Technical Documentation

Case Study Semester 4

Group: SurvHey

1 High-Level Diagram

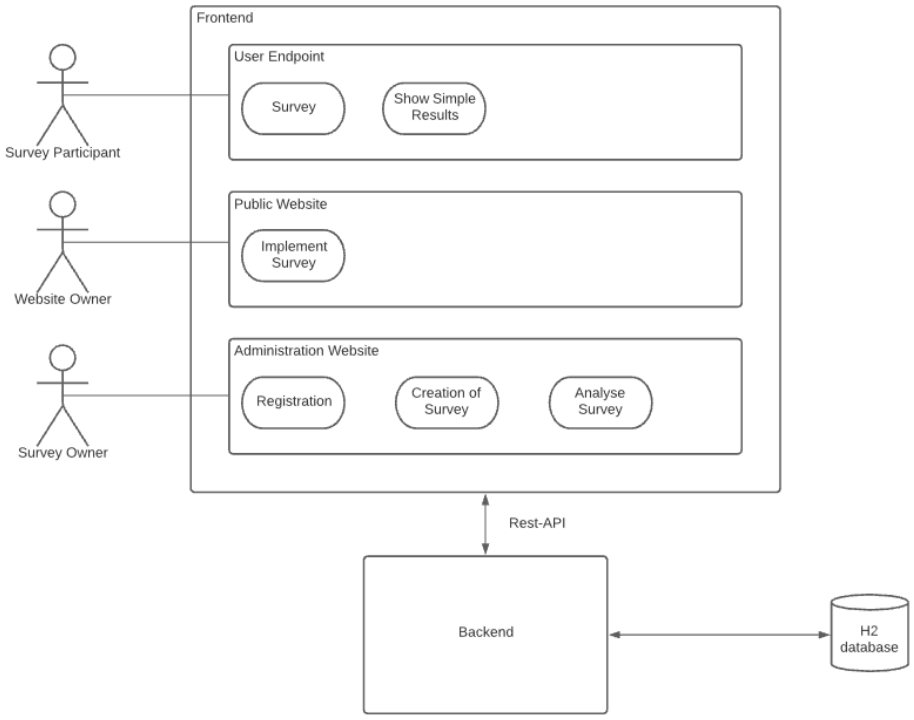

Figure 1: High-Level Diagram

From a high-level perspective, the system involves use cases for three different personas. **Survey participants** aim to fill out the survey on a website. After having answered, they can view a simplified version of the results. **Website owners** run the website the survey is implemented in. The **survey owner** uses the administration website to register herself or himself. After a successful registration he or she is able to create surveys that can be embedded in various websites and analyse the survey's results. The website owner and survey owner can either be the same person or can be different persons.

Those three personas interact with the **front end** of the application. The data used is provided via a REST API to the **back end**. It is stored in a **H2 database**.

2 System Architecture

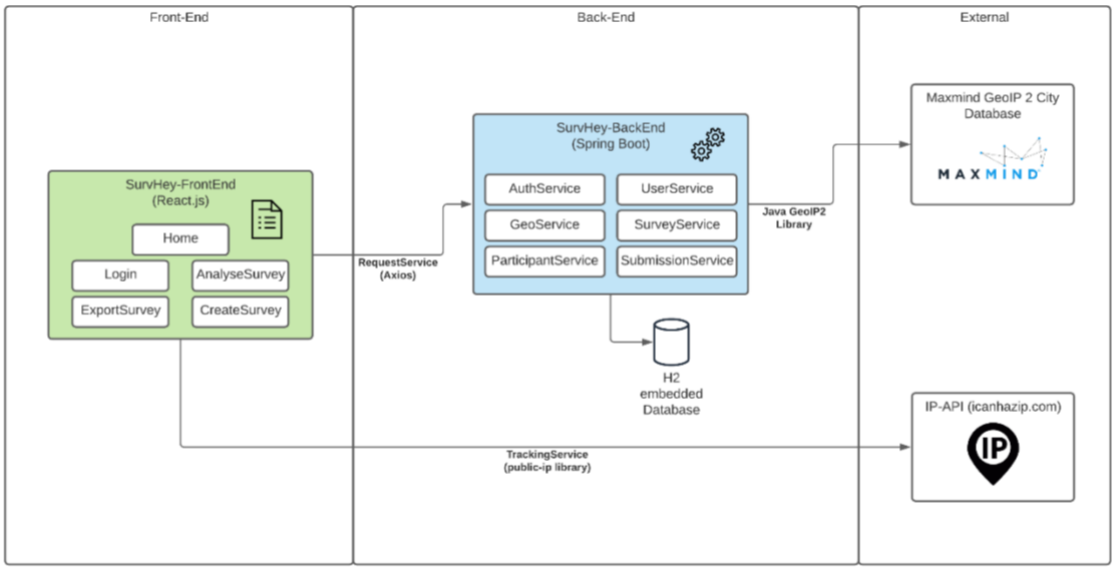

Figure 2: System Architecture

3 Back-End

3.1 ER-Diagram

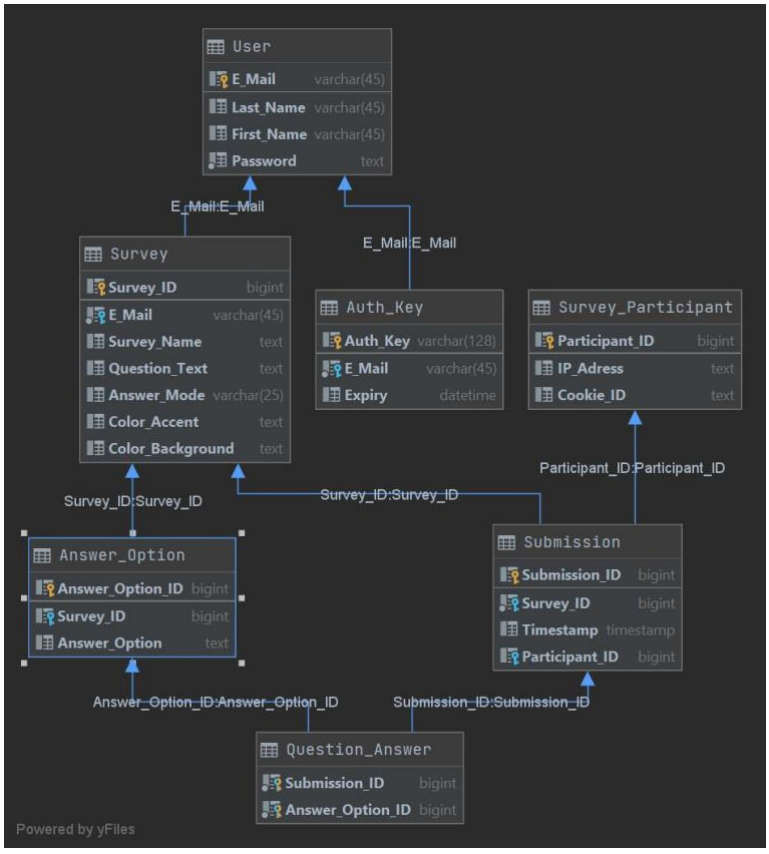

Figure 3: ER-Diagram

3.2 Back-End Sequence Diagrams

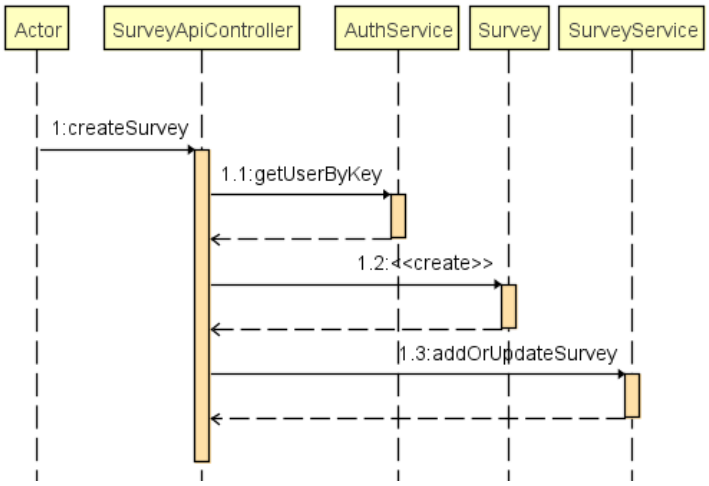

Figure 4: Sequence Diagram: createSurvey

This method creates a new survey. The SurveyApiController fetches the logged-in user through the supplied authentication key. The controller then inserts a new survey with the fetched user and returns a success.

