Project order

Project Motorcycle App

Marc Kälin, Roger Abegg, Wiliam Isenring
FFHS

Version 0.4, 18.03.2021

Table of contents

1. Project content and definition (completed)

1.1. Management Summary

In the CAS-course "Fortgeschrittene und Serverseitige Web-Technologien" a group project has to be planned and implemented. Thereby a web application based on the MERN stack will be built. The according requirements emerge after certain PVAs of the CAS-course.
The group has been created with Marc Kälin, Wiliam Isenring and Roger Abegg as the project team. After a quick brainstorming the project idea and objective has emerged to implement a motorcycle route planner web application.
The technical requirements are based on the "PVA"s and the business requirements are defined by the project team. The project team decided in favor of the agile project management approach "SCRUM".

1.2. Project mandate / project objectives

Development and implementation of a web application based on html, css, javascript and the MERN stack.
This includes the following objectives:

- define business and technical requirements
- according to requirements develop a single page react application with persistence and REST-API interfaces
- in particular meet the business requirement of a motorbike route search

1.3. In Scope

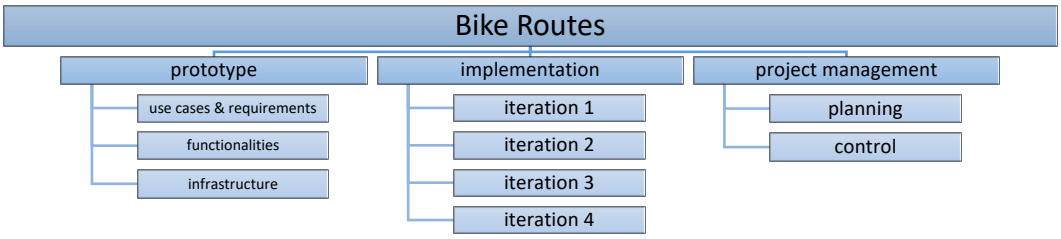
- Use of a NoSQL-Database (MongoDB) for userdata (i.e. login) and route search history
- Provision of a REST interface in order to enable communication between client and server including the use of web sockets for server push use case
- Client and server must use modern JavaScript (ES6, Server Node.js)
- React is used for the front-end
- The application is dockerized
- The application must be deployed in the cloud and must be testable
- SCRUM-project approach
- the bike route planning app is reduced to two preference dimensions: route type and location
- Register, Login, Logout functionality.
- Authentication with JWT token saved in httpOnly cookie.
- Responsive SPA

1.4. Out of Scope

- Web application made public
- Change Management (e.g. coaching, marketing) as no "real" customers are involved
- Project risks and business case
- No steering committee and official status reports
- For the sake of simplicity effort estimations for each deliverable are omitted
- Further preference dimensions such as bike type are not taken into account in the bike route planning app
- Reset password functionality.
- Further hardening measures such as JWT token storage and refresh storage (short initial httpOnly cookie expiry date and refresh token cookie in order to secure against CSRF attacks.

2. Project structure (completed)

2.1. Workbreakdown structure (WBS)



2.1.1. Prototype: Use cases & requirements

Deliverables:

1. Basic use cases are elaborated
2. Flow chart for each use case is elaborated
3. Technical requirements are deducted

2.1.2. Prototype: Functionalities

Deliverables:

1. Wireframe is defined with Invision
2. Mockup is defined based on simple html and css

2.1.3. Prototype: Infrastructure

Deliverables:

1. Technology stack is defined (e.g. sequence diagram, context diagram)
2. Infrastructure has been provided (e.g. git repository, mongo db,…)
3. access management has been elaborated

2.1.4. Implementation - iteration 1 to 4

Deliverables:

1. Requirements are understood and corresponding tasks are created in OneNote (Analysis)
2. Tasks are done e.g. implemented resp. programmed (Construction)
3. Design is adapted and refined (Design)
4. Features are tested (Unit testing with Jest and Browsertesting with Selenium) and reviewed regarding match with requirements (Testing)

2.1.5. Project management - planning

Deliverables:

1. Project objectives (objectives, scope, dependencies) are defined
2. Project organization is defined
3. Workbreakdown-structure is defined
4. Deliverables and work packages are assigned
5. Project plan is defined

2.1.6. Project management - control

Deliverables:

1. Project communication plan is elaborated and followed
2. Project status is continuously monitored. In case of deviations from the plan, measures are elaborated and implemented

2.2. Project timeplan

The project management approach is based on the SCRUM-method. The iterations are oriented on the FFHS-project tasks (P), which are given in the corresponding PVAs. Each iteration consists of the following tasks: (1) Requirements analysis and defer tasks, (2) Implement tasks, (3) Adapt and refine design, (4) Test implemented tasks and review whether requirements are fully met.

	% completed	Jan 21	Feb 21	Mar 21	Apr 21	May 21	Jun 21	Jul 21
Prototyping	0%							
Use cases + requirements	100%							
Functionalities	100%							
Infrastructure	80%							
Implementation	0%							
Iteration P2	0%							
Iteration P3	0%							
Iteration P4	0%							
Iteration P5	0%							
Iteration P6	0%							
Project management	0%							
Planning	80%							
Control	15%							

PVA12 PVA34 PVA56 PVA78 PVA910

2.3. Project communication plan

Meeting	Participants	Frequency	Meeting mode	Agenda
"Steering committee"	Orla Greevy, Marc Kälin, Roger Abegg, Wiliam Isenring	monthly	Feedback via moodle/mail	PVA requirements review and lessons learned

