similarly, if any of the fields are left blank, submission is disabled and the user is notified of the problem. If all inputs are valid the token extracted from the link, as well as username and password are sent to the backend using a PUT request. The password is sent in plain text over HTTPS, as explained in the login section above.

The backend then passes the data to the "createUser" function. It first checks whether the token exists in the database and also if the username is already being used. In both cases, no token or existing username, the user receives an error message telling them the specific problem. If the username is the issue they can just change it and try again, but if it is the token it will never work no matter how often they retry. If both checks pass, the password is hashed using bcrypt's hash function. The database is then updated with

the new username and hashed password. The token is set to null to prevent reuse. If everything is successful the function returns a success message is returned telling the user that the registration is complete. After five seconds they are redirected to the login page, where they can then log in and use the application.

3.2.3 Email

For automatic sending of the invitation emails, we implemented two different approaches. Therefore both will be explained below.

Nodemailer

The first approach uses "nodemailer" to send emails over SMTP. Firstly a transporter is created using nodemailer's "createTransport" function. There several options can be defined including the host, like gmail or a custom SMTP server, the port, secure option and also authentication credentials. After setting up the transporter, the email options are defined. These include

```
1  let transporter = nodemailer.createTransport({
2    host: "smtp.mailersend.net",
3    port: 587,
4    secure: false,
5    auth: {
6      user: test.email@domain.at,
7      pass: Password1,
8    }
9  });
```

Figure 3.3: This is an example for the transporter using mailersend as the SMTP server

the sender and recipient addresses along with the subject and general content of the email. Finally the email is sent using "transporter.sendMail(mailOptions)".

MailerSend API

The second approach uses the MailerSend API to send the invitation emails. This method was implemented because our deployment host blocks all SMTP ports.

```

1  const mailOptions = {
2  from: "Tool" <test.email@domain.at>,
3  to: user@email.at,
4  subject: "Polling tool",
5  text: "Hello this is your invitation."
6  };

```

Figure 3.4: This is an example for the email options

To use the MailerSend API, a transporter is created with "new MailerSend" and the API key. Just like in the first method mail options are created specifying the sender, recipient, subject and message. The email is then sent

```

1  const mailOptions = new EmailParams()
2  .setFrom(new Sender(test.email@domain.at, "Tool"))
3  .setTo([new Recipient(user@email.at)])
4  .setSubject("Polling tool")
5  .setText("Hello this is your invitation.");

```

Figure 3.5: This is an example for the email options for MailerSend

using "mailerSend.email.send(mailOptions)".

3.2.4 Roles

Implementing a role-based system with three distinct roles - "Admin," "Poweruser," and "Normal" - is crucial for the functionality and security of the application. By assigning permissions flexibly, a clear hierarchy is established, enhancing both user experience and data integrity. Admins are granted full control over the application, while Poweruser enjoy extended privileges for managing polls. Normal users can seamlessly participate in polls and view results without jeopardizing sensitive functionalities. This structure facilitates efficient task delegation and scalability, allowing the application to be easily expanded with additional roles in the future. The role system thus significantly contributes to the security, organization, and user-friendliness of the polling application.

This functionality was implemented by introducing a "Roles" table, which

maintains a one-to-many relationship with the "Users" table. Each user is associated with a specific role through the foreign key **roleId**. The roles are defined and managed exclusively within the database, with the following ID assignments: 1 for administrators, 2 for powerusers and 3 for standard users. Role-based access is determined solely based on these database-level associations.

3.3 Group System

3.4 Create Polls

3.4.1 Start-/ Endtime

3.4.2 Questions

Single Choice

Multiple Choice

Weighted Choice

3.4.3 Demographic Questions

To gather the data of our public voters, the implementation of demographic questions was crucial. This feature is only available for public polls since the created user would be part of the organization using our project and therefore have the data already. If the user is not part of the organization there is still the option to contact them via the e-mail used for the registration. Since most of these questions are similar for every poll, a modular system where questions can be created, added, removed and changed is the best solution. Figure 3.6 shows the demographic question in create polls. The options and functionality of these questions are the like ones described in the previous sections.