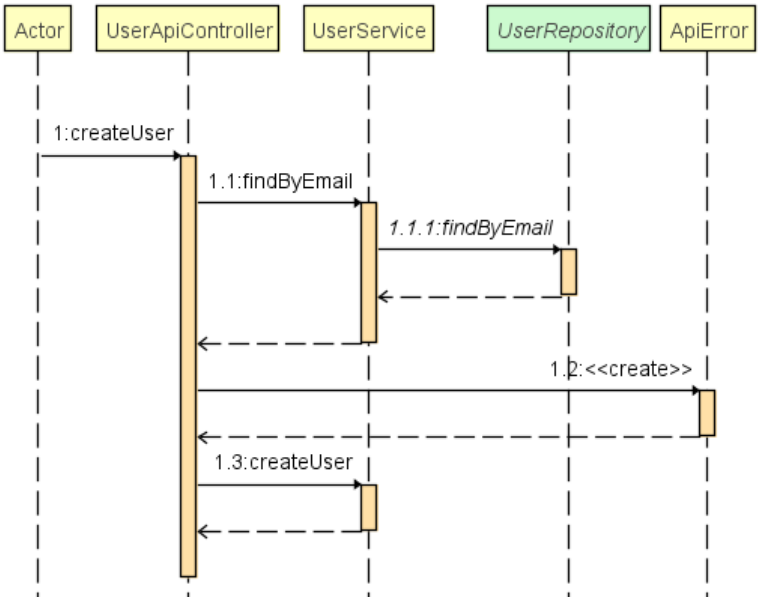

Figure 5: Sequence Diagram: createUser

This function inserts a new user into the database. It checks beforehand if the user with the given email already exists and throws a forbidden error if necessary.

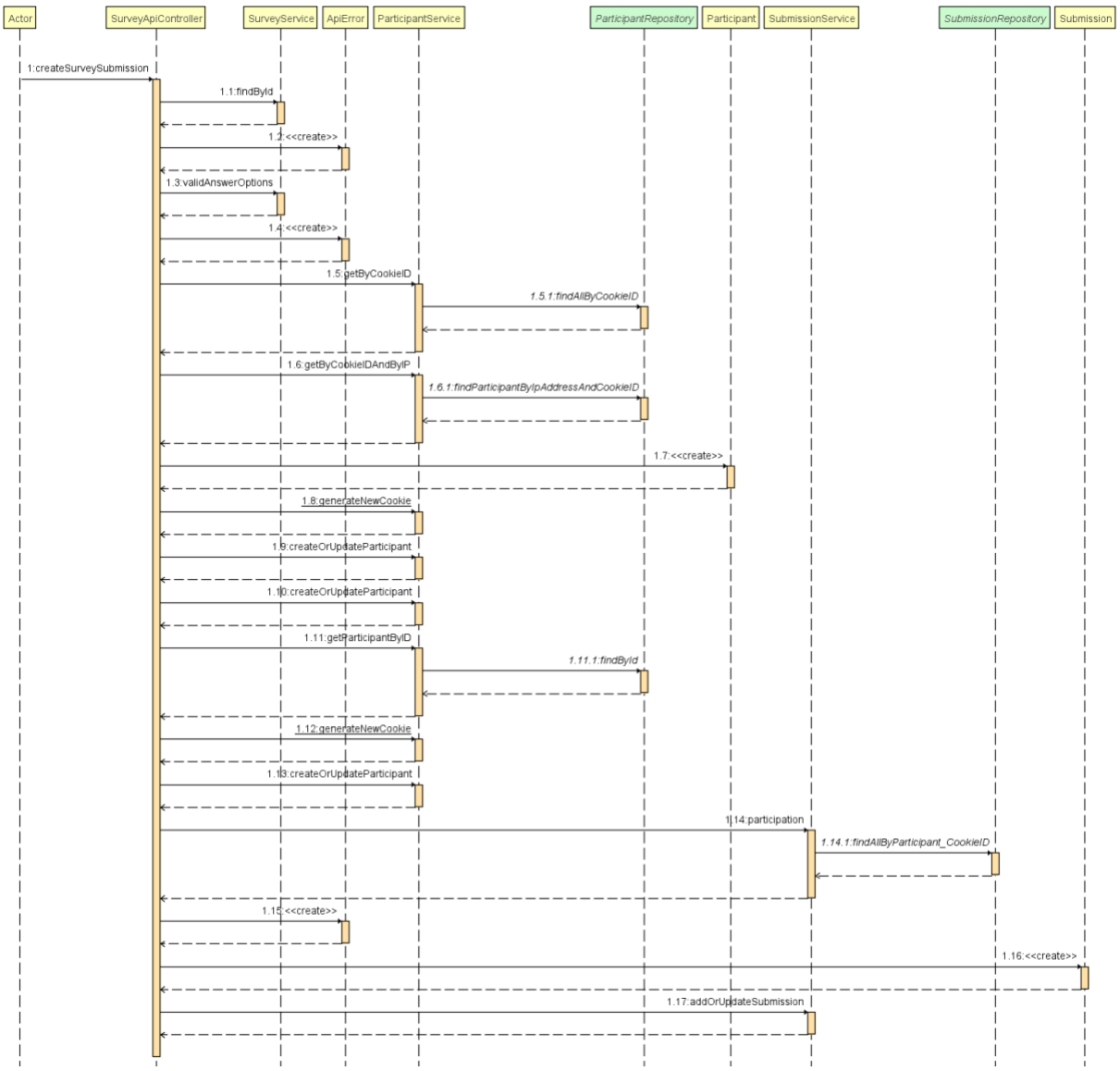

Figure 6: Sequence Diagram: createSurveySubmission

This sequence diagram models the function that creates a new survey submission. 1.1 to 1.3 checks if the given survey exists and if the chosen answerOptions exist. 1.5 to 1.6 check if the participant already exists within the database. The participant is identified through his unique cookieID but can have varying ip-addresses. A new cookieID is generated if the participant is not yet registered in the database (1.8). 1.9 to 1.14 differentiate between participants with this cookieID and new ip-addresses (details on the cases in code-comments). The function inserts the submission with the correct reference to the existing or newly inserted participant.