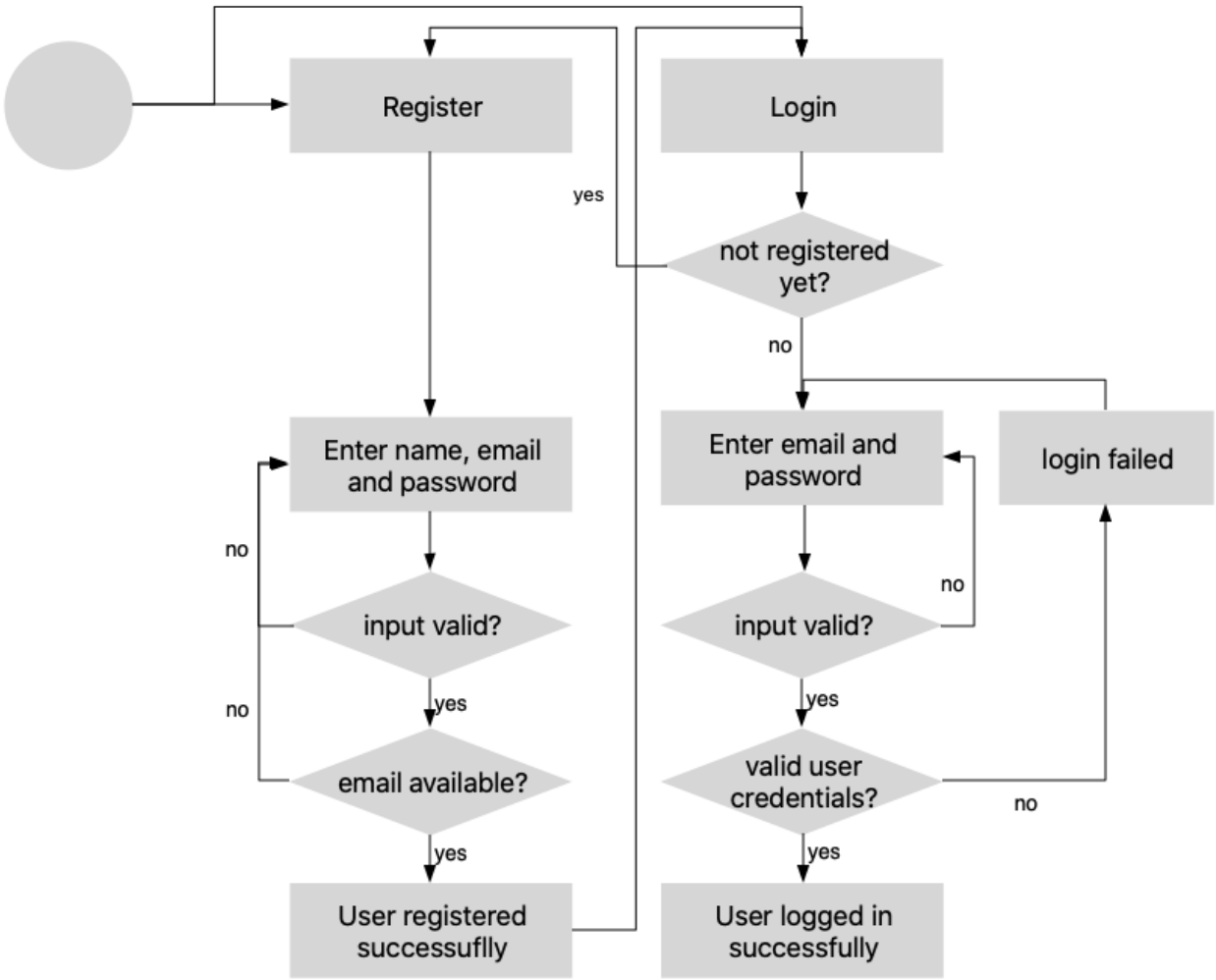
Scrum meetings	Marc Kälin, Wiliam Isenring, Roger Abegg	weekly	Skype	Completed tasks Open tasks Problems / questions
Individual team communication	Wiliam Isenring, Roger Abegg, Marc Kälin	individually	Whatsapp	Problems / questions / dependencies

3. Project documentation (ongoing)

3.1. Use Cases, flow charts and requirements

- Register / Login / Logout
- Profile
- Route search
- Contact Form

3.1.1. Register / Login / Logout (completed)



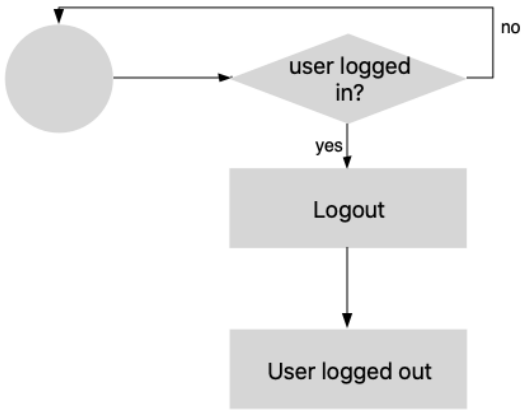
Requirements registration:

- Click on Login button and then Register here (Login component) or Register here (home) leads to Register component
- Registration form requires username, email address and password
- Sign up button in order to complete registration process
- Form input is validated on the client-side (JOI)
- Requested email address is validated against database. If there is no entry yet, user is created in database and user receives success message and gets redirected to Login component

Requirements login:

- Login button is always accessible in header unless the user is logged in
- Click on Login button or successful register leads to Login component

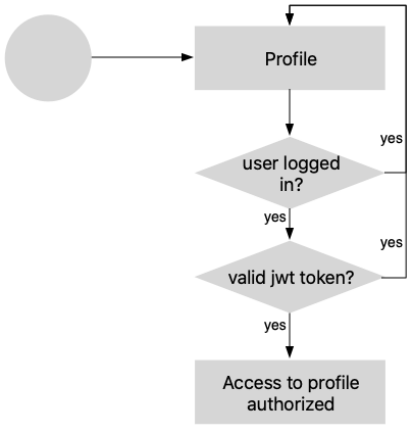
- Login request leads to server-side validation of login credentials (matches of email and hashed password)
- If the login fails error message is shown to the user in the Login component.
- If the login is successful JWT token is signed. Signed JWT token and userId are stored in separate httpOnly cookies.The user is redirected to the Profile component, the Login button in the header becomes the Logout button (with the Logout functionality) and in the header the Profile button appears.
- Note: Reset button functionality is omitted for scope reasons as pointed out in the "Out of scope" chapter



Requirements logout:

- Logout button is always accessible in the header unless the user is not logged in
- Click on Logout button
- User gets redirected to the Login component. The JWT token and userId cookies are set to value="none" and expiry date to 5 seconds. The Logout button in the header becomes the Login button (with the Login redirect) and the Profile button in the header disappears.

3.1.2. Profile (completed)



Requirements profile:

- User logs in successfully or clicks on profile icon as a logged in user
- On the server-side JWT token in httpOnly cookie is verified.
- If JWT token is valid, user information and route search history of the user are retrieved with the userId from the userId cookie.

3.1.3. Select route style

- Domain spezifische Begriffe sollen genau erklärt. Was ist eine Route? Was sind die Kategorien? Kann eine Route zu mehreren Kategorien gehören?
- User chooses route type from drop-down menu

- User chooses location from drop-down menu
- User clicks on "get route" button
- The route will be displayed to the user according to his choices in the map iframe
  Eine Route ist mit einem Gebiet verbunden.

3.1.4. Contact form

The main usage for the contact form is for the user to get in touch with us. It serves as a means for the customer to communicate with the provider (e.g. feedback, questions, problems). The project team will not focus on this feature as data persistence (see Login and Register, Search Route), React, Express and Node.js will be implemented for other features.