The new part for this feature is the search bar. For this the "Select" component of "react-select" is used. The components' controllable state props and modular architecture allows "isMulti" to select multiple options or "isSearchable" to search. These features, allow easy implementation and an already styled search bar in the project. [12]

The database structure for the demographic questions is similar to the standard ones. The table "PublicQuestions" is used to store the question

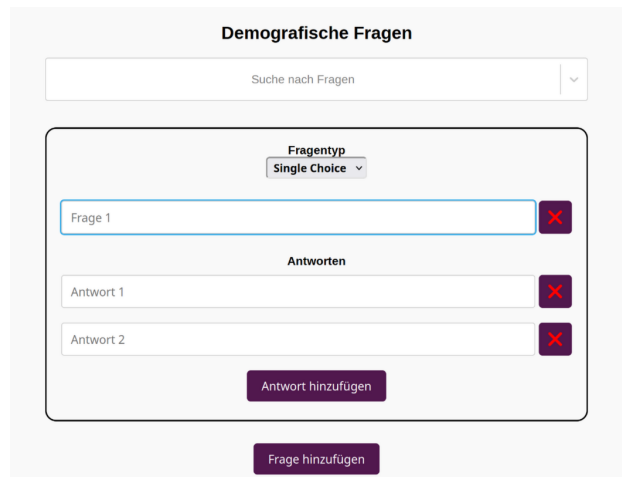


Figure 3.6: Create Demographic Question

specific data like name and type. To enable the reuse of questions on different polls "PublicQuestions" is in an many-to-many relationship with "Polls" through "PublicPollQuestions". Answers are stored within the table "PublicAnswers", which is connected to "PublicQuestions" via "PublicQuestionAnswers". This relation is also many-to-many since the options yes or no for example would be used in multiple questions.

For the select every existing question with its answers is fetched from the database and stored within an array. The options are then mapped with the value being id and the label as name of each element. To display the selected options they are mapped through and use the same functions as the standard ones used for the poll. When saving these questions a problem arises, as the selected ones are already stored in the database and possibly changed. To handle this an `findOrCreate` 3.7 is used to get the existing questions and answers or create new ones. This function also returns each instance found or the created one. With this the question and answer ids can be used for the many-to-many relationship. [6]

```

1  let [createdQuestion, created] = await
    ↪ PublicQuestions.findOrCreate({
2    where: {
3      name: question.name,
4      typeId: questionTypeId,
5    }
6  });

```

Figure 3.7: findOrCreate PublicQuestions

3.5 Edit Polls

3.6 Voting

3.6.1 Disclosed Voting

3.6.2 Anonymous Voting

3.6.3 Public Voting

The public voting allows users without an account to vote. With this feature a wide range of people can be questioned via street surveys or through a shared link. This poll type has two sections, the normal and demographic questions. The order of these plays a major role. Römermann [9] mentions trust, benefits of the demographic data and the ability to abstain. All of these factors have to be taken into one's account when creating these questions. The article also mentions, that at the beginning of a survey the motivation is high and the demographic data are answered, but the trust in full anonymity is decreased. Therefore the author states it is best to put these questions at the end.

Vote integrity

A major problem when having anonymity and no accounts is the data integrity. Without the ability to store information about a voter in the database to check for multiple votes, it is important to prevent them from voting multiple times.

what to write about: how to prevent multiple votes: - captcha against bots - cookies to prevent non techie users - users with knowledge almost

impossible to prevent without storing ip or device fingerprints, problem with ip and device fp is probably legal reasons

- userData: which information is important for polls (gender, age, job,)

3.7 Results

3.7.1 CSV-Export

To allow for the further analysis and documentation of the poll results, a CSV export functionality was implemented. This export provides a structured overview of all user responses, including additional metrics such as the number of votes per answer and the average weight for weighted questions.