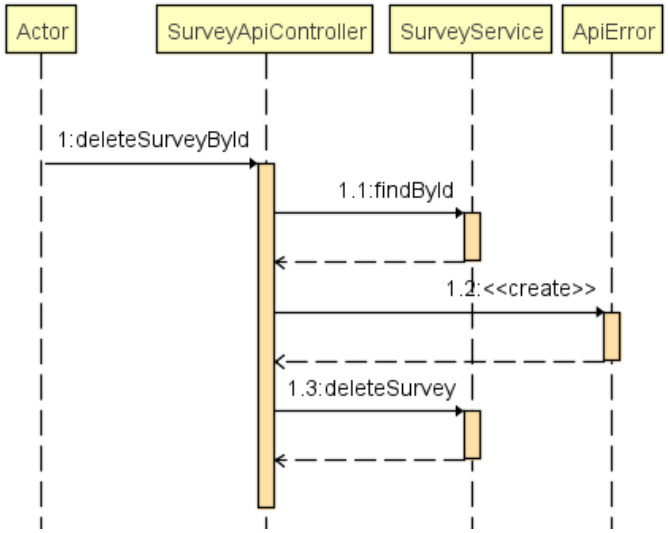

Figure 7: Sequence Diagram: deleteSurveyByID

This function fetches the requested survey by its id and deletes it.

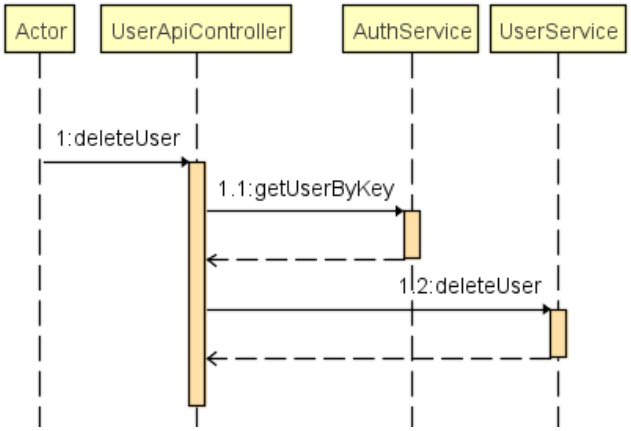

Figure 8: Sequence Diagram: deleteUser

This function fetches the user object through the authentication key of the logged-in user. The user is then deleted.

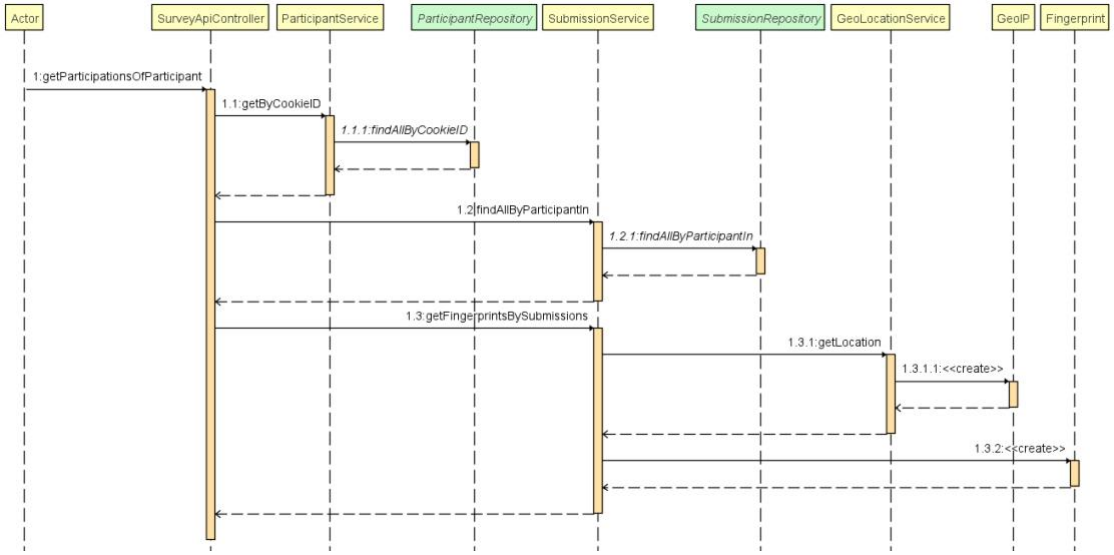

Figure 9: Sequence Diagram: getParticipationOfParticipants

This function returns a list of fingerprints that include the submission and the city where the submission was made for a given participant. All participant objects are retrieved from the database and the ip-addresses are translated into city names through the GeoLocationService (1.3.1). The list of fingerprints is then returned.

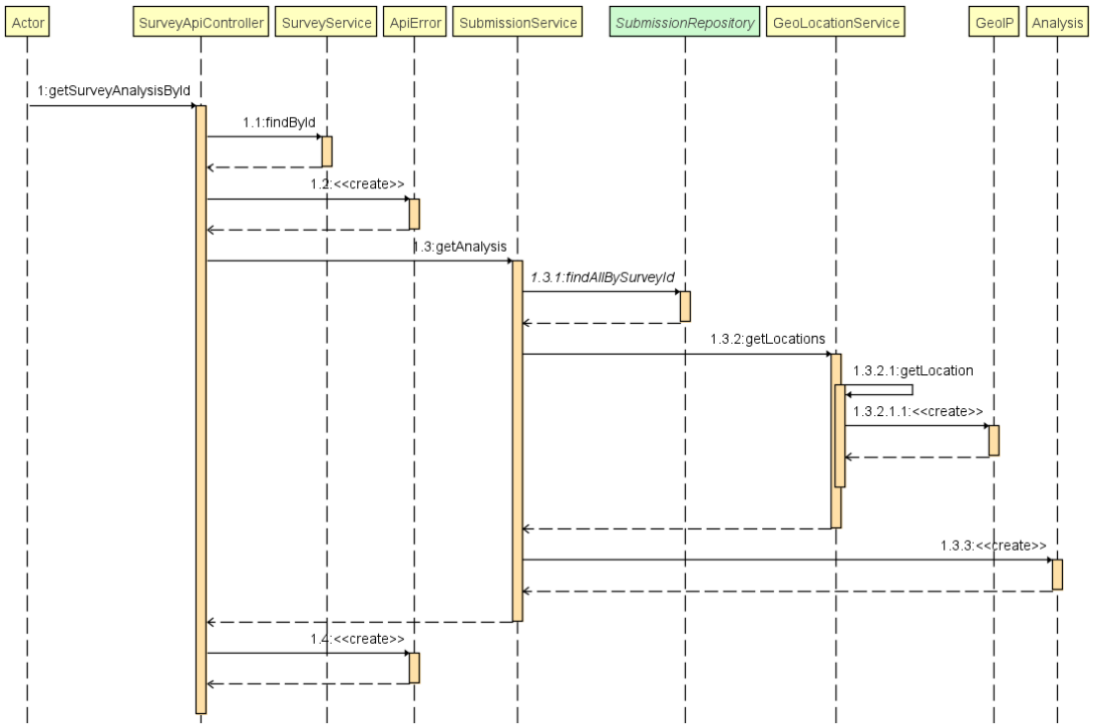

Figure 10: Sequence Diagram: getSurveyAnalysisById

This function returns an analysis object for the given survey. The getAnalysis method finds all submissions for the given survey and translates them into a votes-per-country relation.