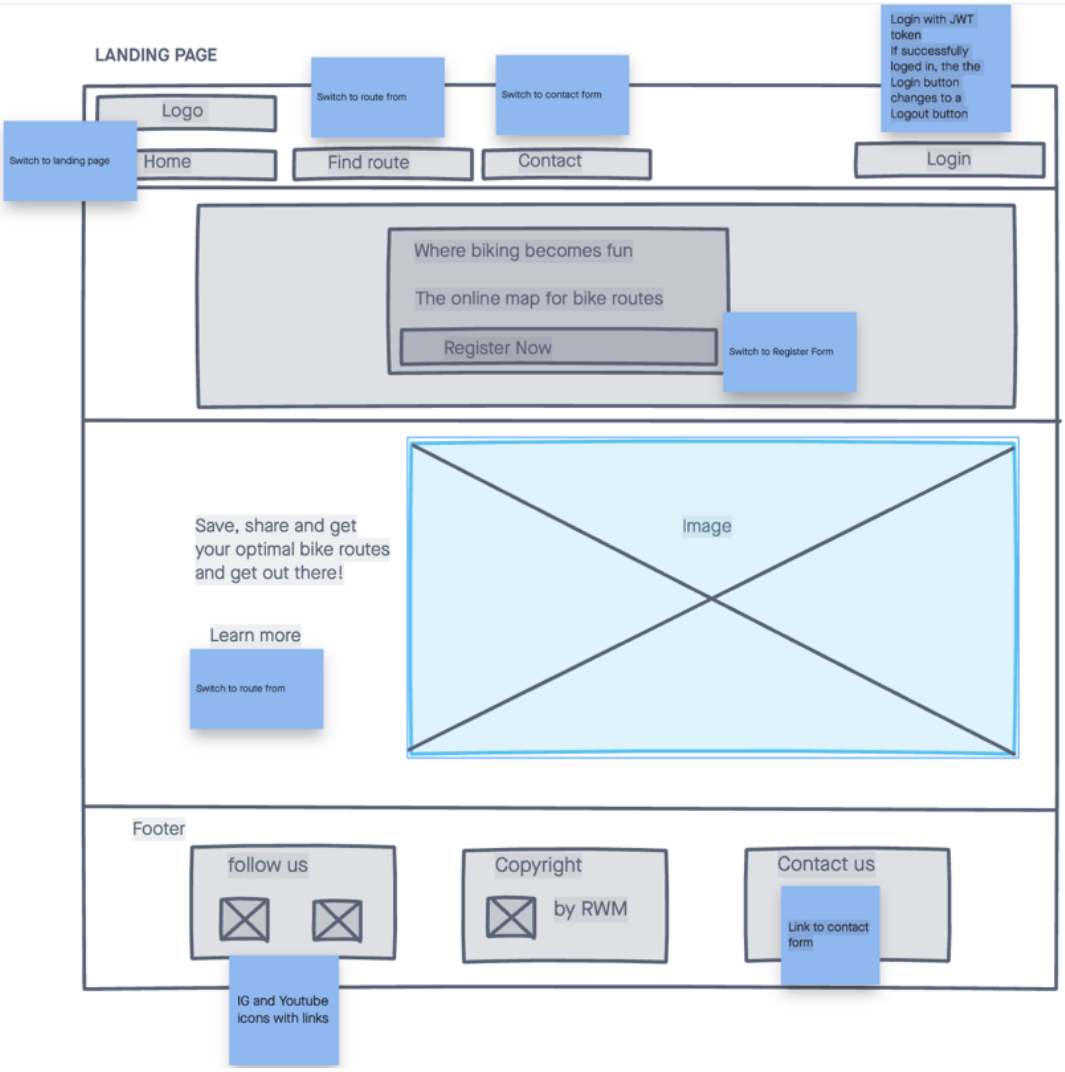
- User navigates to contact form
- User fills out required fields (name and e-mailaddress)
- User fills out required message field
- User clicks on "submit" button and receives a confirmation that his message has been sent

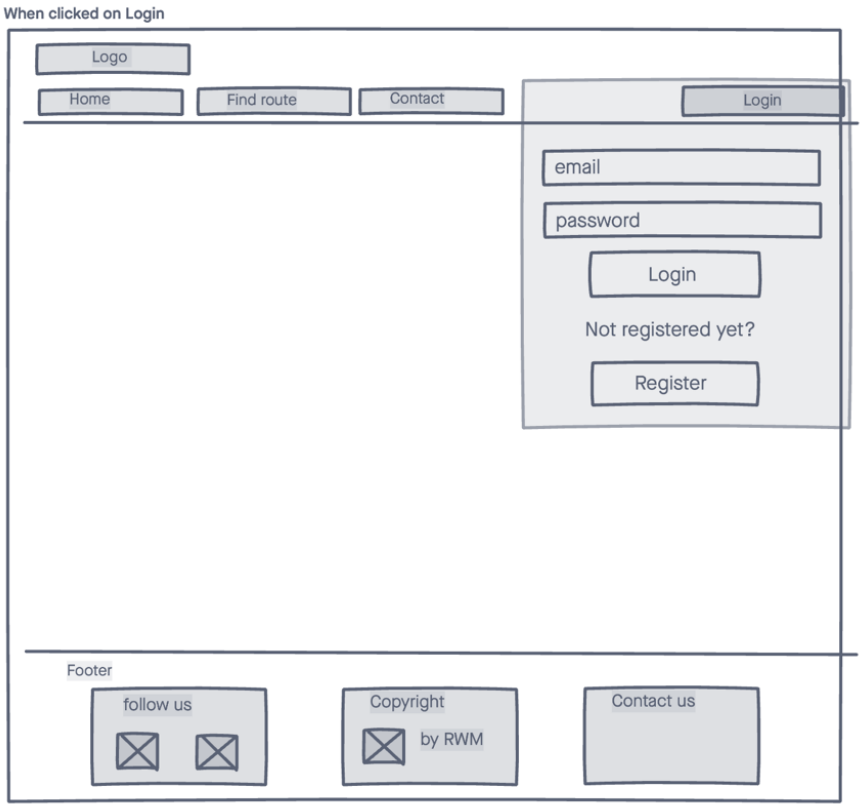
3.2. Functionalities

3.2.1. Wireframes

The single page react application consists of three pages in the frontend:

1. Landingpage


LANDING PAGE
Logo
Home
Find route
Contact
Login
Switch to route from
Switch to contact form
Login with JWT token
If successfully loged in, the the Login button changes to a Logout button
Switch to landing page
Where biking becomes fun
The online map for bike routes
Register Now
Switch to Register Form
Save, share and get your optimal bike routes and get out there!
Learn more
Switch to route from
Image
Footer
follow us
Copyright
by RWM
Contact us
Link to contact form
IG and Youtube icons with links

