Technische Universität Chemnitz

Fakultät für Informatik

Professorship of Data Management

Datenbanken und Web-Techniken

Project Report On

Chemnitz BildungsZentrum

Author:

Roshita Shakya

Email: roshita.shakya@s2023.tu-chemnitz.de

M.Sc. Web Engineering

Professorship of Distributed and Self-organizing Systems

Matriculation Number: **807098**

TABLE OF CONTENTS

1. Project Overview

1.1. Background

Education and care are basic needs of every human being. In today's digital age, finding information about the facilities providing such services is crucial. Knowing where these facilities are located and how to reach them is a key decision-making factor. Therefore, having easy access to detailed information about these facilities on the web is necessary for informed decision-making.

The goal is to create a web application that fits in this domain which simplifies the process of finding such facilities in Chemnitz, thereby supporting easy access to learning and development.

1.2. Introduction

Chemnitz BildungsZentrum is a digital platform designed to streamline the process of locating learning and development facilities. It leverages publicly available datasets of Kindergartens, Schools, Social Child Projects, and Social Teenager Projects. Hence, users are able to look for these facilities using an interactive map interface and a variety of filters. The map displays search results of the filters as markers which are color-coded based on the type of facility it represents. Users can click on these markers to access detailed information about each facility, including address, contact information, opening hours, wheel chair possibility etc. Users also have the option to set their home address to visualize the facilities near their home on the map and estimate the distances between the home and the facility.
The application is built to accommodate both web and mobile users, also ensuring it is supported by a variety of web browsers, including Safari, Mozilla Firebox, and Chromium variants. Users can register themselves on the platform and log in to explore the facilities.

1.3. Features

- The user interface is responsive and easy to navigate.
- Authentication is implemented in JWT.
- Users can filter the facilities through one or a combination of more than one category of facilities (School, Kindergarten, Social Child Project, Social Teenager Project)
- Additionally, users can filter the school through different types (Berufsbildende Schule, Förderschule, Grundschule, Gymnasium, Overschedule, Schule des zweiten Bildungsweges, Sonstige Einrichtung, etc.)
- Users can set one home address by searching the address or dragging the marker to the home location.
- Users can set one facility as Favorite, whose information is shown in the user profile section.
- User can update their profile name and email address.
- After every action, a corresponding success or error notification is displayed.
- User can permanently delete their account.

2. Project Architecture

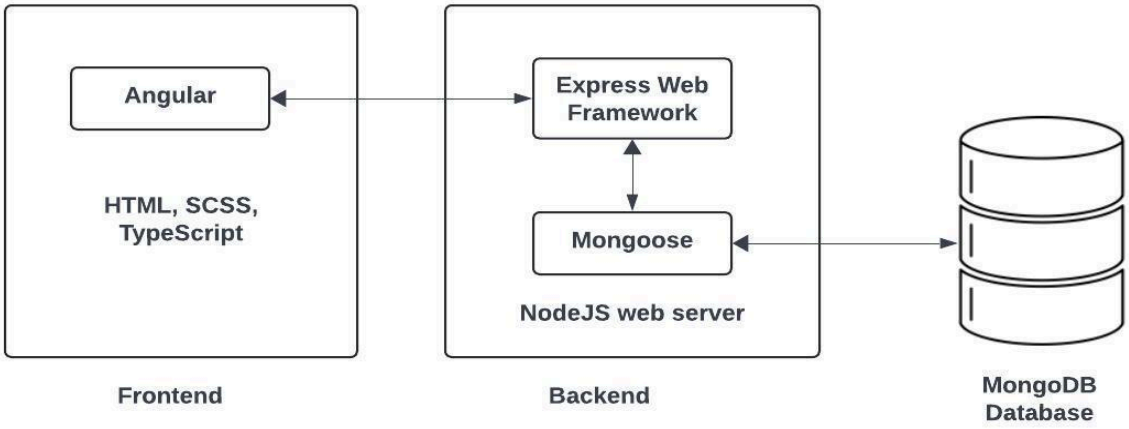

Figure 1: MEAN Stack Development Architecture

This web application is built using MEAN stack technology. MongoDB is primarily used for databases. Express and Node.js have been used to develop the API on the application's backend and Angular is used on the client side for frontend integration of the REST APIs.

Express and Node.js Server Tier: Node.js is a cross-platform JavaScript runtime environment. It is built on the V8 JavaScript engine, which compiles JavaScript to native machine code, resulting in fast execution times [2]. Express is the standard server framework for Node.js which is minimal but at the same time provides a robust set of features with a streamlined configuration process [3]. This is the reason Node.js and Express were used for the backend development of this web application, allowing for rapid prototyping without much configuration overhead.

Angular Frontend Tier: Angular is a typescript-based front-end framework for developing web applications. Its two-way data binding synchronizes the model and views automatically [4]. This feature is important for our web application to allow real-time updates of the map based on the changes in an underlying data model. Additionally, the component-based architecture of Angular has made it easier to manage the state and propagate the changes in the map data throughout this application.

MongoDB Database Tier: MongoDB is a NoSQL document-based database whose architecture comprises collections (the SQL equivalent of tables) and documents, which are made of key-value pairs [1]. The document model allows for the omission of fields rather than storing null values, ensuring a cleaner data set and simplifying data management. Since this web application is based on diverse open data, where the consistency can vary between the records, MongoDB was chosen over other relational databases to avoid the complexity associated with null values handling.

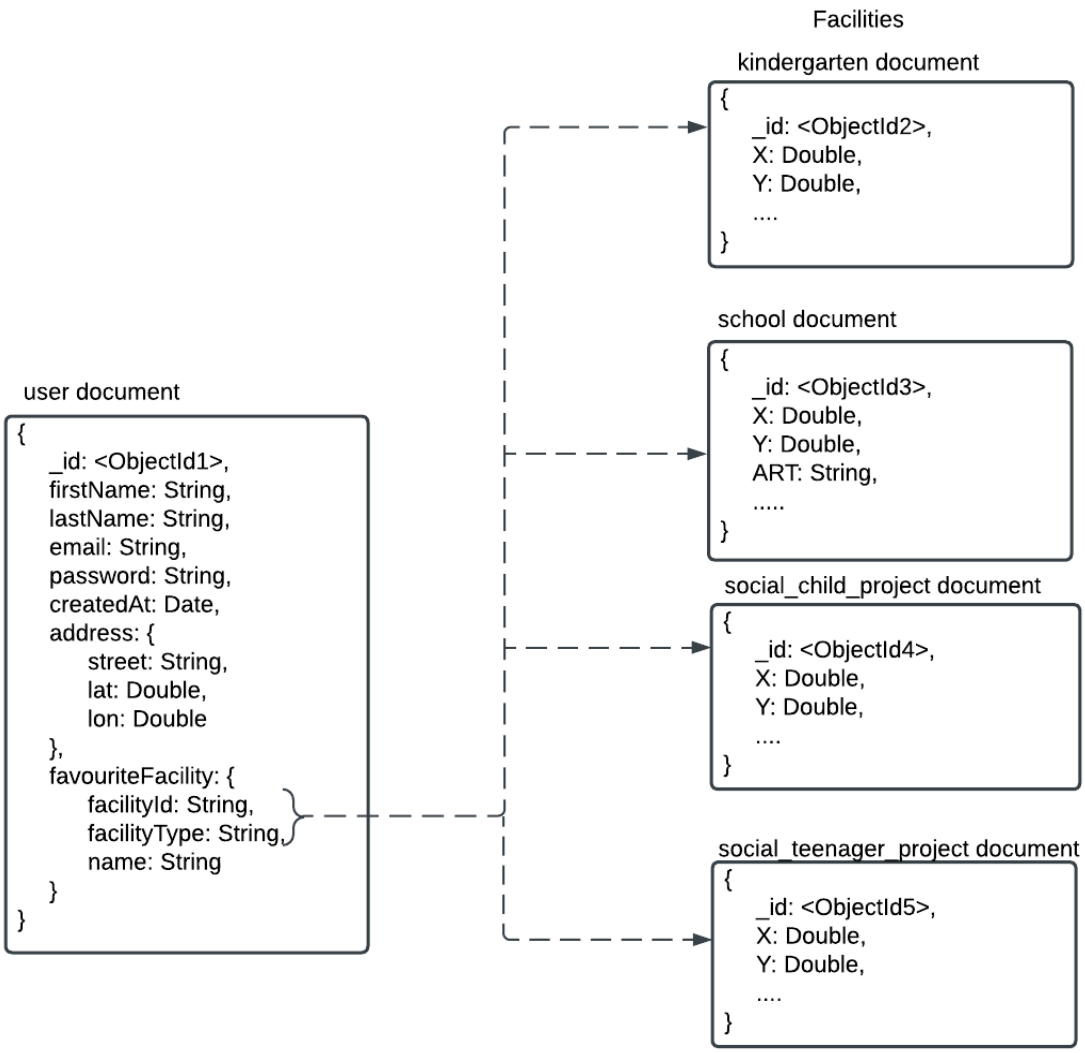

Figure 2: Database Model of the application

The above diagram shows a brief overview of the MongoDB model of the web application. We have a user document containing user details such as firstName, lastName, email & password. Given that, each user can have only one home address and only one favorite facility, an embedded data model is used instead of creating a separate collection for user address and user favorite facility. These values are by default null at the beginning when the user gets created. For the facilities, we have 4 different collections namely: Kindergarten, School, Social Child Project, and Social Teenager Project, for which we have integrated the dataset provided by Open Data Portal: Stadt Chemnitz. The favorite facility object of a user contains the facility ID and facility type which is used to identify the collection for the favorite facility. Additionally, the name of the favorite facility is also stored since we are also depending on OpenStreet API for displaying the name of the facility to the user in case the Open dataset does not contain name fields.

3. Used Tools and Technologies

3.1. Client Side

Bootstrap: It is a powerful, open-source frontend toolkit offering predefined classes [5]. With its help, we have created the responsive design for this web application using the grid structure.

SCSS: It is a part of larger SASS (Syntactically Awesome Style Sheets) which extends the capabilities of regular CSS by providing additional features. This application leverages the features of SCSS like defining reusable values for colors to use throughout the application and mixins to define and reuse style blocks which can be included with other styles.