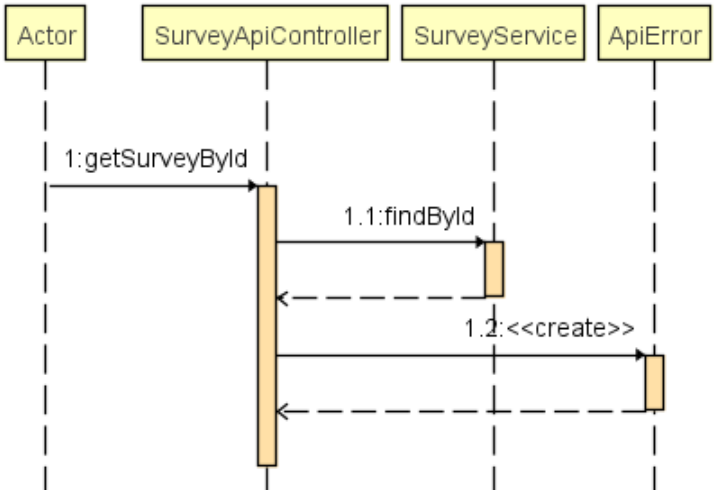

Figure 11: Sequence Diagram: getSurveyById

This function returns the survey object through the given surveyID.

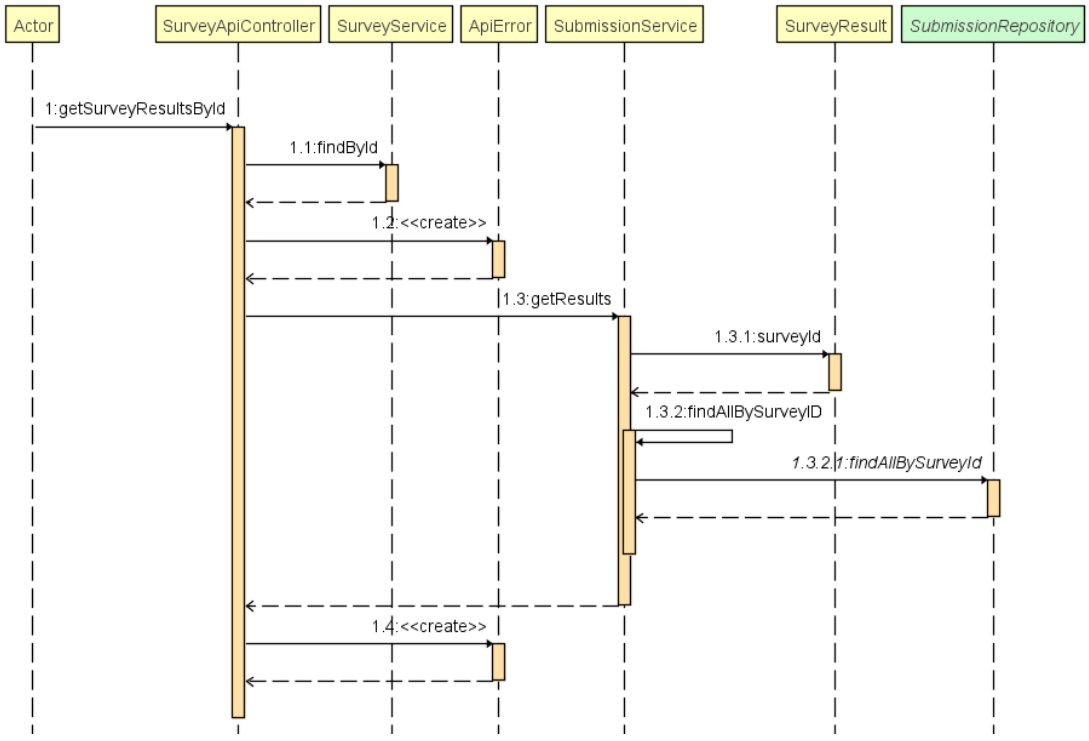

Figure 12: Sequence Diagram: getSurveyResultsByID

This function returns the results of a survey. The provided survey is validated and its results are summarized to the possible answers. These are then returned.

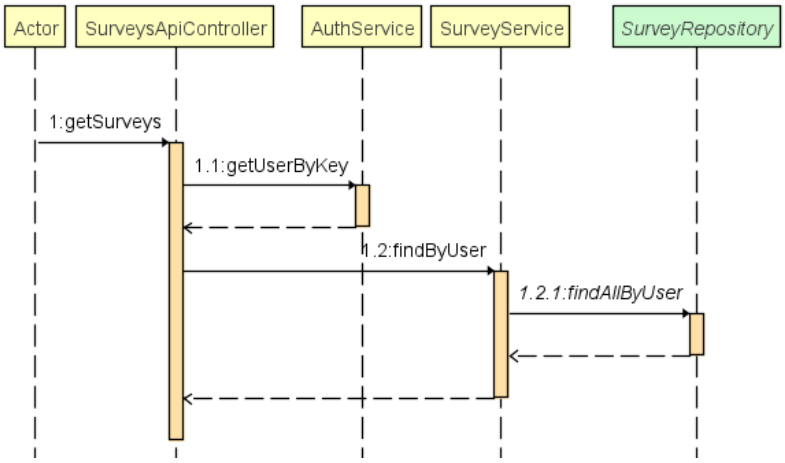

Figure 13: Sequence Diagram: getSurveys

With this function all surveys assigned to a user are retrieved and returned. The user object is loaded with the provided authentication key and the associated surveys are loaded based on the user object. They are then returned.

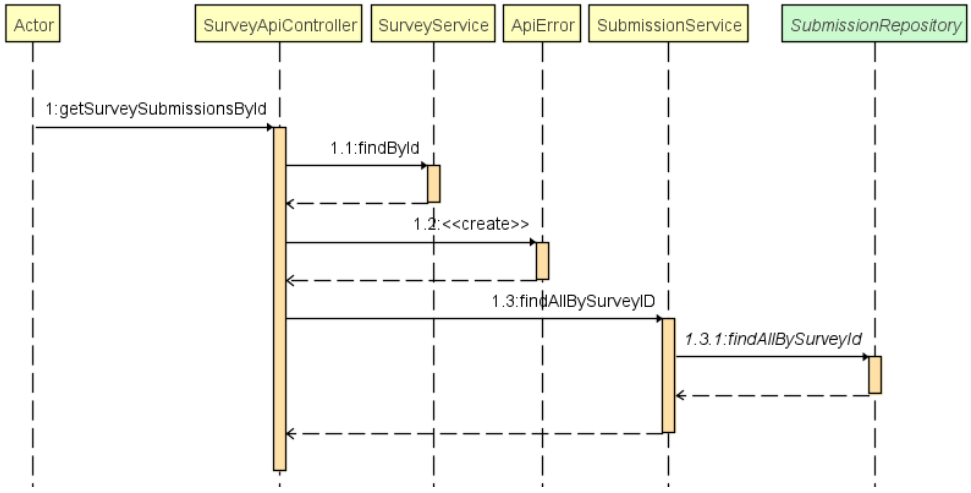

Figure 14: Sequence Diagram: getSurveySubmissionsByID

All submissions that are assigned to a survey are returned in this function. The survey object is fetched from the database using the survey ID. Then all submissions of the given survey are retrieved and added to a list, which is then returned.

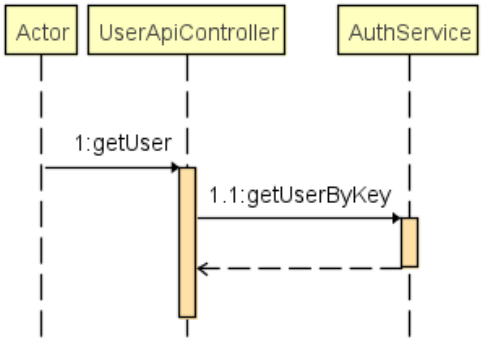

Figure 15: Sequence Diagram: getUser

This function returns a user object. With the help of the authentication key the corresponding user is generated from the database. This is then returned.