The logic is handled in the **csvExportController.js** controller. Upon receiving a request, the system retrieves all relevant data for a given poll by its ID, including associated question, answers, and question types. The raw voting data is processed to count the number of responses per answer and to calculate the average weight if applicable. Special care is taken to distinguish between different types: for instance, "Single Choice" questions ensure that only one answer per user is counted, while "Weighted Choice" questions include the weight value in the computation.

The processed data is then converted into a structured CSV format using the `json2csv` library. The final file includes columns such as Poll Name, Question, Question Type, Answer, Vote Count and Average Weight, and can be downloaded directly by the user. This feature ensures transparency and enables further statistical evaluation using external tools like Excel.[1]

3.8 MyPolls

3.8.1 Polllink

To make sharing of the Surveys easier, a QR code generation was implemented. This was done using the React component "qrcode.react" which requires the link to the poll.[2]

3.8.2 Delete Polls

The deletion of polls is crucial for two main reasons: first, to ensure that all associated data of a survey can be safely removed when necessary and

second, to determine when a poll is still eligible for deletion. In this case, the deletion of polls is strictly limited to those that have not yet received any user votes.

This constraint ensures the integrity of the data and avoids loss of user-generated information, which could distort statistical analysis or transparency within the system.

To maintain data consistency, the deletion process is implemented as a transaction. This guarantees atomicity, meaning that either all steps of the deletion process succeed or none do - preventing partial data deletion and potential corruption. Sequelize's transaction management is used here to wrap the entire process in a rollback-safe structure. [6]

The deletion logic performs the following steps

1. Validate that the poll exists.
2. Fetch all related questions and their IDs.
3. Check if any user answers exist for these questions. If so abort the operation with an error.
4. Remove the many-to-many relations between answers from the "QuestionAnswers" table.
5. Delete the associated answers.
6. Delete the questions.
7. Delete any group relations in the "PollGroups" table.
8. Finally, delete the poll itself.

Using Sequelize's transaction mechanism not only helps to avoid data inconsistency, but also ensures that no information is accidentally deleted once users have participated in a poll. This feature is particularly important in environments where transparency and trust are key, such as in political or organizational voting systems.

For additional security to prevent unauthorized polls from being deleted, the delete button has been grayed out if the poll has already been voted on, with an tooltip added to the button the user gets further information why they can not delete the survey anymore.

```

1  const deletePoll = async (pollId) => {
2    const transaction = await sequelize.transaction();
3    try {
4      ...
5      await PollGroups.destroy({
6        where: { pollId },
7        transaction,
8      });
9      await Polls.destroy({
10       where: { id: pollId },
11       transaction,
12     });
13     await transaction.commit();
14     return { pollId, questionsDeleted: questionIds.length };
15   } catch (error) {
16     await transaction.rollback();
17     throw error;
18   }

```

Figure 3.8: Example for a Sequelize transaction

3.9 Accessibility

3.9.1 Tooltips

To enhance the usability and user experience of the application, tooltips were integrated into all input field, providing users with contextual guidance and reducing the likelihood of input errors. For this purpose, the React library **react-tooltips** was utilized, offering a high degree of customizability in both design and behavior. This allowed for the implementation of consistent and informative tooltip elements throughout the interface.[3]

Each form element is assigned a unique **data-tool-id**, which serves as a reference key within the central **Tooltips.js** configuration file. This file defines the content, styling, and interaction logic for each tooltip, ensuring that the visual presentation and timing behavior are consistent across the application. To prevent the tooltips from becoming distracting during regular use, a delay of 1500 milliseconds was introduced before displaying the tooltip on hover.

The overall purpose of this component was, to improve the intuitiveness of the application, especially for first-time users or those unfamiliar with specific input requirements.

3.9.2 Screenreader

3.10 Styling

3.10.1 General styling

The styling of the application was done later in the making process, because only then it was demanded by the customers. The same applies to the color scheme which has two main colors: Yellow and Blackberry.

To ensure consistency of the palette user defined characteristics from CSS where used to assign variables for each main color. [11]

```
1  :root {  
2    --primary-color: #51184e;  
3    --secondary-color: #F9BB03;  
4    --primary-hover-color: rgb(163, 131, 168);  
5  }
```

Figure 3.9: Variables of the main colors

Those variables were then used in each element of where it was needed. Therefore changing the color scheme to a different one is less afford for the end user.