The following npm packages were used:

ng-zorro-antd: It is an Angular UI component library. It includes a wide range of pre-built, customizable UI components which are designed to work together seamlessly [6]. UI components such as icons, modals, forms, buttons, and notification services of this library have been used extensively to build this web application.

leaflet: It is an open-source, JavaScript library for interactive maps [7]. Due to its simplicity and lightweight nature with good performance, this package is used for inserting the map and placing the markers in this application.

leaflet.markercluster: It is a plugin for Leaflet, that provides a way to group nearby markers into clusters when the map is zoomed out [8]. Since the user of this application can search for the facilities by more than one category, this can result in a larger number of markers on a map. To prevent the overcrowding of markers and enhance readability, this package is used.

leaflet-control-geocoder: This is another plugin for Leaflet that allows users to search for specific locations on the map [9]. This package is used in this application to add the user's ability to search for addresses to plot their home address on the map.

3.2. Server Side

axios: It is a JavaScript library used for making HTTP requests [10]. Axios is used in this application for making HTTP GET requests to the open street map API to get more information about a facility like opening hours, wheelchair possibility, etc. based on the geodetic data.

bcrypt: bcrypt is a node js library used for encrypting passwords in the form of hash [11]. It is used in this application for hashing the password before saving to the database as well as comparing the user-entered password with the hashed one during the login process.

cors: It is a browser-implemented HTTP-header-based mechanism [12]. The backend of this application serves APIs that can be accessed by clients via desktop web browsers or mobile browsers running on a different domain, CORS enable the clients to make HTTP requests to the server and send a response by adding appropriate CORS Headers to the server's HTTP responses.

dotenv: DotEnv is a little NPM package that automatically inserts process.env object environment variables from a .env file [13]. The secret token for the JWT authentication is being stored in the env file, which is retrieved using the dot env package.

jsonwebtoken (JWT): It is used for securely transferring information as a JSON object [14]. This web application uses JWT for authentication, where a user receives a JWT when they login to the system. This token contains information about user identity.

mongoose: This Object Data Modeling (ODM) library provides a high-level abstraction over the MongoDB Node.js driver [15]. Leveraging this library, schema, data types, and validation rules have been defined for the data of this application.

nodemon: This is a tool that helps develop Node.js based applications by automatically restarting the node application when file changes in the directory are detected [16]. This package has been used to ease the development process of this application.

4. Features

4.1. Sign Up Interface

Users can come to this website and register themselves through the signup interface. To lower the user drop-off rate, the registration process is made pretty straightforward, requiring users to enter just their First name, Last name, email address, and the desired password to log into the system. This basic information is required to complete the signup. Email must be unique, and the password should be at least 8 characters long. If the entered email exists in our system, it will show an error message that there is an existing user with that email address. Furthermore, the password and confirm password fields should match each other. All these conditions must be met to enable the Sign Up button. The bcrypt package hashes the password before it is stored in the database. After successful signup, a corresponding success message is displayed to the user with a Sign-In button to redirect them to the sign-in page.

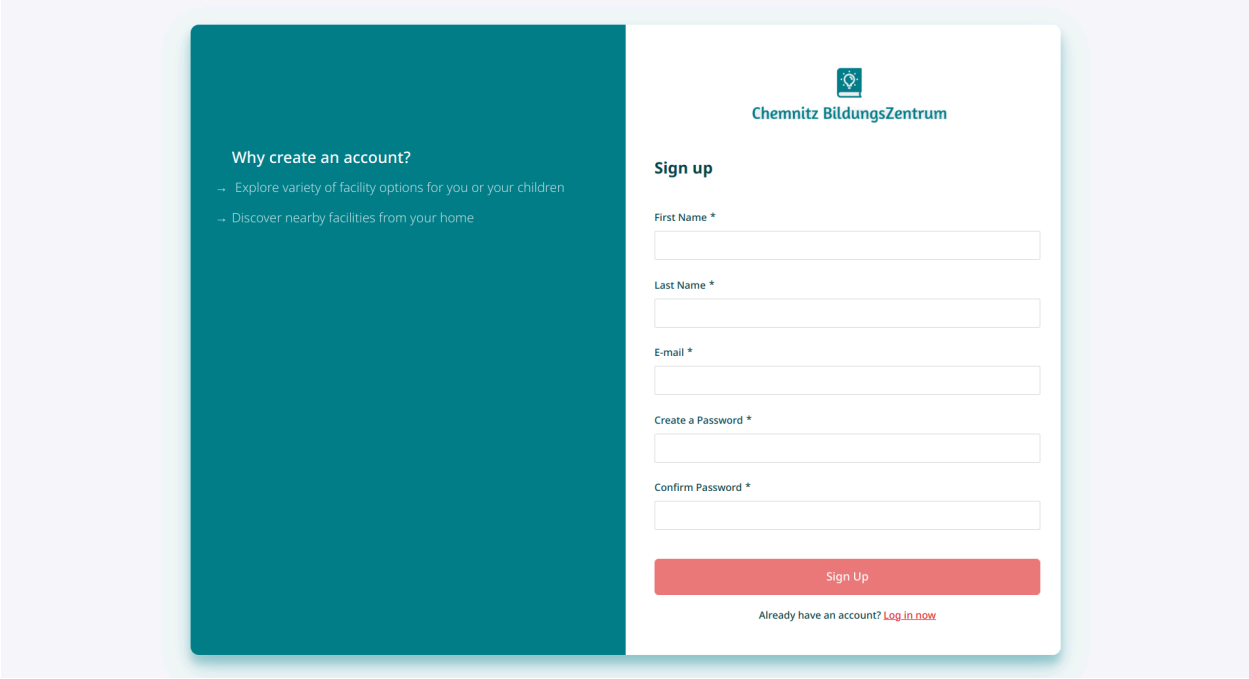

Figure 3: Sign Up Interface

4.2. Sign In Interface

The sign-in interface includes an email and password field. When the Sign-In button is clicked, the entered password gets compared with the encrypted hash password. If any of the email and password do not match, then the user is shown a corresponding error message. After all the fields are entered correctly, the user sign-in process succeeds and the user gets a JWT token in the response body, which is used for authorization of the user in the further API requests. The user is then redirected to the home page of the web application, where the user can search for facilities.

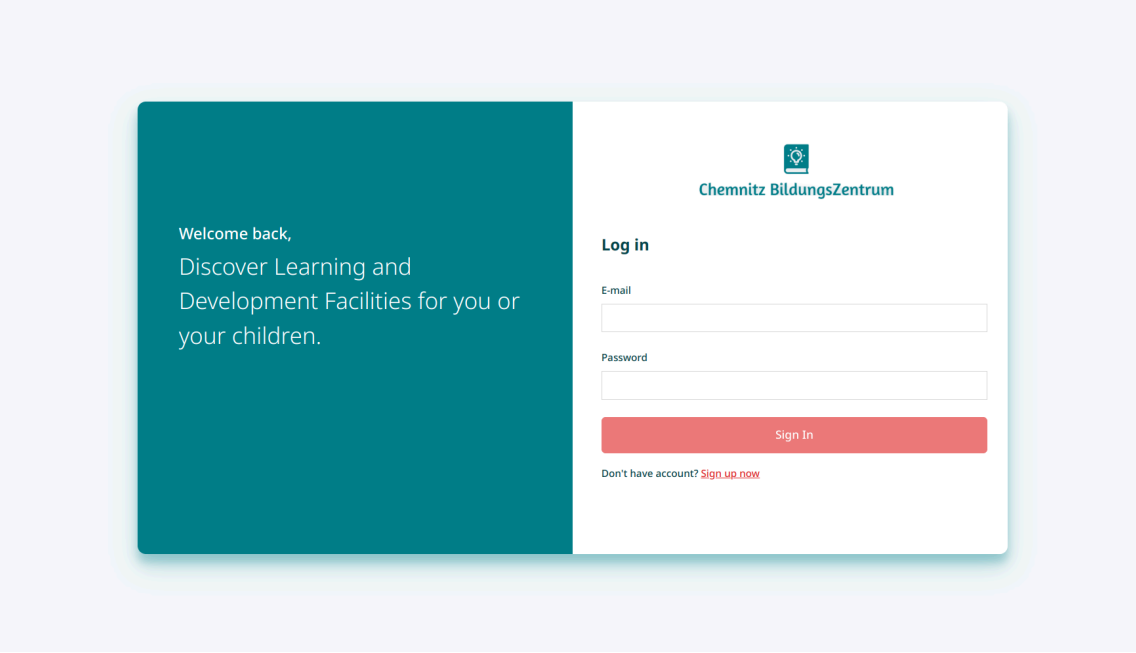

Figure 4: Login Interface

4.3. Home page Interface

This is the main page of our web application where users can see a map that is focused on Chemnitz. Above the map, there is an input box to set the user's home address, which will be displayed on the map. Next to that, there are filters to search for the desired facility type.