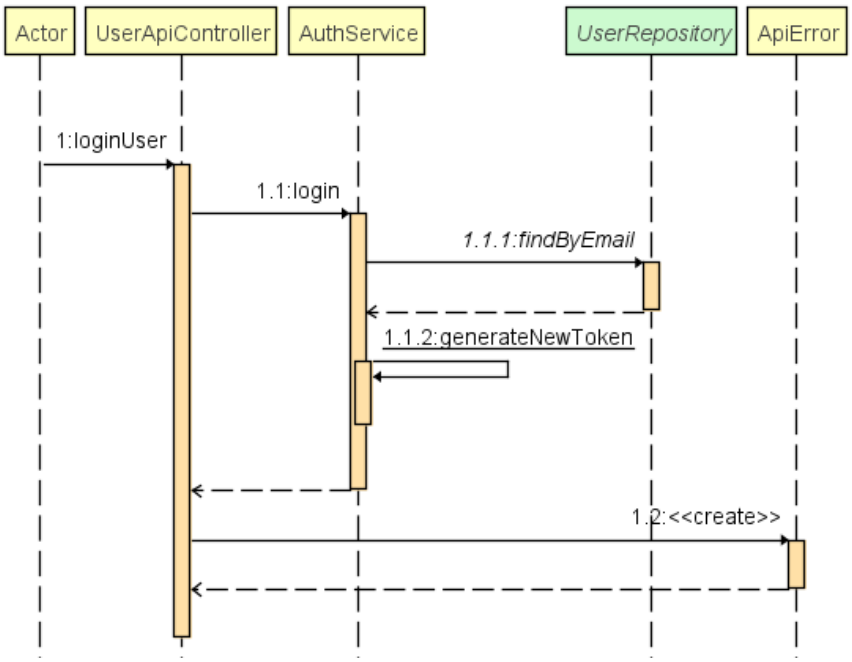

Figure 16: Sequence Diagram: loginUser

With this function a user is logged into the system. The matching of the login credentials is checked, and the corresponding user object is loaded. If the login credentials match, an authentication token is generated and send to the user.

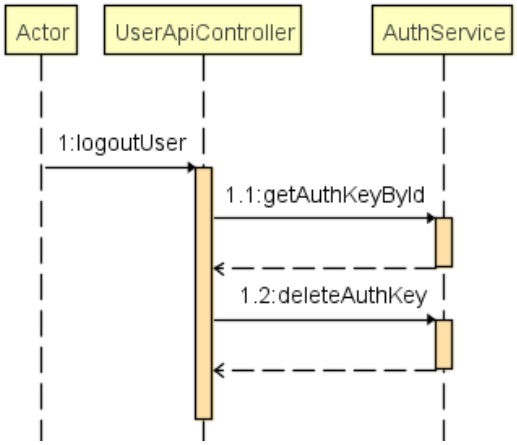

Figure 17: Sequence Diagram: logoutUser

This function is used to log out a user from the system. The supplied authentication key is loaded and deleted from the database.

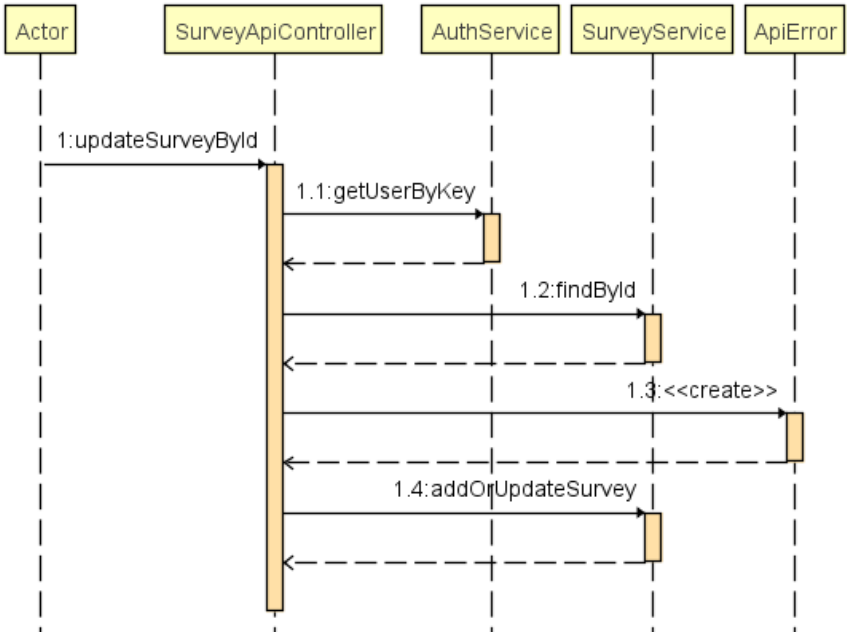

Figure 18: Sequence Diagram: updateSurveyById

With this function the values of an already existing survey can be edited and updated. The user object is fetched through the authentication key. The survey object is updated with the Survey-ID through the function addOrUpdateSurvey.

4 Front-End

4.1 Flow Charts

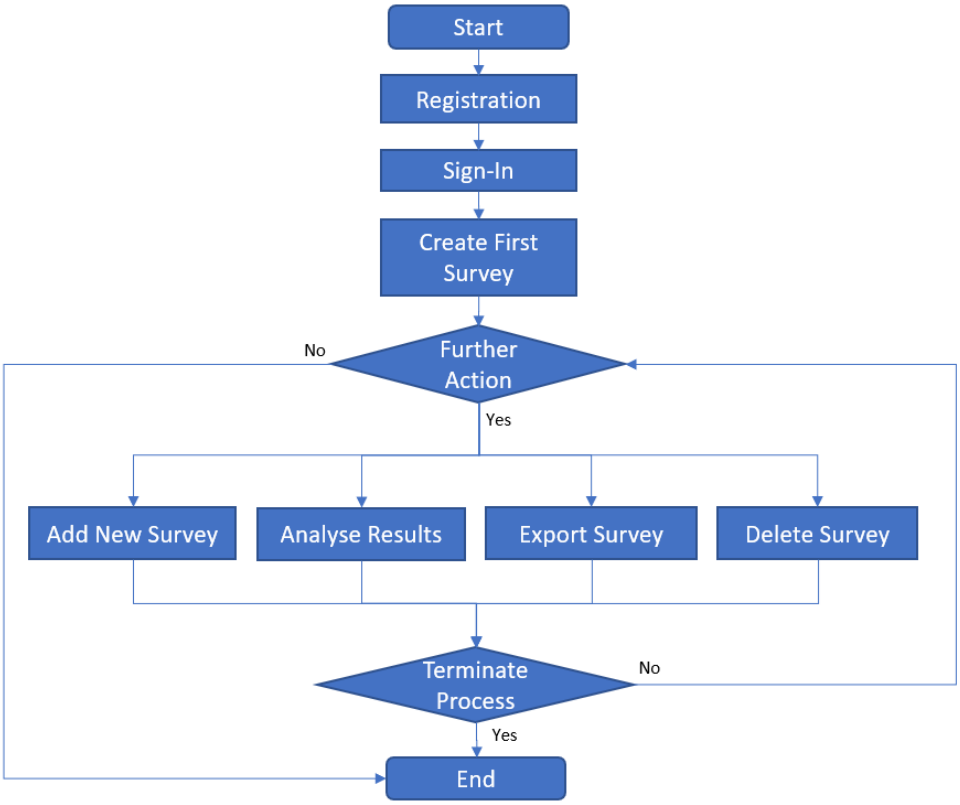

Figure 19: Flow Chart: Website Owner Process

From a website owner's perspective, the process starts with the registration. After having registered, the user signs in with his login credentials. As a next, step the system leads through the survey creation process. With the finished survey, the user interface offers to add another survey, analyse the answers given by the participants, export the survey results, or delete them.