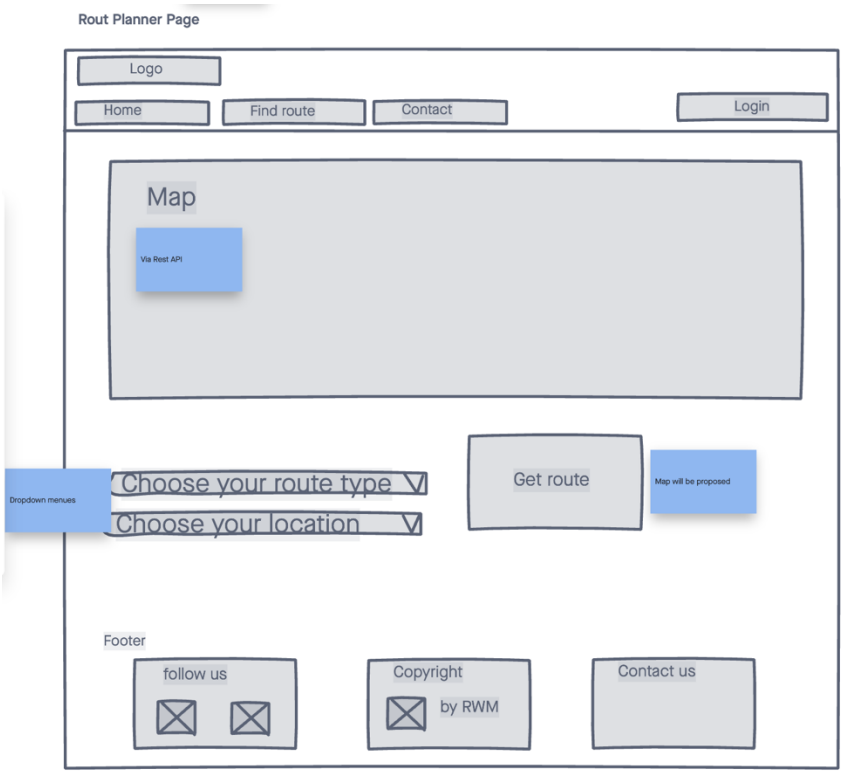


2. Routesearch

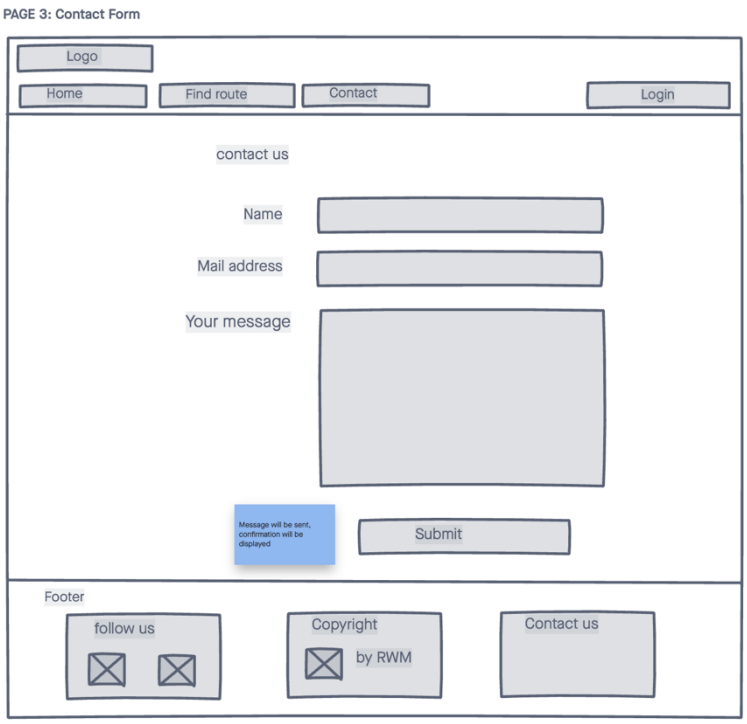
    a. 2 routetypes (curvy, scenic)

    b. 2 locations (Luzern, Graubünden)

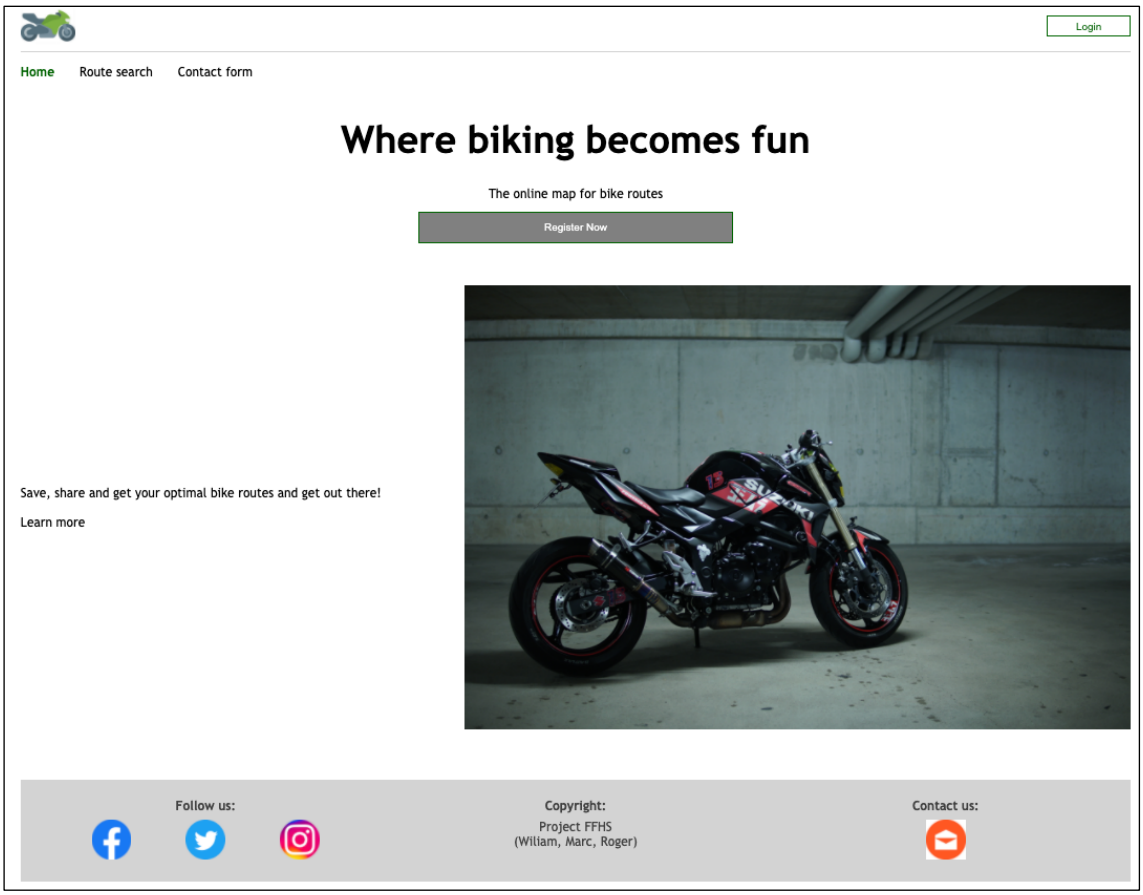
Sind das EInschränkungen des Prototyps, dass nur 2 Locations berücksichtigt werden?