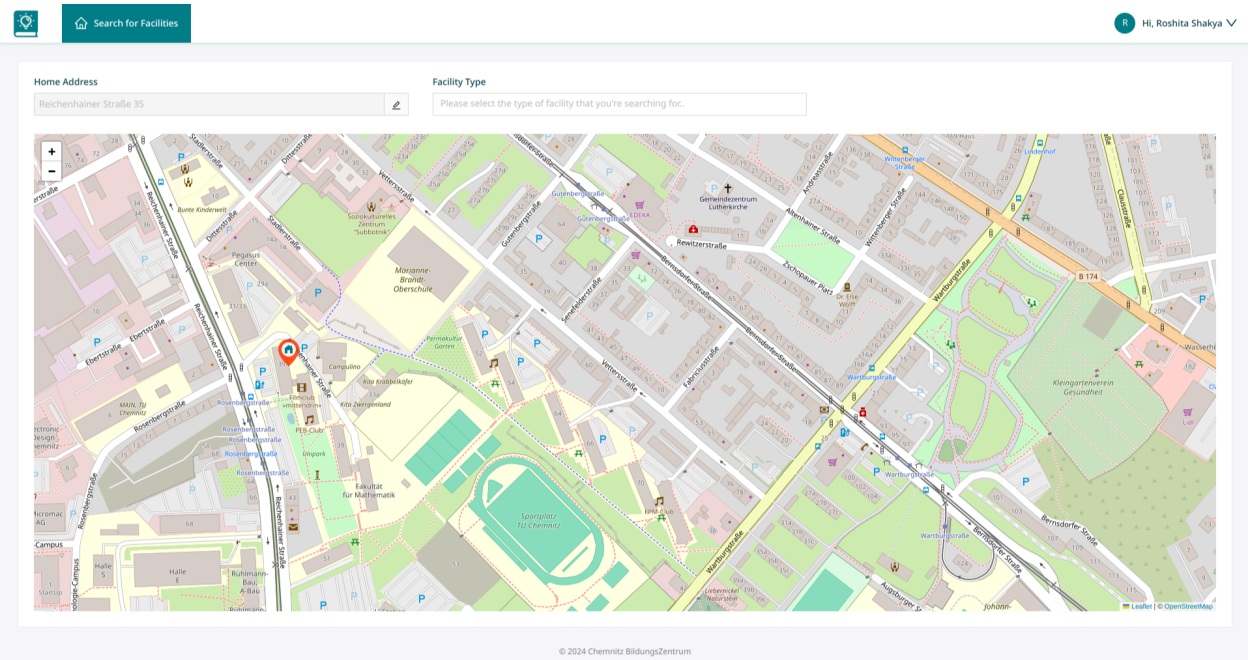

Figure 5: Home page Interface

4.4. Add Home Address Interface

Upon clicking the edit icon in the home address input box, a modal to set the user's home address appears. The modal includes a map with a marker which can be dragged to the desired home address of the user. An instruction note is placed below the map to guide the user to drag the marker for setting their address. Additionally, there is a search box on the top right side of the map where the user can directly search for the address by providing the text input. Once the search input for the address is entered, the search lists appear in a dropdown. If the user clicks an address from the search list, the marker is automatically dragged to that location. After the save button is clicked on the modal, the latitude and longitude along with the address name is sent to the backend for storing in the database. The modal closes, and the user can see the updated home address name in the input box as well as the marker of the home address is also displayed on the main map of the home page interface.

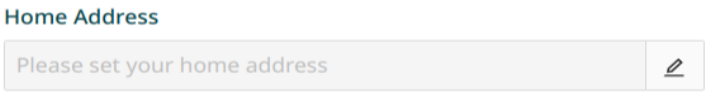

Figure 6: Home Page input group in the filter bar

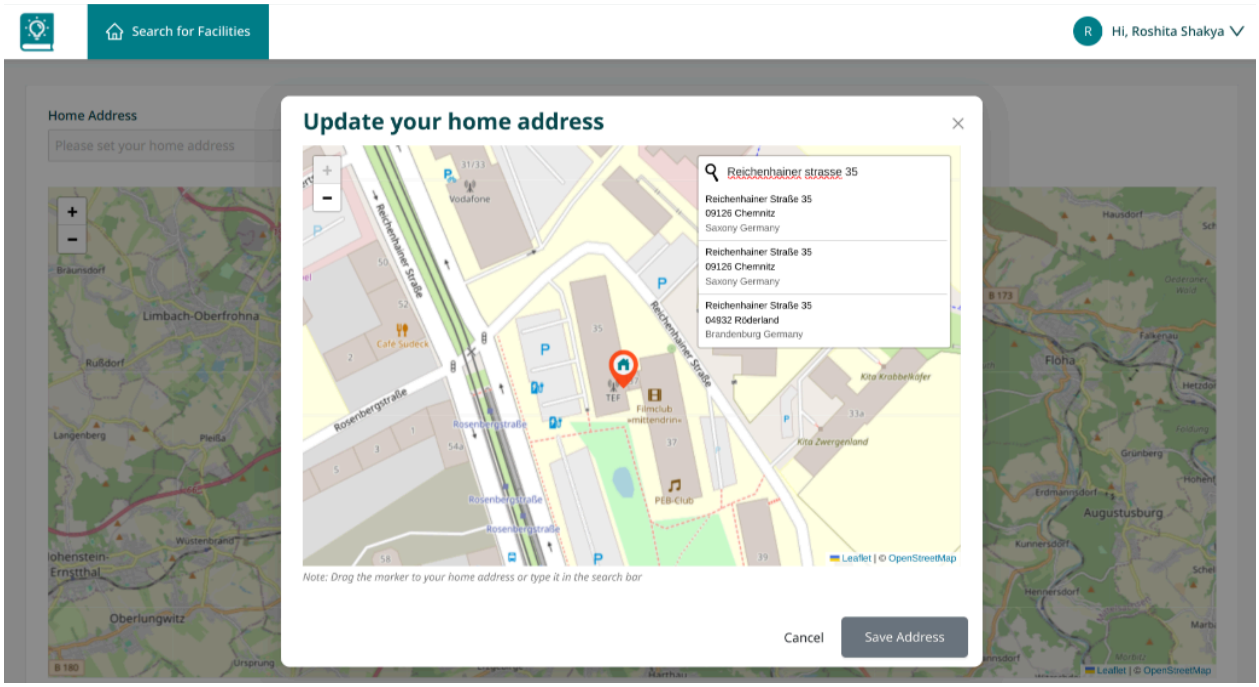

Figure 7: Modal for updating the home address

4.5. Filters in the map interface

The search filter for the facility includes a dropdown for the facility type. Since this web application has integrated the 4 data sets of Kindergarten, School, Social Child projects, and social teenager projects, these options are made available in the dropdown. It is important to note that this is a multi-select dropdown that means the user can filter by one or even more than one facility type. All these 4 facility types are distinguished by 4 colors which are visible in the tags once the user selects the facility type. Kindergarten, School, Social Child Project, and Social Teenager Projects are denoted by Yellow, Blue, Pink, and Purple colors respectively.

When the facility type of "School" is selected, an additional filter option for the different types of Schools as present in the database appears next to the "Facility Type" filter. The options in the "School Type" filter are dynamic based on the type of school data present in the dataset. Like the facilities filter, school-type filters can also be selected by one or a combination of more than one type.

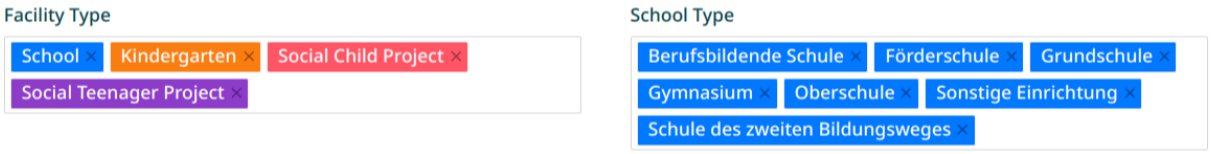

Figure 8: Filter bar (with filters selected)

4.6. Map Interface with Markers

Upon selecting a filter, the map is updated with the markers for the facilities. The map view is also automatically adjusted based on the markers plotted in the map. To prevent marker overlap and clutter on the map, at first the markers are grouped into clusters denoting the number of facilities in a particular area. Then, users can click or zoom into the cluster, to view the individual markers.