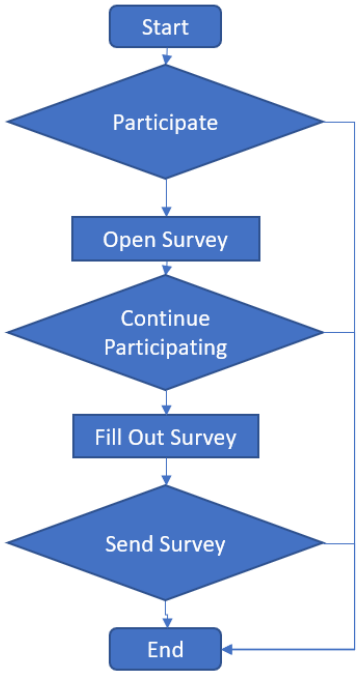

Figure 20: Flow Chart: End User Process

From a survey participant perspective, the process starts with opening the website on which the survey is implemented. If the participant decides to fill in the data, they can send it to transmit their results to the administrator.

Figure 21: Flow Chart: Website Owner Process

The website owner chooses the survey to implement on their website by copying the link or code from the iFrame. The survey is ready to be answered by website visitors.

4.2 UML Diagrams

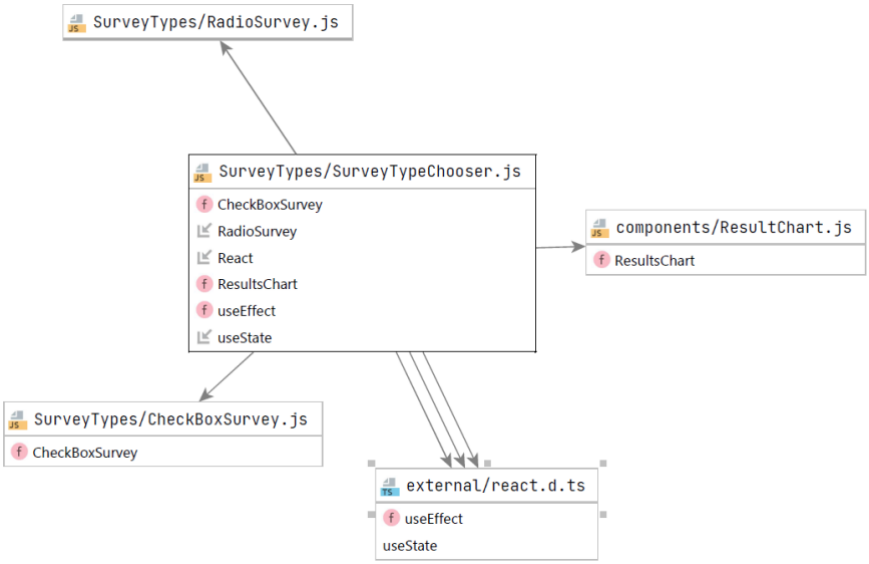

Figure 22: UML Diagram: SurveyTypeChooser

Before displaying, the surveyTypeChooser decides which surveyType has to be displayed. It gets passed the surveyType as a prop and then checks via if-statements, which surveyTypes the survey has and renders the according survey component. If the surveyTypes do not match, an error message is displayed.

SurveyTypeChooser
+ surveyType + showResults
+ useEffect()

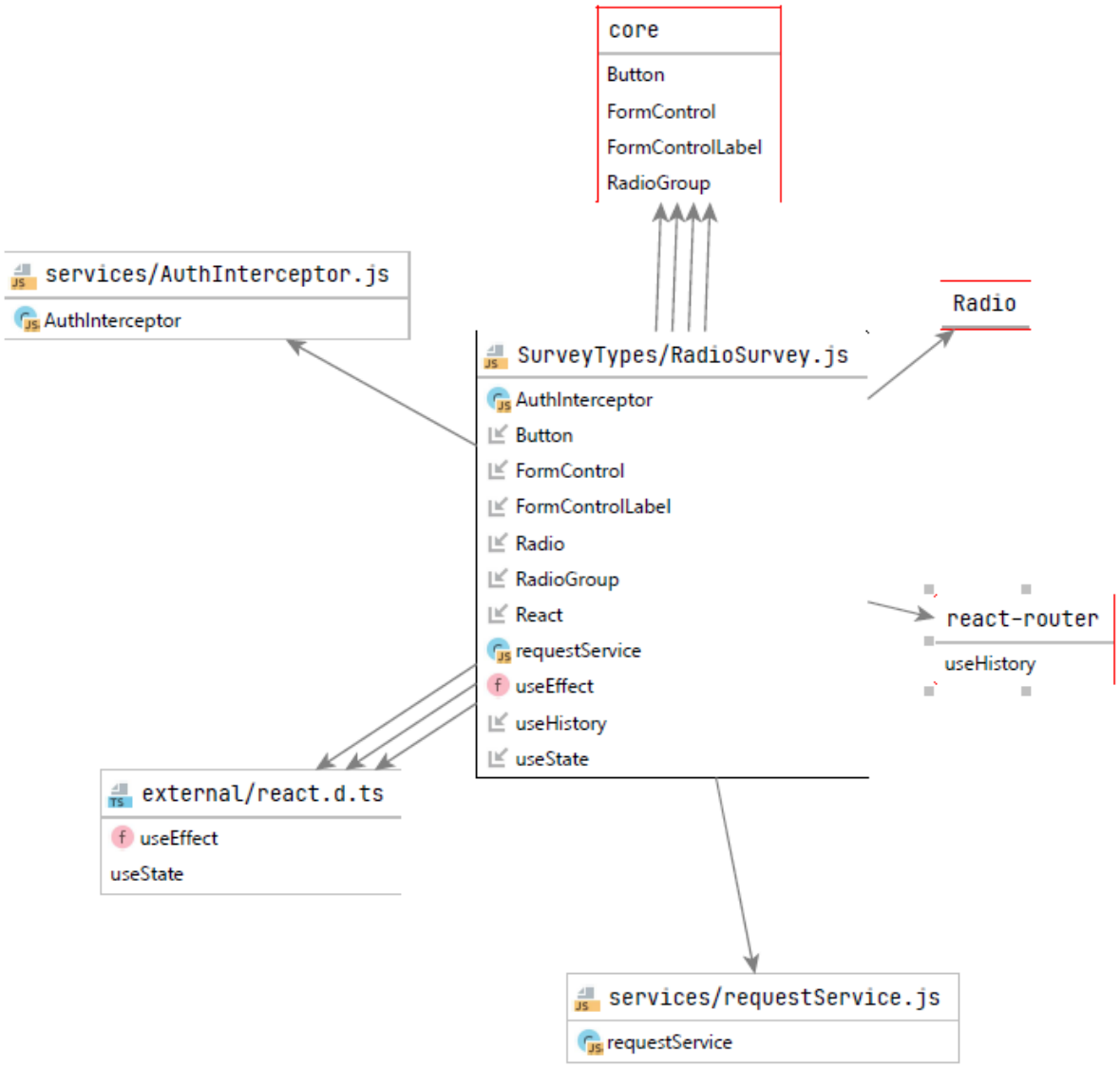

Figure 23: UML Diagram: RadioSurvey

Each radio survey gets passed in ist survey ID via a prop. By default, the survey is closed, and the toggle property is set to false. If the user decides he wants to open a survey, he can press the openButton, which calls the openSurveyFunction that is setting the toggle to true. Depending on the state of toggle (true/false), different elements are conditionally rendered. After opening it, the user can see the full form of the survey where all the survey information is displayed. He can choose one answer option when selecting an option, the according id of the answer is saved in the state. When the user decides to submit a survey, the form submit function is called: answerID is getting posted and sent to the back end as one submits. Additionally, the showResultChart-prop is getting set to true so that a pie chart of results is being displayed.