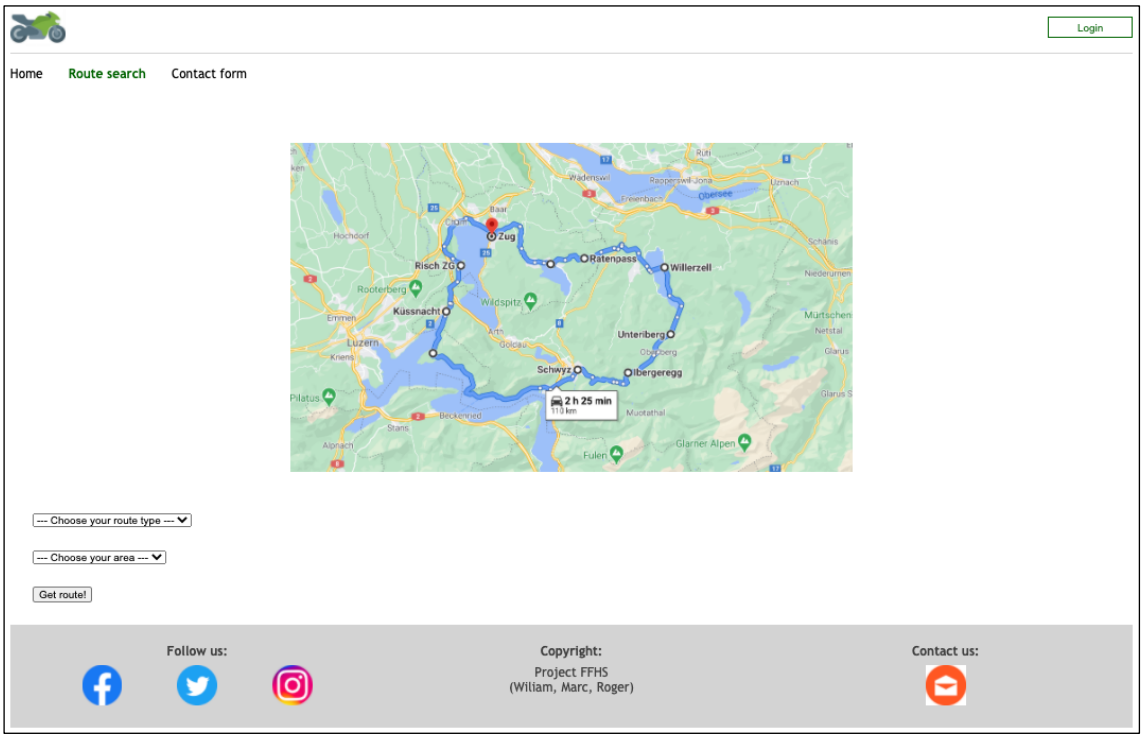


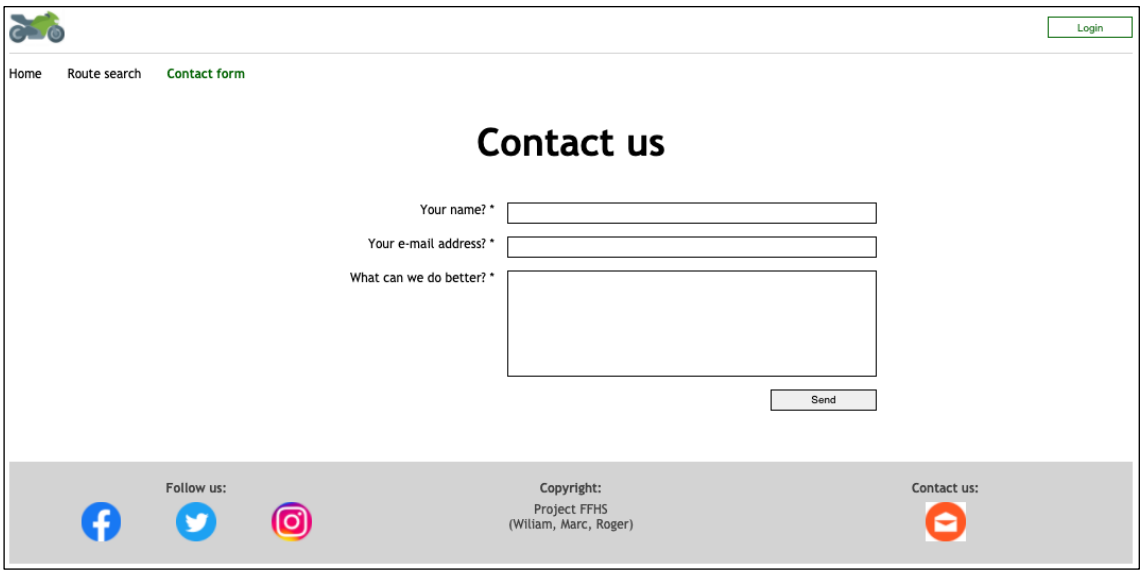
3. Contact form



3.2.2. Mockups (completed)





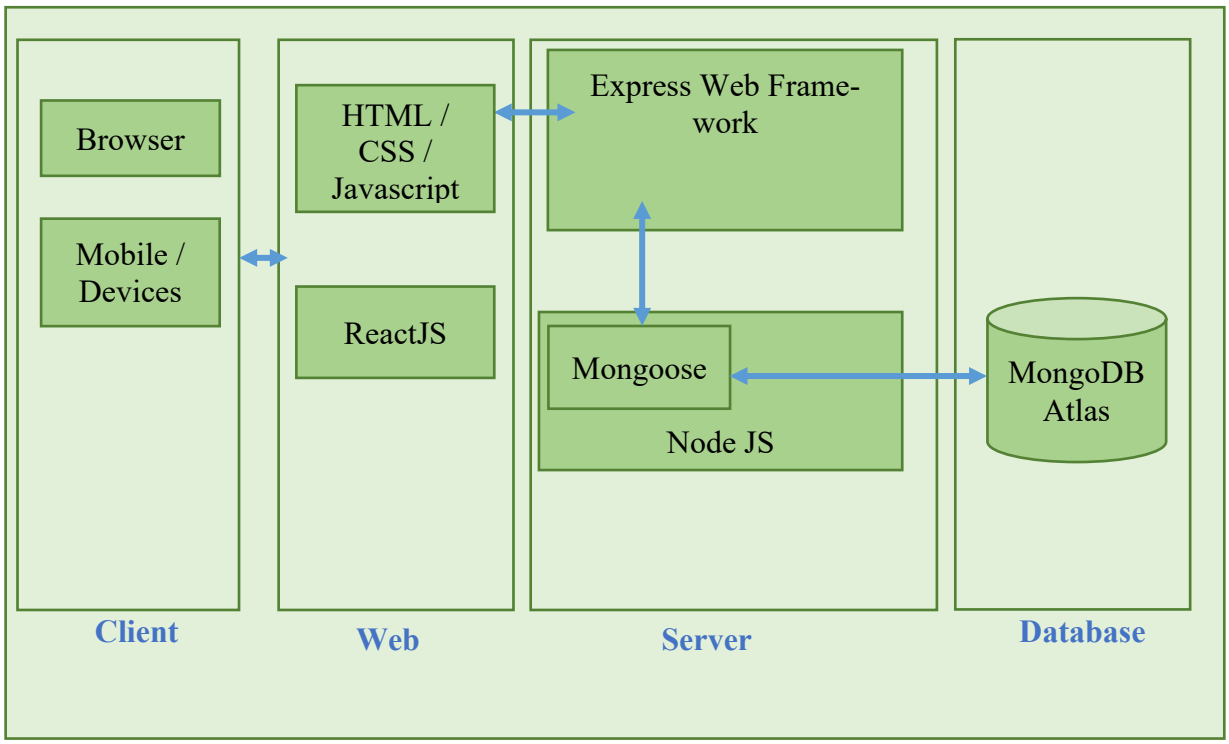


3.3. Infrastructure

3.3.1. Context diagram

Input Wiliam

3.3.2. Architecture diagram (completed)



3.3.3. Technical Requirements

The following is a list of the required libraries / packages for the application, deducted from the flow charts and use cases, scope specification and ongoing development.