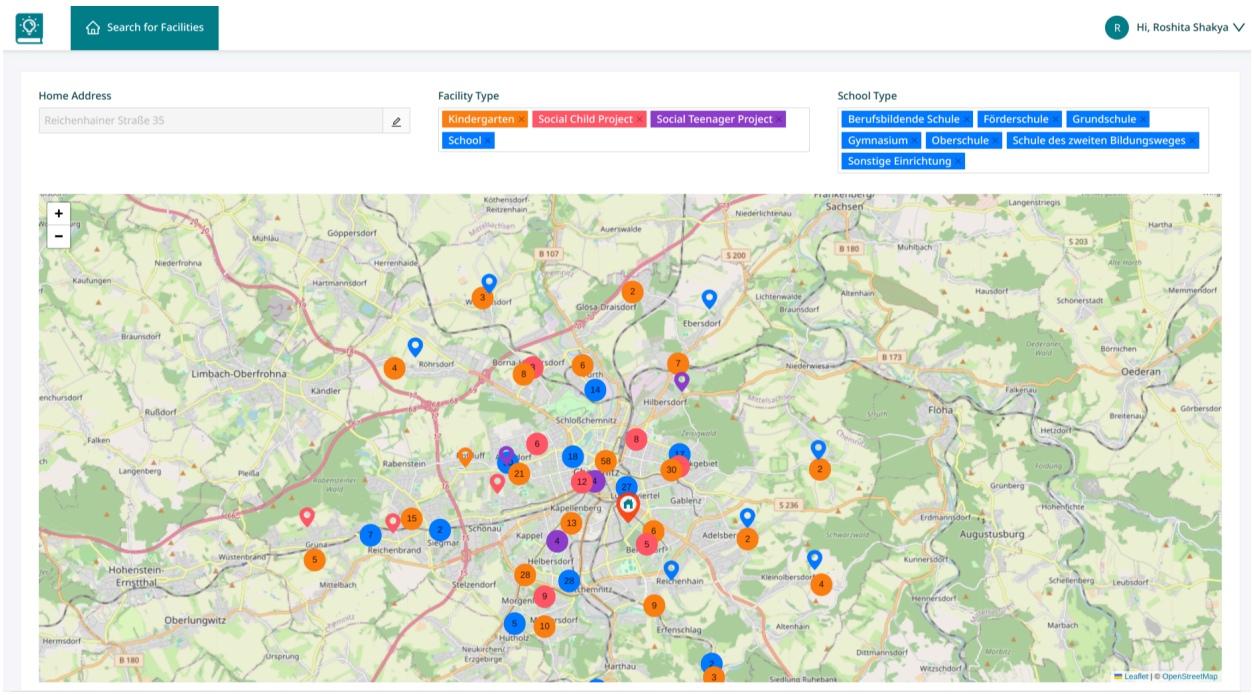

Figure 9: Facility Marker clusters in the map when all filters are selected

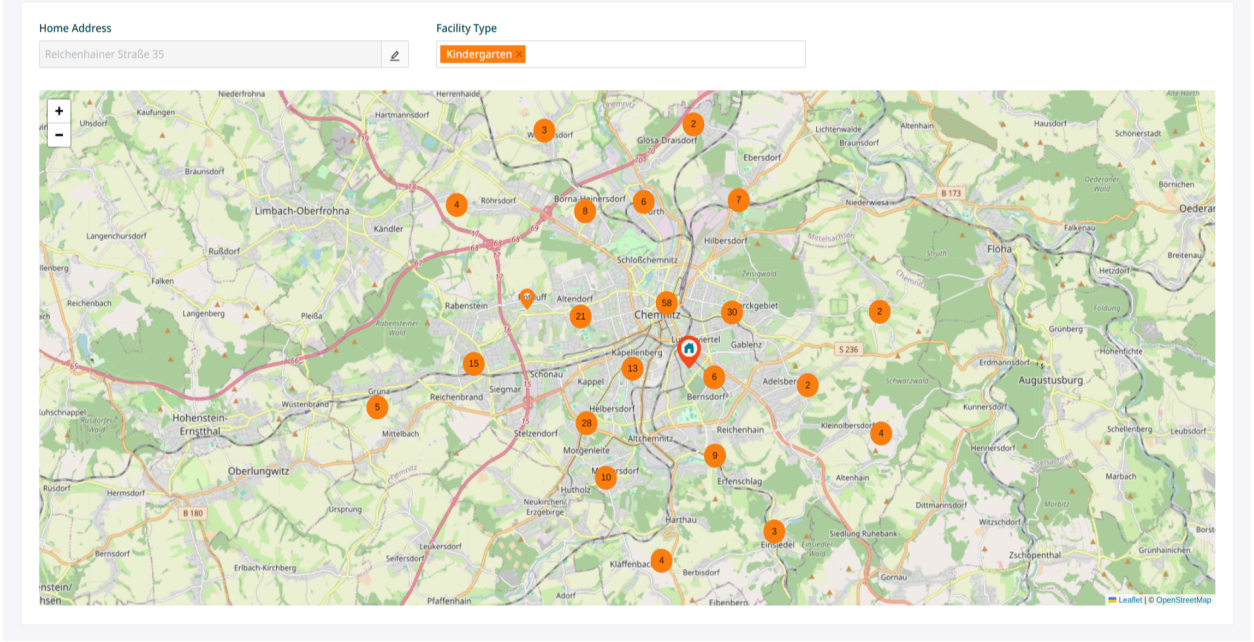

Figure 10: Marker clusters when only one of the filters is selected

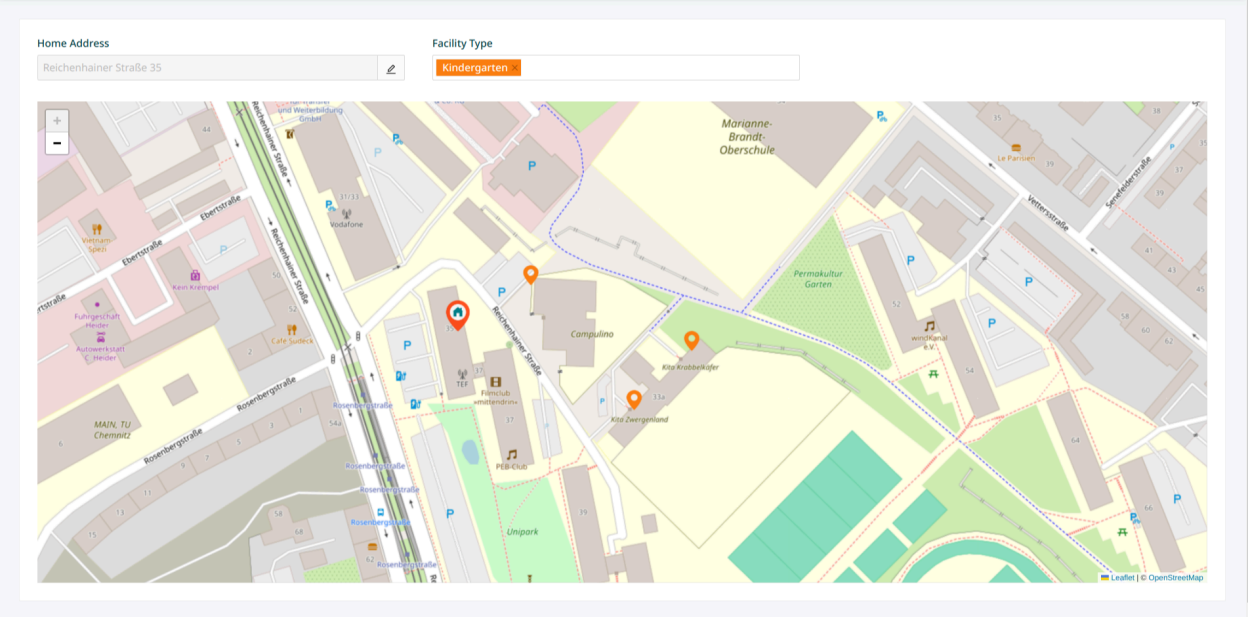

Figure 11: Map view when marker clusters are zoomed or clicked

4.7. Facility Popup Interface

When a marker is clicked on the map, a popup with information about the facility opens. To get the information about the facility, an API is called to the backend. The backend retrieves information such as name, address, telephone, fax, website, and email address from the dataset. If the information about if the place is Barrier-free and integrative is available, then a checkmark or cross mark is shown in the popup depending on the value. Additionally, the backend also calls the OpenStreetMap API with the latitude and longitude of the clicked facility to retrieve more information about the place, such as the opening hours, wheelchair accessibility, etc. Name, address, and other contact details are also retrieved from the external API if the dataset does not contain such information. Finally, all these data are combined and displayed in the front end. Along with this information, the routing distance of the selected facility from the home address is also displayed in the popup if the home address has been set for that user.

The popup also includes a “Mark as Favorite” button that allows users to select a facility as a favorite. Users can differentiate if the facility has been marked as their favorite based on the fill of the favorite icon. If the opened facility has already been marked as favorite, the favorite icon is filled with color and the button text “Marked as Favorite” appears. Users can toggle this favorite icon to set and unset their favorite facility. Only one facility can be set as a favorite for one user. Therefore, if a facility is marked as a favorite, the old favorite gets removed automatically.