RadioSurvey
+ selectedIDs + id + toggle + history
+ openSurvey() + onValueChange() + getResults() + formSubmit()

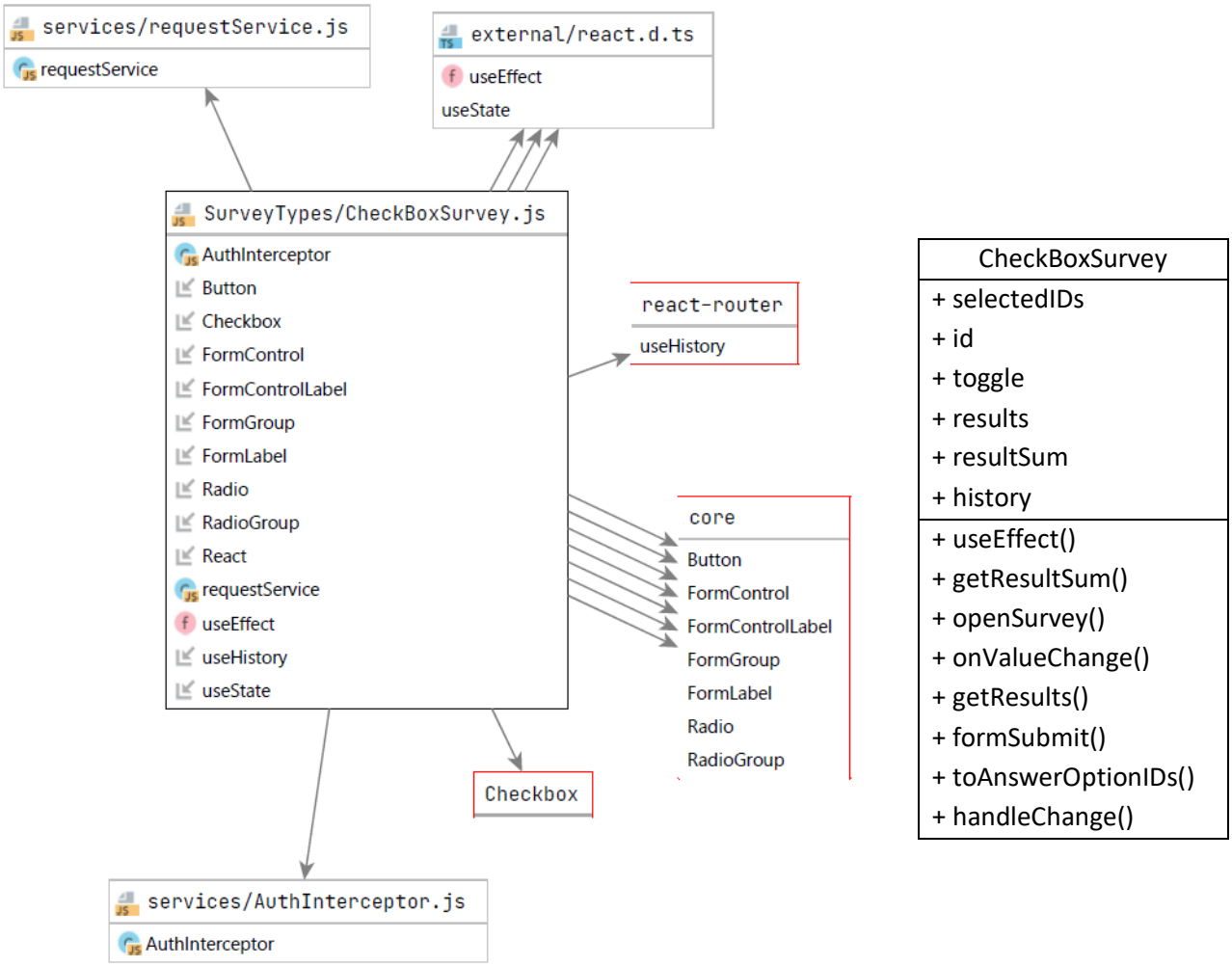

Figure 24: UML Diagram: CheckBoxSurvey

Each checkBoxSurvey gets passed in its survey ID via a prop. By default, the survey is closed, and the toggle property is set to false. If the user decides he wants to open a survey, he can press the openButton, which calls the openSurveyFunction that is setting the toggle to true. Depending on the state of toggle (true/false), different elements are conditionally rendered. After opening it, the user can see the full form of the survey where all the survey information is displayed. He can choose multiple answer options. When selecting an option, the according id of the answer is saved via a function call in an array. When the user decides to submit a survey, the form submits function is called: the answerOptionsArray is getting posted and sent to the back end as one submits. Additionally, the showResultChart-prop is getting set to true so that a pie chart of results is being displayed.

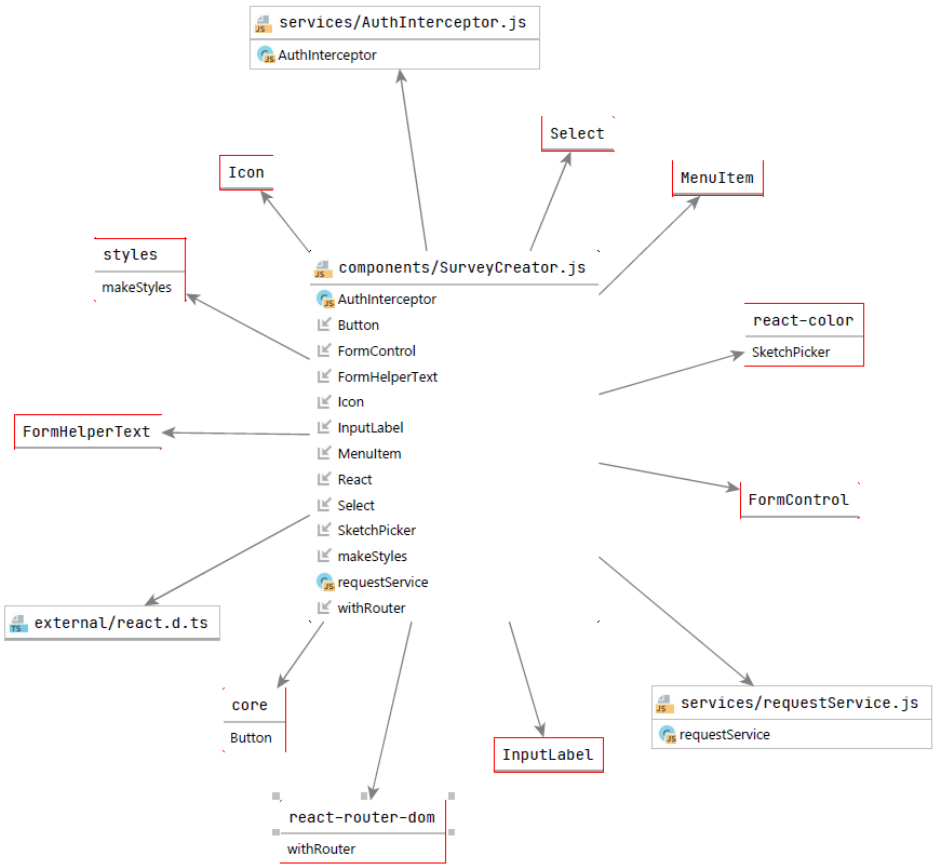

Figure 25: UML Diagram: SurveyCreator

SurveyCreator
+ surveyName + questionText + surveyTyp + selectedSurveyTyp + phase + answerOptions + selectedOption + background
+ onSurveyTypeChange(event) + onValueChange(event) + switchPhase(event) + goBack(event) + onChangeSurveyName(event) + onChangeQuestionText(event) + saveInput(e) + addNewItem() + handleChangeComplete(color) + cancelOptionInput() + onClickInputButton(event) + render()