Frontend

```
"dependencies": {
  "@testing-library/jest-dom": "^5.11.10",
  "@testing-library/react": "^11.2.6",
  "@testing-library/user-event": "^12.8.3",
  "cookie-parser": "^1.4.5",
  "mapbox-gl": "^2.2.0",
  "react": "^17.0.2",
  "react-dom": "^17.0.2",
  "react-router-dom": "^5.2.0",
  "react-script-tag": "^1.1.2",
  "react-scripts": "^4.0.3",
  "web-vitals": "^1.1.1",
  "worker-loader": "^3.0.8"
```

- **React v17.0.2**: open-source, component-based JavaScript library. It is used to build front-end i.e. to create dynamic elements on the website.
- **React – useContext**: useContext is used in order to define global state variables and make the components share the global states.
- **React – dom:** needed to render App.js
- **React – router – dom:** enables nested components in the react environment

Backend

```
"@hapi/joi": "^17.1.1",
"bcryptjs": "^2.4.3",
"body-parser": "^1.19.0",
"cookie-parser": "^1.4.5",
"cors": "^2.8.5",
"dotenv": "^8.2.0",
"express": "^4.17.1",
"jsonwebtoken": "^8.5.1",
"mapbox-gl": "^2.2.0",
"mongoose": "^5.12.2",
"nodemon": "^2.0.7",
```

- **@hapi/joi v17.1.1:** The node.js Package is used for validation of form inputs.
- **bcryptjs v2.4.3:** As described in 3.1.1 and 3.1.2 the web app will feature a register and login feature. In the registration process the user defines a password, which is then stored in the database. In the login process the user's password input needs to be compared with the one in the database (if there is a match the jwt token will be generated for the user). The bcryptjs library allows encrypting the password i.e. hashing the user's password. Admins will not be able to access user's account
- **body-parser v1.19.0:** The node.js middleware will be used in order to parse the body data i.e. read the request body as a JSON object (see index.js).
- **cookie-parser 1.4.4:** Once the JWT token is stored in a secure and httpOnly cookie, the cookie-parser middleware helps to parse cookies on incoming requests. In order it allows to read the cookie value for further services (such as granting access to the profile page and providing the user's history and data).
- **cors v2.8.5:** The Node.js package CORS (cross-origin resource sharing) is used for providing a Connect/Express middleware and to enable CORS with various options. It generally defines the access to resources from another origin (domain, protocol, port). E.g. with regards to the authentication process (storage of JWT token in httpOnly cookie) in this group-project, the front-end and back-end need to exchange information which is enabled by defining the origin option of the cors component.
- **dotenv v8.2.0:** The web app of this project will use various data that should not be visible to anyone but the developer. Secrets such as JWT secret and MongoDB connection string can be stored in a .env file. The zero-dependency module dotenv allows to load these secrets as environment variables from the .env file into process.env. It allows to separate secrets such as JWT secret and absolute connection URL to MongoDB such that this information is not shared with other people.
- **express v4.17.1**: Express is the Node web framework. The web app of this project will make use of express in order to (1) start the web server, (2) define the port to use for connecting , (3) register middleware and controllers and (4) assign request handlers to client requests.
- **jsonwebtoken v8.5.1**: While bcryptjs serves in the login and register process as the means to hash any sensitive data such as the user password, the jsonwebtoken package makes use of the JSON Web Token in the same processes. The package allows to encrypt the data payload on registration and return a token. A successful login grants a jwt token with which the logged in user can access protected content (sign jwt token). The verify method allows to check for a valid jwt token. Additionally, it can be assigned an expiration time (token expiration)
- **mapbox-gl: The mapbox-gl library is used for rendering the map component and gaining access to all the methods contained within the Map object – such as event listeners, adding map controls, adding layers on the map etc.**
- **mongoose 5.12.2**: As the app will have certain persistent elements such as the user account information and the route search history, MongoDB will be used as the database. The object data modeling library mongoose allows the translation between coded data (objects) and its representation in the database MongoDB, connects the backend with the database and has a schema validation
- **nodemon v2.0.7**: Nodemon helps developing by restarting the server whenever changes have been made to the source.
- **npm**: pre-installed package manager for the Node.js server platform. Responsible for the organization of installation and management of third-party Node.js programs (i.e. programs from the npm registry)

Database

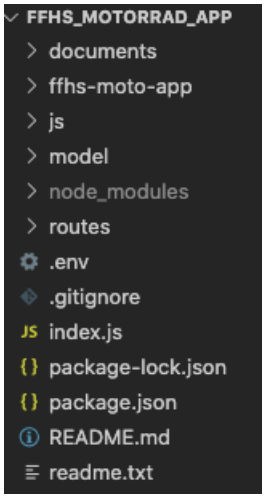
- **MongoDB atlas:** As the app will have certain persistent elements such as the user account information and the route search history, MongoDB atlas will be used as the cloud database.

API

For fetching the data for the routes, API-calls to Mapbox will be made. The responses will then be used to display the route coordinates on a map, which will be presented to the user.

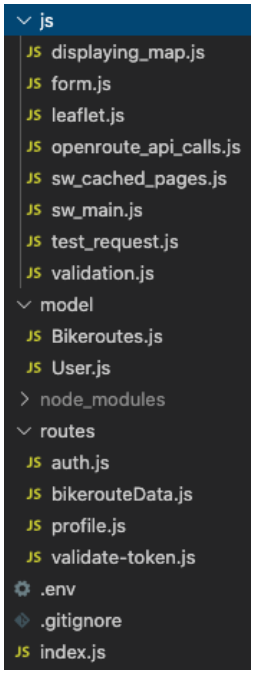
3.3.4. Project structure

Overview



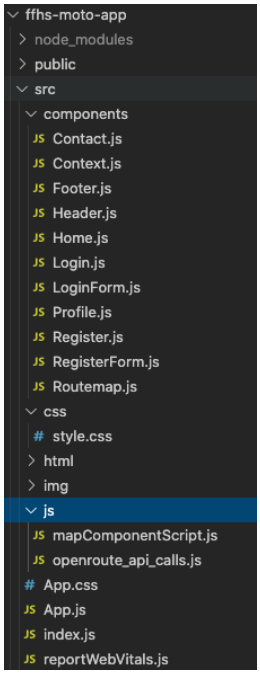
documents: folder entails a pdf version of this project documentation
ffhs-moto-app: react codes (components, css, JS, App.js, index.js)
js: includes all JavaScript codes on the server-side
model: mongoose models
node-modules: contains all the npm packages necessary to run the web application (see also technical requirements)
routes: entails the route middlewares
.env: Sensitive values are stored as environmental variables in .env and loaded with dotenv.
.gitignore: where files or folders are specified that are not tracked by git.
index.js: Middleware, which imports and initializes the necessary modules, defines the connection when running the web server, steers the routes and request handlers according to the client request

Server



js_openroute_api_calls.js: *This is the original file which i used to write the scripts for fetching data from the API and then sending it to our backend. I have then created a component and used the scripts from this file.*
js_sw_cached_pages.js: *This is the serviceworker that caches the pages and makes it available even when offline*
js_validation.js: *Uses the JOI HAPI package in order to validate register form inputs (name, email, password against min, max, required and regex)*
model_Bikeroutes.js: user schema for mongoose (coordinates, waypoints, date). It also creates the collection on the MongoDB, which is used to store the data.
model_User.js: user schema for mongoose (name, email, password, date, history). It maps with the MongoDB Atlas collection "users"
routes_auth.js: POST-routes for register and login, GET-route for logout (react components Login, LoginForm, Register, RegisterForm, Header). E.g. it validates the input, hashes sensitive data (here password), checks for duplicate email (input versus MongoDB), creates the user object for further handling, executes modifications or lookups in the MongoDB and creates a JWT token for the access of protected content
routes_bikerouteData.js: The routes aswell as the handlers are defined for requests made to the corresponding route.
routes_profile.js: GET-route for profile (Profile)
routes_validate-token.js: Middleware which is used by index.js in order to verify the client's token for validity and grant access to protected routes (e.g. myProfile)
index.js: imports Express.js, creates an instance of an Express application and starts it as Express server on PORT 3000. Establishes the connection to the MongoDB Atlas, sets the CORS options and defines the routing of the middlewares as described above (routes_x)