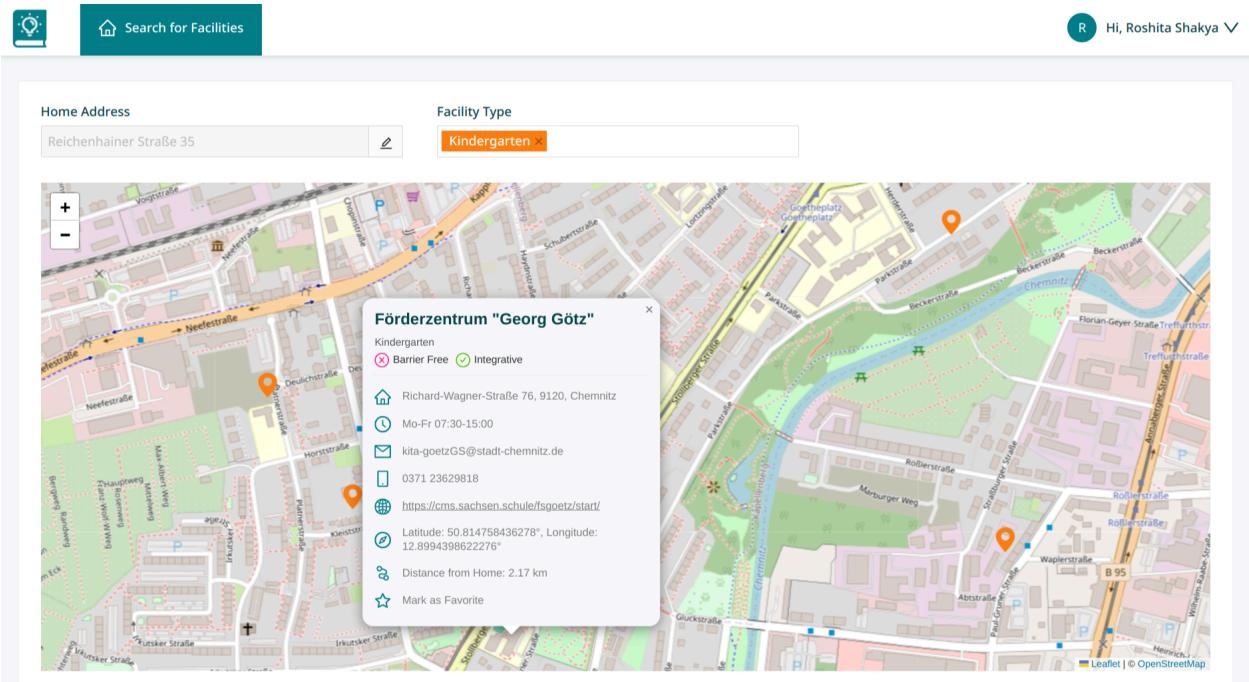

Figure 12: Popup when a marker is clicked

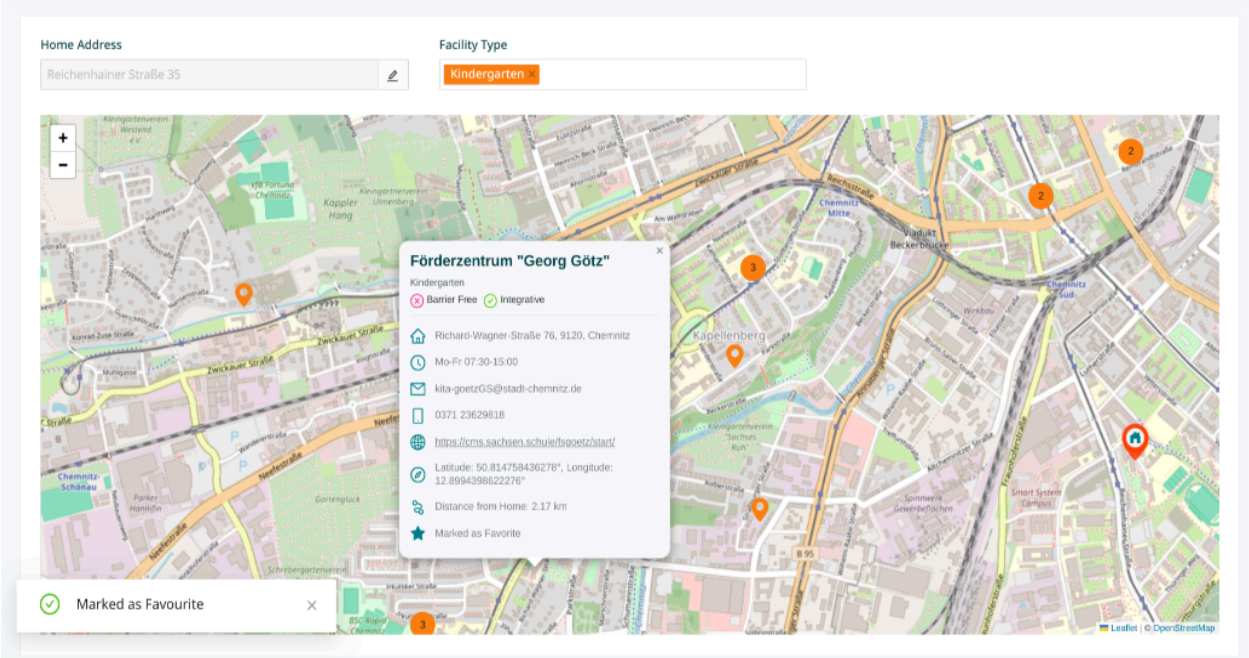

Figure 13: Popup when the facility is marked as a favorite

4.8. User Profile Interface

Users can go to their profile by clicking the “Profile” button which appears on a dropdown when the user name is hovered on the top right of the navigation bar. The following page with user details and the user's favorite facility is shown:

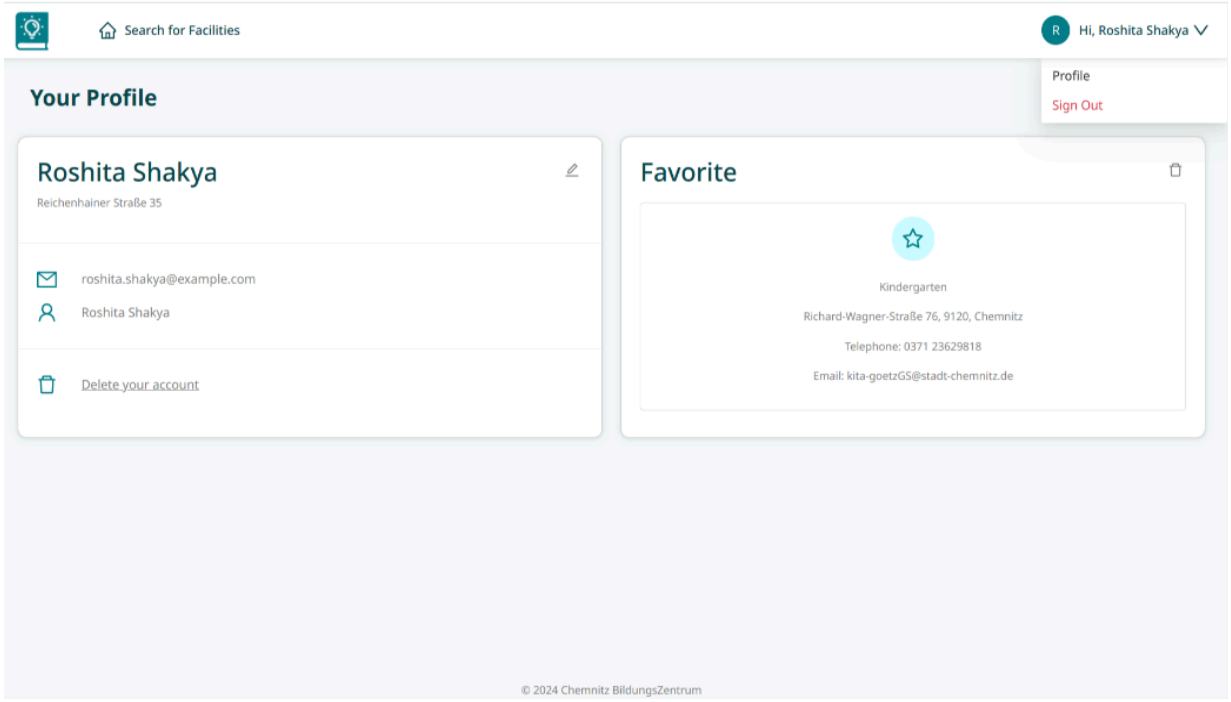

Figure 14: User Profile

4.8.1. Update User Profile Interface

Upon clicking the edit icon on the top right of the user profile card, an edit modal appears. Here, user can update their First Name, Last Name, and Email address. A unique email address has to be entered, otherwise an error message is displayed to the user.

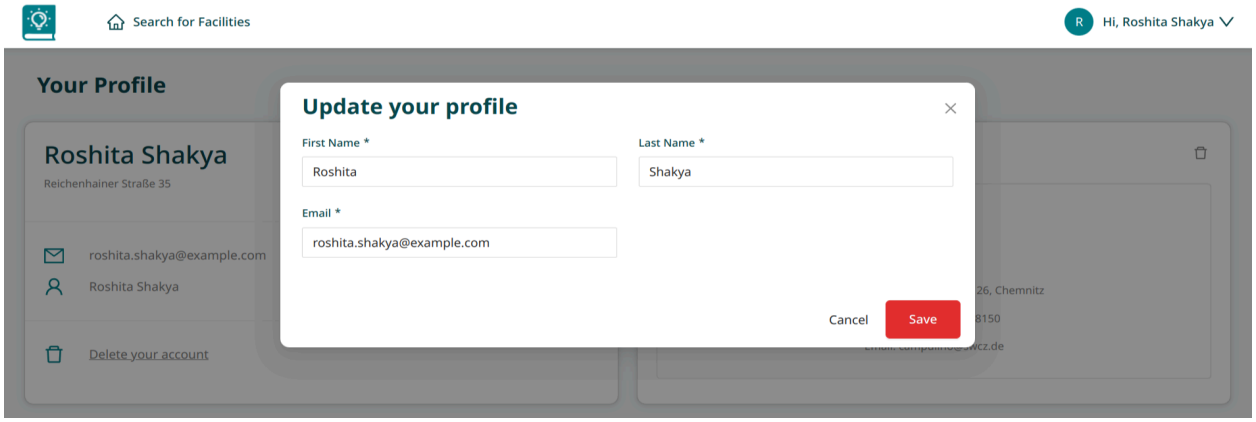

Figure 15: Modal for updating user profile

4.8.2. Delete Account

User have the ability to delete their account in this web application. When a user clicks the "Delete your account" button from the user profile card, a modal appears as below, which asks the user if they want to permanently delete their user account. If the user confirms this button, then the user account will be permanently deleted from the database and the user will be logged out and redirected to the login page.

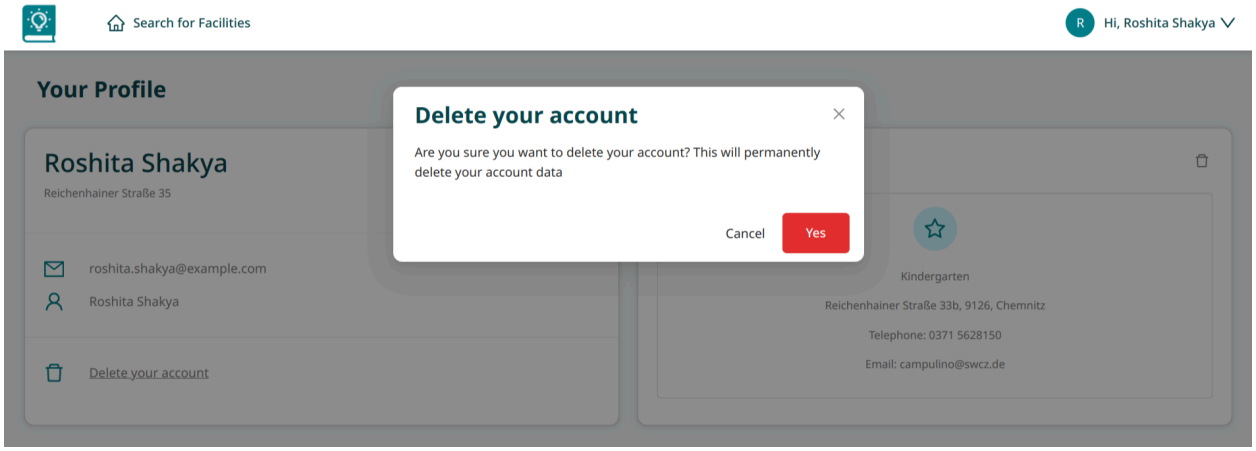

Figure 16: Modal for confirming the delete user account

4.8.3. User Favorite Interface

The User Profile section also includes a card that displays the details of the facility that the user has marked as a favorite. The details include the name, Address, Telephone and email address of the favorite facility. The favorite facility can also be deleted by clicking the delete icon on the top right of the favorite card. Once the delete button is clicked, a modal is shown to the user to confirm if they want to remove the facility from favorites. If the user clicks "Yes", the facility gets unmarked from favorites, and empty data is shown on the Favorite card. The corresponding success or error notification appears on the bottom left corner of the web browser.