The user starts creating the survey by entering a survey name, deciding on a surveyType (radio button or checkbox). After choosing type, colour, questions, and answer options, the survey owner can use the survey to implement it on a website.

Every input by the user is saved and controlled via the state of the component. When submitting a survey, the form submit function is called, and all the given inputs by the user are posted to the back end as one survey. After that, the user gets redirected to the home page.

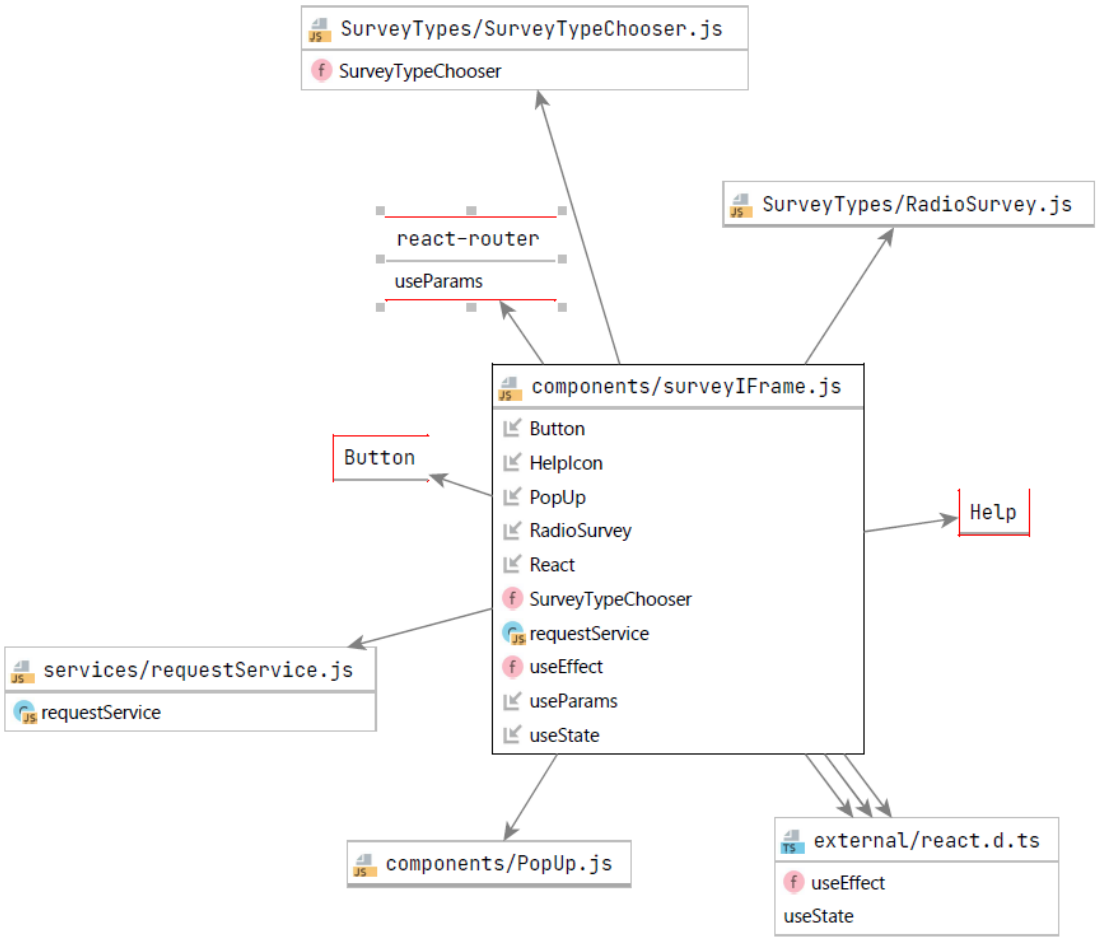

Figure 26: UML Diagram: surveyIFrame

The IFrame component fetches the specific surveyID from the URL path and then starts a request to get the survey by its specific id. One can also copy the code for the frame. It then uses surveyTypeChooser to display a specific survey in the format of an IFrame.

SurveyIFrame
+ id + survey + seen + buttonPopUp
+ useEffect()

5 Testing

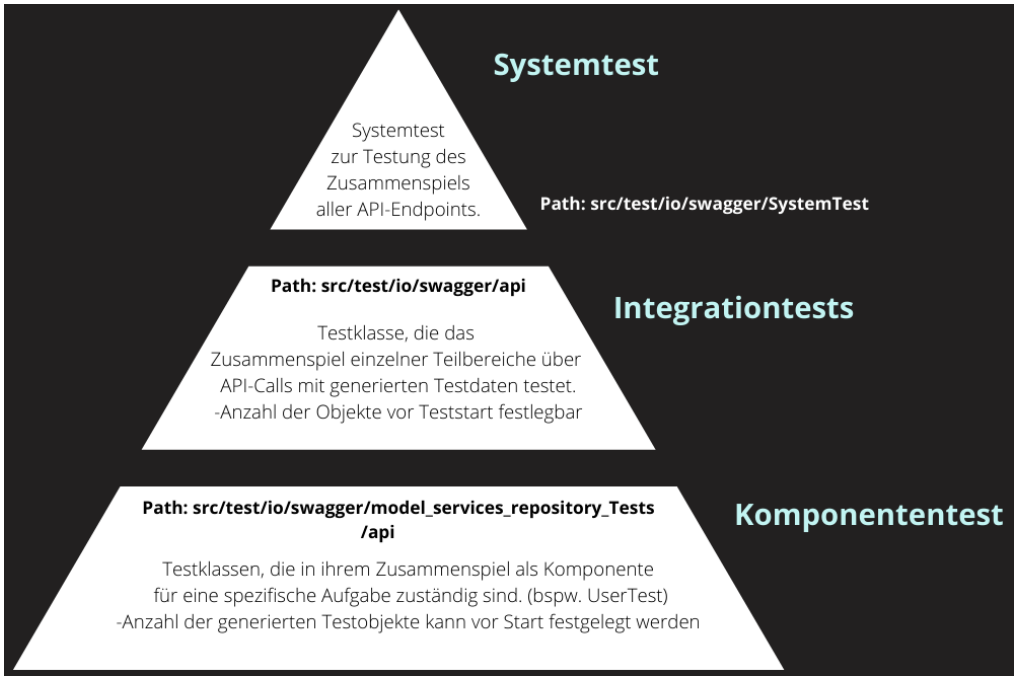

Figure 27: Tests overview