The basic structure of the page is split into three sections: header with navigation, main with content and footer with imprint and privacy policy. The style is based on the client's homepage therefore the standard background is purple but of the content it still is white to ensure readability. [10]

3.10.2 Responsive design

Responsive design is essential nowadays because users access websites from a wide variety of devices - phones, tablets, laptops and desktops - each with different screen sizes and resolutions. It ensures a seamless, user-friendly experience across all platforms, which improve engagement and accessibility.

To ensure responsiveness in each section of the application, "@media CSS at-rule" was used to apply different styles based on the screen size or device characteristics. This approach allows the layout, font sizes and element spacing to adapt dynamically, creating a smoother experience for users on any device. The navigation bar was designed with media queries to transform into so-called "burger menu" when the screen width falls below a certain threshold. This ensures usability on smaller devices like smartphones because the elements within the navigation get displayed in the menu now.



Figure 3.10: Navigation



Figure 3.11: Burger menu

3.10.3 Individual styling

Chapter 4

Summary

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Hier ein Zitat aus einer Quelle [8].

Bibliography

- [1] Node.js Contributors. *json2csv*. URL: <https://github.com/juanjoDiaz/json2csv> (visited on 04/13/2025).
- [2] React Contributors. *qrcode.react*. URL: <https://www.npmjs.com/package/qrcode.react> (visited on 04/13/2025).
- [3] React Contributors. *react-tooltip*. URL: <https://github.com/ReactTooltip/react-tooltip?tab=readme-ov-file> (visited on 04/15/2025).
- [4] Sequelize Contributors. *Migrations*. URL: <https://sequelize.org/docs/v6/other-topics/migrations/> (visited on 04/15/2025).
- [5] Sequelize Contributors. *Sequelize*. URL: <https://github.com/sequelize/sequelize> (visited on 04/15/2025).
- [6] Sequelize Contributors. *Sequelize Documentation*. URL: <https://sequelize.org/docs/v6/core-concepts/model-querying-finders/> (visited on 04/09/2025).
- [7] W3Schools Contributors. *PostgreSQL - Install Introduction*. URL: https://www.w3schools.com/postgresql/postgresql_install.php (visited on 04/15/2025).
- [8] Autor Name. *Beispieltitel des Buches*. Verlag, 2023.
- [9] Carina Römermann. *Was ist bei soziodemografischen Angaben in Befragungen zu beachten?* URL: <https://www.rogator.de/soziodemografischen-angaben-mitarbeiterbefragungen/> (visited on 04/09/2025).
- [10] tbd. *Liste Madeleine Petrovic*. URL: <https://liste-petrovic.at/> (visited on 04/13/2025).
- [11] tbd. *Verwendung von CSS-Benutzerdefinierten Eigenschaften (Variablen)*. URL: https://developer.mozilla.org/de/docs/Web/CSS/CSS_cascading_variables/Using_CSS_custom_properties (visited on 04/12/2025).
- [12] Jed Watson and contributors. *React-Select*. URL: <https://www.npmjs.com/package/react-select> (visited on 04/09/2025).

List of Figures

| | | |
|------|---|----|
| 1 | Create Demographic Question | 1 |
| 2.1 | New Entity Relationship Diagram | 8 |
| 3.1 | Hier ist ein Beispielcode | 10 |
| 3.2 | Hier ist ein Beispielcode | 11 |
| 3.3 | This is an example for the transporter using mailersend as the SMTP server | 14 |
| 3.4 | This is an example for the email options | 15 |
| 3.5 | This is an example for the email options for MailerSend | 15 |
| 3.6 | Create Demographic Question | 17 |
| 3.7 | findOrCreate PublicQuestions | 18 |
| 3.8 | Example for a Sequelize transaction | 21 |
| 3.9 | Variables of the main colors | 22 |
| 3.10 | Navigation | 23 |
| 3.11 | Burger menu | 23 |