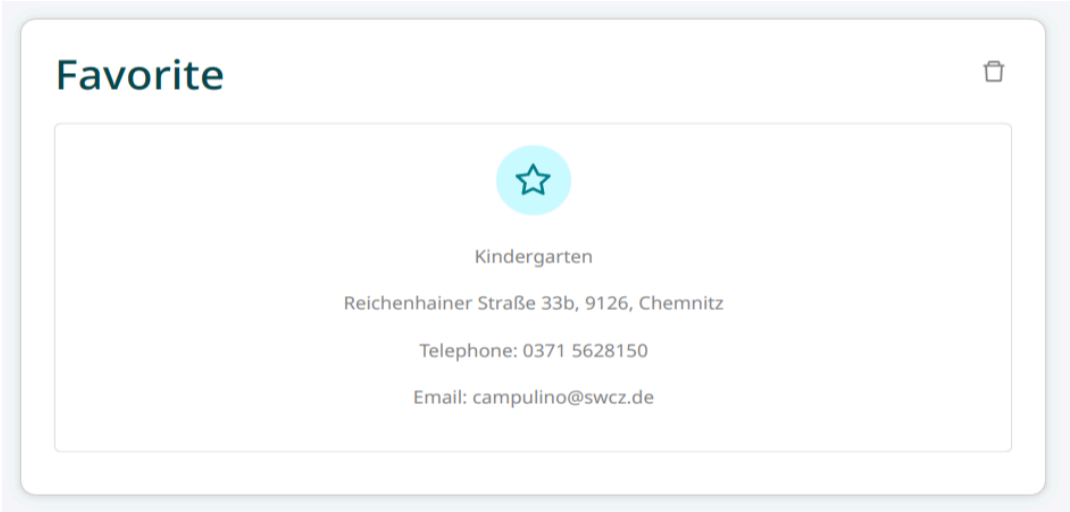

Figure 17: Favourite Facility Card in User Profile

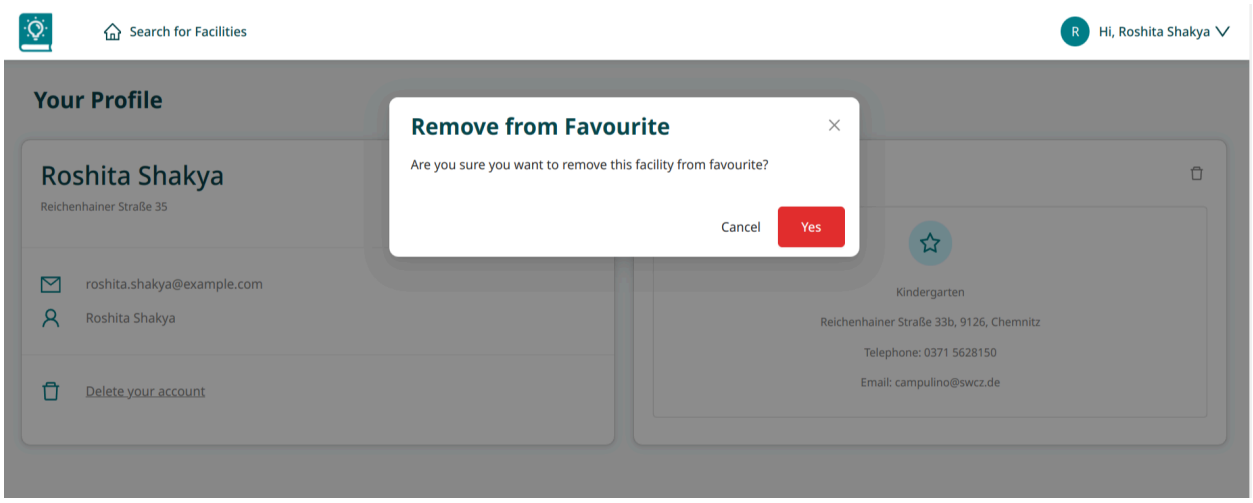

Figure 18: Modal for removing the user's favorite

4.9. Sign Out Interface

Users can sign out from the application by clicking the Sign Out menu from the account dropdown menu which is located in the top right corner of the navigation bar. A modal for signout confirmation appears and if the user confirms, he gets signs out and gets redirected to the sign in interface.

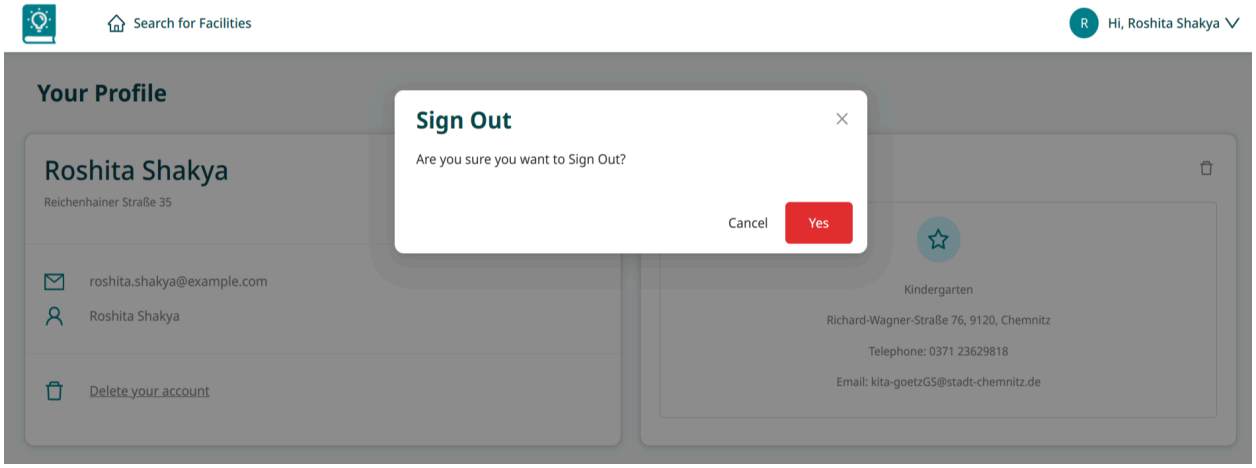

Figure 19: Modal for confirming Sign out action

4.10. Responsiveness

All the screens in this web application have been developed carefully considering the responsiveness in mind, ensuring an optimal user experience across a wide range of devices and screen sizes. The following screenshots provide an overview of the application's view in mobile devices.