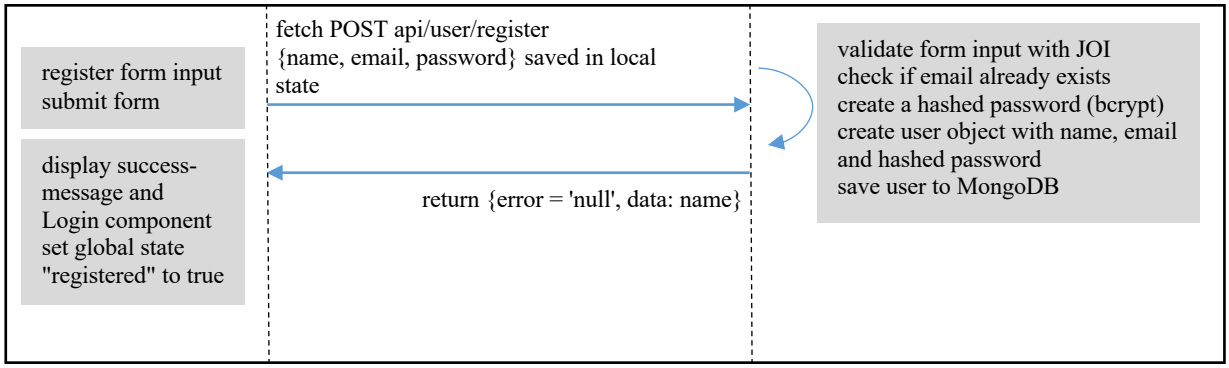
Front-End



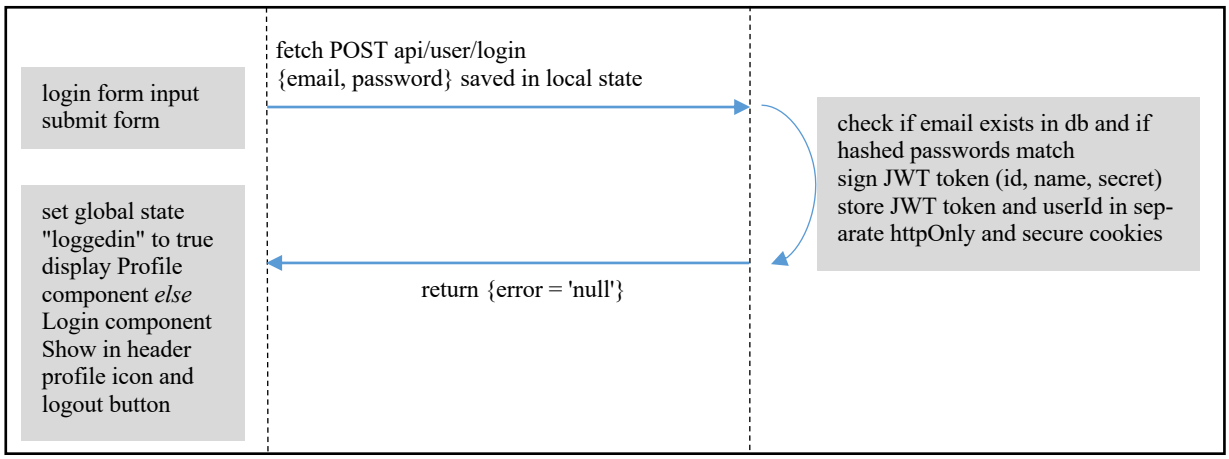
public: can be deleted?
src_comp_Contact.js: React component for the contact application. Fetches a POST request to the contact.js in the backend.
src_comp_Context.js: React component in order to use "useContext" from react. Allows to use and share states between the React components.
src_comp_Footer.js: React component for the footer application
src_comp_Header.js: React component for the header application. Includes a responsive navigation bar (burger-menu) and a login/logout button which reacts to the state loggedin. In order to logout fetches a GET request to the auth.js in the backend
src_comp_Home.js: React component for the home application.
src_comp_Login.js: React component for the login application. Fetches a POST request to the auth.js in the backend (Login)
src_comp_LoginForm.js: Includes login form and executes login form in Login.js when submit button is clicked with form input as body.
src_comp_Profile.js: React component for the profile application. Fetches a GET request on the profile.js route, which is protected by JWT validation with validate-token.js. Useeffect is used to execute the fetch call. Renders response for user information and search history.
src_comp_Register.js: React component for the register application. Fetches a POST request to the auth.js in the backend (Register)
src_comp_RegisterForm.js: Includes register form and executes register form in Register.js when submit button is clicked with form input as body.
src_comp_Routemap.js: This is the component that renders the map
css_style.css: All css styles for the react components. Based on mobile first-approach.
html: can be deleted?
img: All images used by the react components
App.js: Main react-component. Defines react-front-end-routes using react-router-dom. Defines all global states using create-context from react.
index.js: Renders App.js in root.