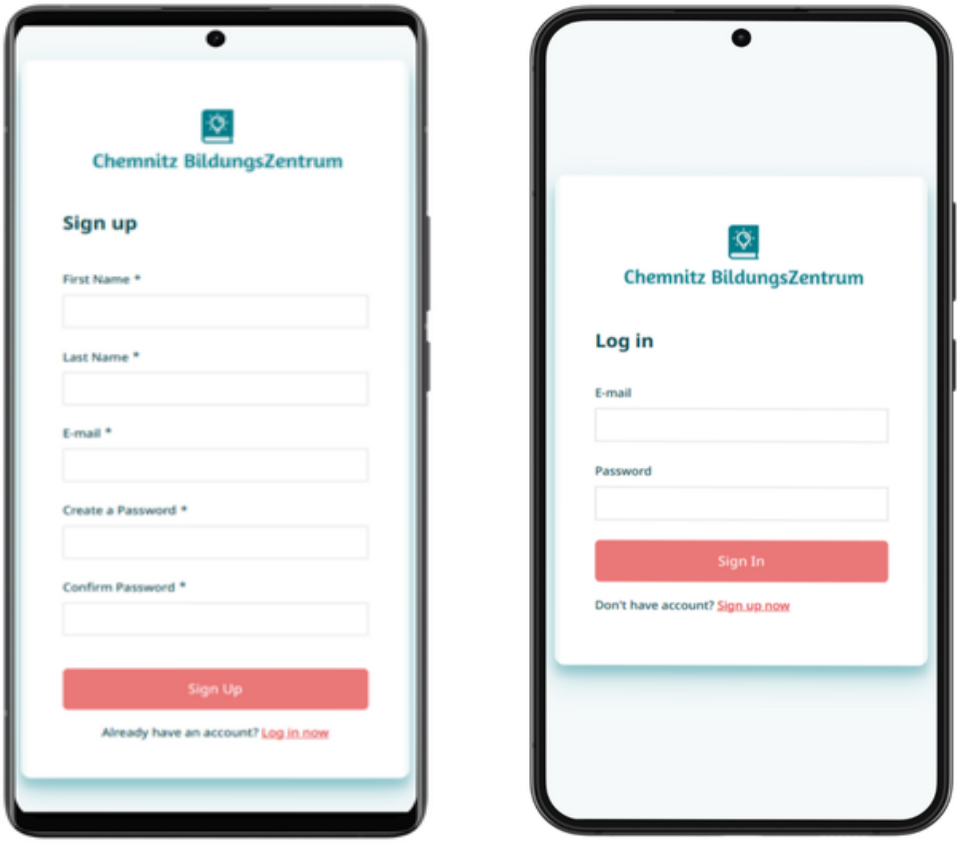

Figure 20: Sign In and Sign Up interface for mobile devices

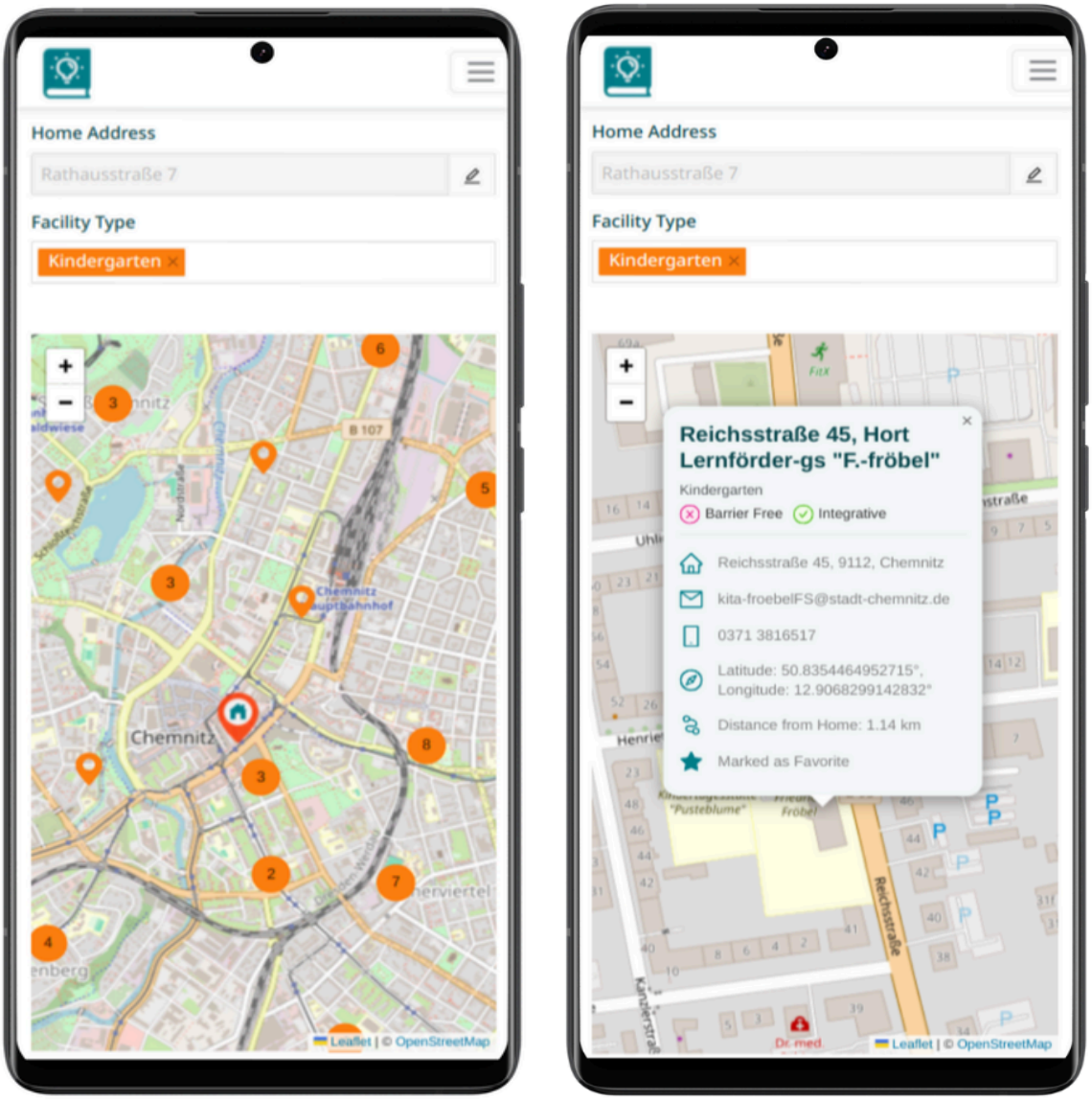

Figure 21: Map Interface with filters, markers, and popup in mobile devices

Figure 22: Update home address in mobile devices

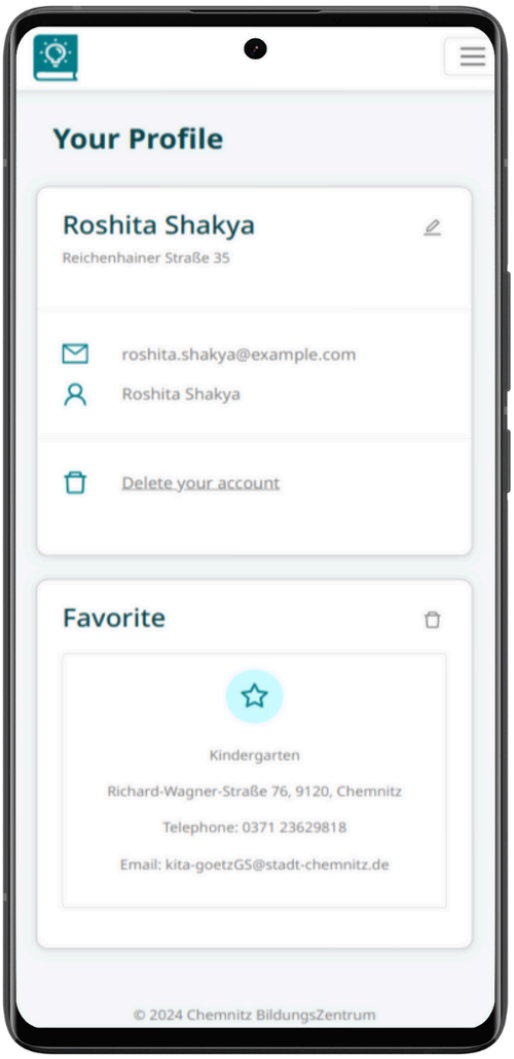

Figure 23: User Profile Interface in mobile devices

5. Conclusion

Therefore, following the three-tier architecture of MEAN stack technology, the project was built. REST API was used to facilitate communication between the front end and back end. Additional details about the frameworks, packages, and libraries that were used can be found below. The Appendix section can be found at the end of the document which contains detailed documentation about all the API endpoints that were used to build the features in this web application.

6. Bibliography

[1] https://www.mongodb.com/

[2] https://nodejs.org/en

[3] https://expressjs.com/

[4] https://angularjs.org/

[5] https://getbootstrap.com/docs/5.3/getting-started/introduction/

[6] https://ng.ant.design/docs/introduce/en

[7] https://leafletjs.com/index.html

[8] https://www.npmjs.com/package/leaflet.markercluster

[9] https://www.npmjs.com/package/leaflet-control-geocoder

[10] https://www.npmjs.com/package/axios

[11] https://www.npmjs.com/package/bcrypt

[12] https://www.npmjs.com/package/cors

[13] https://www.npmjs.com/package/dotenv

[14] https://www.npmjs.com/package/jsonwebtoken

[15] https://www.npmjs.com/package/mongoose

[16] https://www.npmjs.com/package/nodemon

7. Appendix (API Documentation)

7.1. Register a User

Description: This is the endpoint for registering a user based on first name, last name, email, and password. All fields are mandatory for signup.

URL	/api/v1/register
HTTP Method	POST
Header	Content-Type: application/json

Request Body	{
	"firstName": string,
	"lastName": string,
	"email": string,
	"password": string
	}

Example Body	{
	"firstName": "Emily",
	"lastName": "Johnson",
	"email": "emily.johnson@example.com",
	"password": "P@ssw0rd123"
	}

Success Response

When all the required fields are entered and validations are satisfied, user registration becomes successful and user data is sent in response.

Status	201 Created

Response Body	{
	"success": boolean,
	"message": string,
	"data": Object
	}

Example Response Body	{
	"success": true,
	"message": "User registered successfully"
	}

Error Response
When an email is not unique, user registration will not be successful and the error message is sent in the response body.

Status	409 Conflict

Response Body	{
	"success": boolean,
	"message": string
	}

Example Body	{
	"success": false,
	"message": "A user with that email already exists."
	}

7.2. User Login
Description: This is the endpoint for user login with email and password.

URL	/api/v1/login
HTTP Method	POST
Header	Content-Type: application/json

Request Body	{
	"email": string,
	"password": string
	}

Example Body	{
	"email": "emily.johnson@example.com",
	"password": "P@ssw0rd123"
	}

Success Response
When the supplied credentials match with a database record, login is successful and a json web token along with user data is sent in the response body.

Status	200 OK

Response Body	{
	"success": boolean,
	"message": string,
	"token": string,
	"data": Object
	}

Example Response Body	{
	"success": true,
	"message": "Authentication successful",
	"token": “eyJhbGciOiJIUzI1NiIsInR5cCVCJ9.eyJpZCI6Ij”,
	"data": {
	"_id": "66624d410760cca467de11f9",
	"firstName": "Emily",
	"lastName": "Johnson",
	"email": "emily.johnson@example.com",
	"address": {
	"_id": "66625f91337f9ca55eac8725",
	"userId": "66624d410760cca467de11f9",
	"street": "Börnichsgasse 2",
	"lat": 50.833819752050125,
	"lon": 12.91730497563156
	}
	}
	}

Error Response
When a user's email or password does not match with any of the database records, an error message is sent in the response.

Status	404 Not Found

Response Body	{
	"success": boolean,
	"message": string,
	}

Example Body	{
	"success": false,
	"message": "Authentication failed. User with that email address does not exists."
	}
	
	OR
	
	{
	"success": false,
	"message": “"Authentication failed. Password incorrect.”
	}

7.3. User Logout

Description: This is the endpoint for user logout.

URL	/api/v1/users/logout
HTTP Method	POST
Header	Authorization: <JWT Token> Content-Type: application/json
Request Body	Null

Response Body	{
	"success": boolean,
	"message": string
	}

Example Response	{
	"success": true,
	"message": “User successfully signed out”
	}

Response Status	● Success: 200 OK ● Error: 500 (Internal Server Error)

7.4. Update User Address

Description: This is the endpoint for a user to update their home address

URL	/api/v1/users/updateAddress
HTTP Method	PATCH
Header	Authorization: <JWT Token> Content-Type: application/json

Request Body	{
	“lat”: number,
	“lon”: number,
	“street”: string
	}

Example Body	{
	“lat”: 50.833819752050125,
	“lon”: 12.91730497563156,
	“street”: "Börnichsgasse 2"
	}

Success Response

When the supplied data are valid, the user address is updated successfully and address data is sent in the API Response.

Status	200 OK

Response Body	{
	"success": boolean,
	"message": string,
	"data": Object,
	}

Example Response Body	{
	"success": true,
	"message": "Address updated successfully",
	"data": {
	"street": "Börnichsgasse 2",
	"lat": 50.833819752050125,
	"lon": 12.91730497563156
	}
	}

Error Response
When either of the latitude and longitude fields are missing, user address validation fails and an error message is sent accordingly.

Status	400 Bad Request

Response Body	{
	"success": boolean,
	"message": string
	}

Example Body	{
	"success": false,
	"message": "Latitude field is required"
	}
	
	OR
	
	{
	"success": false,
	"Message": “Latitude and longitude field is required”
	}

7.5. Update User
Description: This is the endpoint for user login with email and password.

URL	/api/v1/users/update
HTTP Method	PATCH
Header	Authorization: <Token> Content-Type: application/json

Request Body	{
	"firstName": string,
	"lastName": string,
	"email": string,
	}

Example Body	{
	"firstName": “Martin”,
	"lastName": “Schröder”,
	"email": "martin.schroeder@example.com"
	}

Success Response

When the supplied fields are valid, user data is successfully updated in the database and an appropriate message with the updated user data is sent.

Status	201 Created

Response Body	{
	"success": boolean,
	"message": string,
	"data": Object
	}

Example Response Body	{
	"success": true,
	"message": "User updated successfully",
	"data": {
	"_id": "66624d410760cca467de11f9",
	"firstName": "Martin",
	"lastName": "Schröder",
	"email": "martin.schroeder@example.com",
	"createdAt": "2024-06-07T01:09:41.139Z",
	"address": {
	"_id": "66625f91337f9ca55eac8725",
	"userId": "66624d410760cca467de11f9",
	"street": "Börnichsgasse 2",
	"lat": 50.833819752050125,
	"lon": 12.91730497563156
	}
	}
	}

Error Response
If an existing email is sent in the request body while updating the user data, user unique validation fails and an error message is sent.

Status	409 Conflict

Response Body	{
	"success": boolean,
	"message": string,
	"data": Object
	}

Example Body	{
	"success": false,
	"message": "A user with that email already exists."
	}

7.6. Delete User Account
Description: This is the endpoint for permanently deleting a user account.

URL	/api/v1/users/delete
HTTP Method	DELETE
Header	Authorization: <Token> Content-Type: application/json
Request Body	Null

Success Response
When a user makes a delete request, the user is verified through the passed token, and the user gets deleted from the database. The below response body gets generated.

Status	200 OK

Response Body	{
	"success": boolean,
	"message": string,
	}

Example Response Body	{
	"success": true,
	"message": "User deleted successfully"
	}

Error Response
When a user cannot be identified by a given token or the user does not exist anymore, an error message is sent.

Status	404 Not Found

Response Body	{
	"success": boolean,
	"message": string
	}

Example Body	{
	"success": false,
	"message": "User not found"
	}

7.7. Get Facility Subtypes

7.7.1. Get School Subtypes
Description: This is the endpoint for retrieving the subcategories of schools. The retrieved categories of school types are used for displaying the options in the school type filter to search the different categories of schools

URL	/api/v1/facilities/subTypes
Query Parameters	“facilityType”: “school”
HTTP Method	GET
Header	Authorization: <Token> Content-Type: application/json
Request Body	Null
Response Body	{ "success": boolean, "data": string[]

	}

Example Response Body	{
	"success": true,
	"data": [
	"Berufsbildende Schule",
	"Förderschule",
	"Grundschule",
	"Gymnasium",
	"Oberschule",
	"Schule des zweiten Bildungsweges",
	"Sonstige Einrichtung"
	]
	}

7.8. Get a List of Facilities

Description: This endpoint is used to access the list of facilities based on the filters provided in the query parameters. The returned list of facilities only contains geodetic data to plot the markers in the map.

URL	/api/v1/facilities
Query Parameters	“school”: boolean “kindergarten”: boolean “socialChildProject”: boolean “socialTeenagerProject”: boolean “subType”: string
HTTP Method	GET
Header	Authorization: <Token> Content-Type: application/json
Request Body	Null
Status	Success: 200 OK Error: 500 Internal Server Error

Response Body (Success)	{
	"success": true,
	"data": Object[],
	}

Response Body (Error)	{ "success": false,

	"message": string, }

7.8.1. Get a List of Kindergartens

This endpoint URL is used to get the list of kindergartens with geodetic data. Query parameter with “kindergarten” boolean as true must be passed to get the lists of kindergartens.

URL	/api/v1/facilities
Query Parameters	“kindergarten”: true
HTTP Method	GET
Header	Authorization: <Token> Content-Type: application/json
Response Status	Success: 200 OK Error: 500 Internal Server Error

Example Response Body (Success)	{
	"success": true,
	"data": [
	{
	"_id": "6647a51b944f60b2384357a4",
	"X": 12.8843699240534,
	"Y": 50.8068085244813,
	"facilityType": "Kindergarten"
	},
	{
	"_id": "6647a51b944f60b2384357a5",
	"X": 12.909709819865,
	"Y": 50.8205584525106,
	"facilityType": "Kindergarten"
	}
	]
	}

7.8.2. Get a List of Schools of the given school type(s)

This endpoint is used for retrieving the lists of schools of a given schoolType. ”schoolType” value is sent in the query parameters. Schools of multiple school types can be retrieved by passing “schoolType” in query parameters separated by “&”.
The value of schoolType can be any school subtypes retrieved from 6.7.1

URL	/api/v1/facilities
Query Parameters	“school”: true “schoolType”: “Grundschule”
HTTP Method	GET
Header	Authorization: <Token> Content-Type: application/json
Response Status	Success: 200 OK Error: 500 Internal Server Error

Success Response Body	{
	"success": true,
	"data": [
	{
	"_id": "6647a4ad944f60b238435717",
	"X": 12.8872762658422,
	"Y": 50.7922749102438,
	"facilityType": "School"
	},
	{
	"_id": "6647a4ad944f60b238435718",
	"X": 12.9218298722116,
	"Y": 50.8280539252831,
	"facilityType": "School"
	},
	]
	}

7.8.3. Get a List of Social Child Projects

Description**:** This endpoint is used for retrieving the lists of social child projects with geodetic data for plotting the markers in the map.

URL	/api/v1/facilities
Query Parameters	“socialChildProject”: true
HTTP Method	GET

Header	Authorization: <Token> Content-Type: application/json
Response Status	Success: 200 OK Error: 500 Internal Server Error

Success Response Body	{
	"success": true,
	"data": [
	{
	"_id": "6647c44b944f60b2384358a7",
	"X": 12.8850360603135,
	"Y": 50.8633389905466,
	"facilityType": "SocialChildProject"
	},
	{
	"_id": "6647c44b944f60b2384358a8",
	"X": 12.8864253328506,
	"Y": 50.8622459346951,
	"facilityType": "SocialChildProject"
	}
	]
	}

7.8.4. Get a List of Social Teenager Projects

This endpoint is used for retrieving the lists of social teenager projects with geodetic data for plotting the markers in the map.

URL	/api/v1/facilities
Query Parameters	“socialTeenagerProject”: true
HTTP Method	GET
Header	Authorization: <Token> Content-Type: application/json
Response Status	Success: 200 OK Error: 500 Internal Server Error
Success Response Body	{ "success": true, "data": [{

	"_id": "6647c7c0944f60b2384358de",
	"X": 12.8774888058499,
	"Y": 50.8310585585904,
	"facilityType": "SocialTeenagerProject"
	},
	{
	"_id": "6647c7c0944f60b2384358df",
	"X": 12.8904748101941,
	"Y": 50.809930951272,
	"facilityType": "SocialTeenagerProject"
	},
	]
	}

7.9. Get a Facility

Description: This is the endpoint for retrieving information about a facility. Facility ID has to be supplied in the URL along with the type of facility, for e.g. school, kindergarten, socialChildProject, or socialTeenagerProject in the query parameters.

URL	/api/v1/facilities/{{facilityId}}
Query Parameters	“facilityType”: “string”
HTTP Method	GET
Header	Authorization: <Token> Content-Type: application/json
Request Body	Null

Response Body	{
	"success": true,
	"data": Object
	}

Success Response

When a corresponding facility of a given facility type and ID is found, then a success message is sent along with the facility details.

Status	200 OK

Response Body	{
	"success": boolean,
	"message": string,
	"data": Object
	}

Example Response Body	{
	"success": true,
	"data": {
	"name": "Pablo-Neruda-Grundschule",
	"display_name": "Pablo-Neruda-Grundschule, Hoffmannstraße, Kaßberg, Chemnitz, Sachsen, 09112, Deutschland",
	"lat": 50.8285791667057,
	"lon": 12.8968045064718,
	"openingHours": "Mo-Fr 06:00-17:00",
	“barrierFree”: false,
	“integrative”: false,
	"wheelchair": "yes",
	"website": "https://cms.sachsen.schule/gscneruda/start/",
	"operator": "Stadt Chemnitz",
	"isFavourite": false,
	"facilityId": "6647a4ad944f60b23843572b",
	"facilityType": "school",
	"traeger": "Kommunal",
	"telephone": "0371 369670",
	"email": "gs-neruda@schulen-chemnitz.de",
	"fax": "0371 36967214",
	"address": "Hoffmannstraße 35, 9112, Chemnitz"
	}
	}

Error Response

When a facility with a given ID and facility type cannot be found in the database, an error message is sent in the response.

Status	404 Not Found

Response Body	{
	"success": boolean,
	"message": string,
	}

Example Body	{
	"success": false,
	"message": "Facility not found"
	}

When an invalid facilityType is supplied in query params, an error message is sent in the response.

Status	400 Bad Request

Response Body	{
	"success": boolean,
	"message": string,
	}

Example Body	{
	"success": false,
	"message": "Invalid facility type"
	}

7.10. Add to User Favorite

This is the endpoint for marking a facility as a favorite for a user. Facility ID along with the type of facility, for e.g. school, kindergarten, socialChildProject, or socialTeenagerProject has to be supplied in the query parameters.

URL	/api/v1/userFavourites
Query Parameters	“facilityId”: string “facilityType”: “string”
HTTP Method	PATCH
Header	Authorization: <Token> Content-Type: application/json
Request Body	{“name”: string}

Success Response

When the supplied facility id and facility type exists and are valid, the facility is marked as use favorite. A corresponding success message and data are sent in the response body.

Status	200 OK

Response Body	{
	"success": boolean,
	"message": string,
	"data": Object
	}

Example Response Body	{
	"success": true,
	"message": "User Favorite updated successfully",
	"data": {
	"facilityId": "6647a51b944f60b23843588f",
	"facilityType": "kindergarten",
	“name”: “Kirchwinkel 4a”,
	"address": "Stelzendorfer Straße 9, 9116, Chemnitz",
	"telephone": "0173 5946681",
	"email": "romy.vogel@hotmail.de"
	}
	}

Error Response

When an invalid facility type is given in the query parameters, an error message as following is sent in the response body.

Status	400 Bad Request

Response Body	{
	"success": boolean,
	"message": string,
	}

Example Body	{
	"success": false,
	"message": "Invalid facility type"
	}

7.11. Remove from User Favorite

Description: This is the endpoint to remove the user's favorite facility.

URL	/api/v1/userFavourites
HTTP Method	DELETE
Header	Authorization: <Token> Content-Type: application/json
Request Body	Null

Success Response

When the user has a facility marked as a favorite, it gets removed and a success message is sent accordingly.

Status	200 OK

Response Body	{
	"success": boolean,
	"message": string,
	}

Example Response Body	{
	"success": true,
	"message": "Facility removed from favorite"
	}

Error Response

When a user does not have any facility marked as a favorite, an error message is sent.

Status	404 Not Found

Response Body	{
	"success": boolean,
	"message": string
	}

Example Body	{
	"success": false,
	"message": "User does not have any favorite facility"
	}