3.4. Register, Login, Logout, Profile

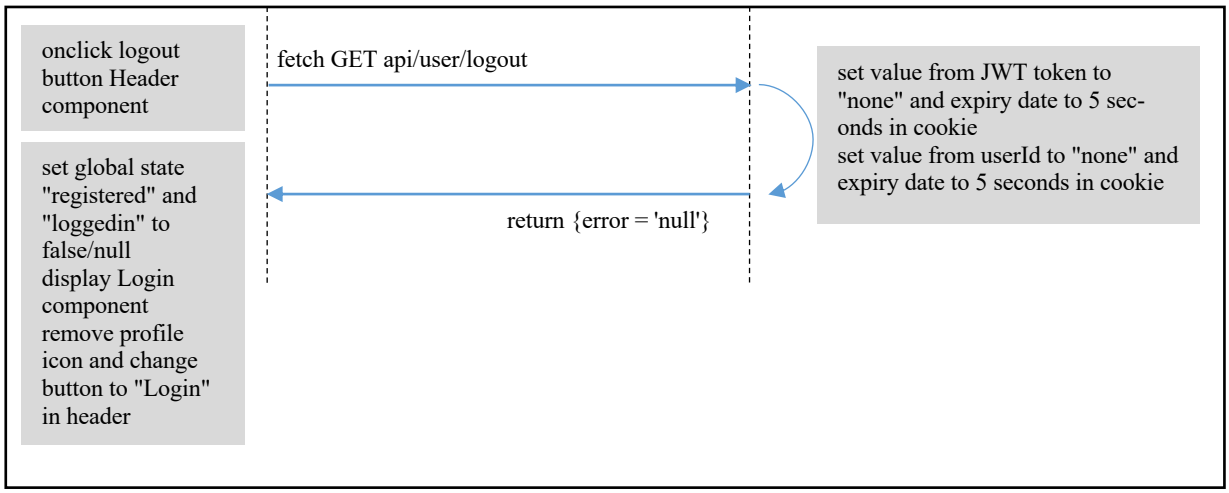
3.4.1. Register



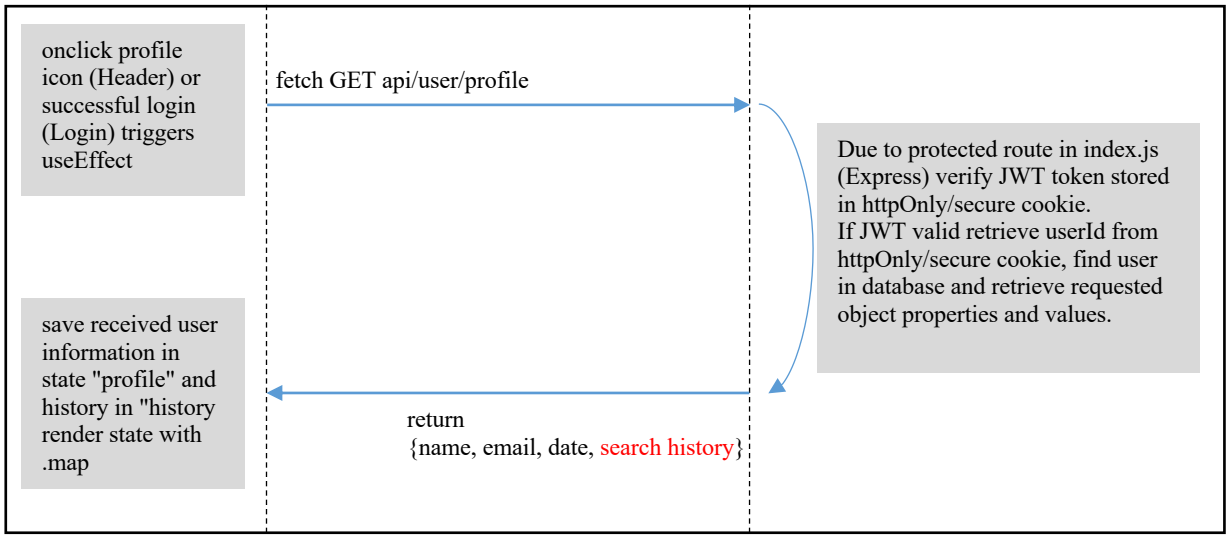
3.4.2. Login



3.4.3. Logout



3.4.4. Profile



3.4.5. Security aspects

Passwords: All passwords are hashed before saving or validating them in the backend with the database.

Sensitive backend-information: Sensitive and crucial security information such as the db-connect URL and JWT token secret are stored in an environment variable in .env.

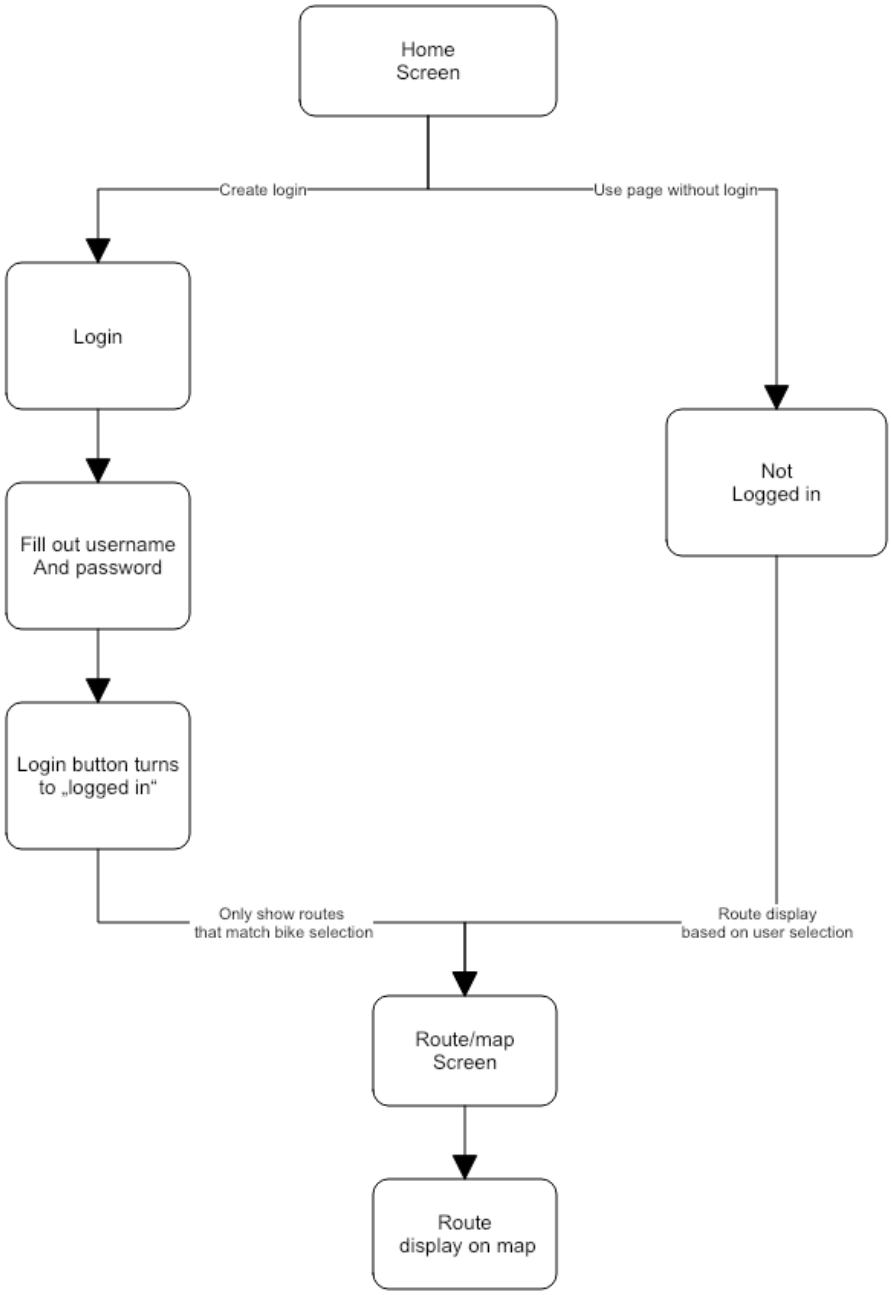
JWT token: Once the user has successfully logged in, the jwt token needs to be stored in order to access protected content (in this case the profile page with the user's history and data) throughout the session. According to the web there are two possible options for the storage: localStorage or cookies. While the localStorage option is easier to implement, it is vulnerable to XSS attacks. The locally stored JWT token is accessible via JavaScript and hence can be accessed by an attacker. Therefore, the application of the project uses the cookie option which is more secure regarding XSS attacks.
By using httpOnly and secure cookies, the cookies are not accessible via JavaScript (e.g. by using document.cookie). However, they can still be sent back to the server in http requests. Hence the implemented JWT storage in cookies is still vulnerable to CSRF attacks. Due to the scope of this project no further hardening regarding security have been done. A possible measure would be to create one initial JWT token with an expiry date of 15 minutes and a refresh token which verifies the initial JWT token and signs a new JWT token. Additionally, all expired JWT token could then be blacklisted.

CORS: As the cookie is created in the backend and is not visible in the frontend due the cookie option httpOnly set to true, CORS needed to be set up accordingly (needed origins in origin and credentials = true). On the frontend, headers need to be send with the requests to the backend. Hence, the fetch calls need to be set to "include" in the credentials option.

Sensitive user-information: When the frontend tries to access sensitive user information to display, the call is protected by a JWT token validation.

3.5. Bike route planner

Screenflow



Swimlane diagram