IT UNIVERSITY OF COPENHAGEN

Research Project

Implementation of the Progressive Web App - Woodle

Author:
Vilfred Sikker Dreijer
(vidr@itu.dk)

Supervisors:
Fabricio Batista Narcizo

UNIVERSITY
IT University of Copenhagen

Website:
`www.sikkersoftware.dk`

Github:
`https://github.com/VilfredSikker/woodle`

May 2020

Contents

1 Introduction

This project aims to create a native-like application on the web to test the current capabilities of Progressive Web Apps[3.1].

Woodle is an application where users can register an account and track the activities through GPS location. After completing an activity, it's saved so the user can keep track of activity stats and previous activities. Users can add friends and see their activity history as well.

2 User Guide

This section covers the main functionalities of the app on a non-technical level. It presents the features and the design choices behind.

2.1 Map

Clicking the 'MAP' button in the top center shown in Figure 1 shows the map. The map displays the user's current location, through GPS tracking, in the form of an orange circle.

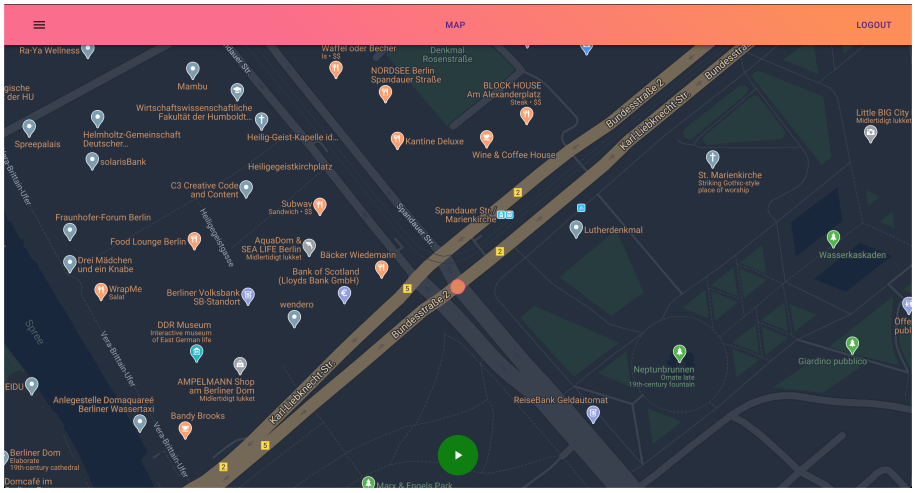

Figure 1: Map Start

To start an activity, the user presses the green start button located at the bottom-center of the map. While active, the user creates a red path from its movement (See figure 2). Click the red button, located in the bottom-center, to stop an active activity. Navigating to different views, while an activity is active deletes the activity (See section 4). Woodle doesn't track the user when the app is closed (See Section 4.1), though it continues an active activity when re-opened.

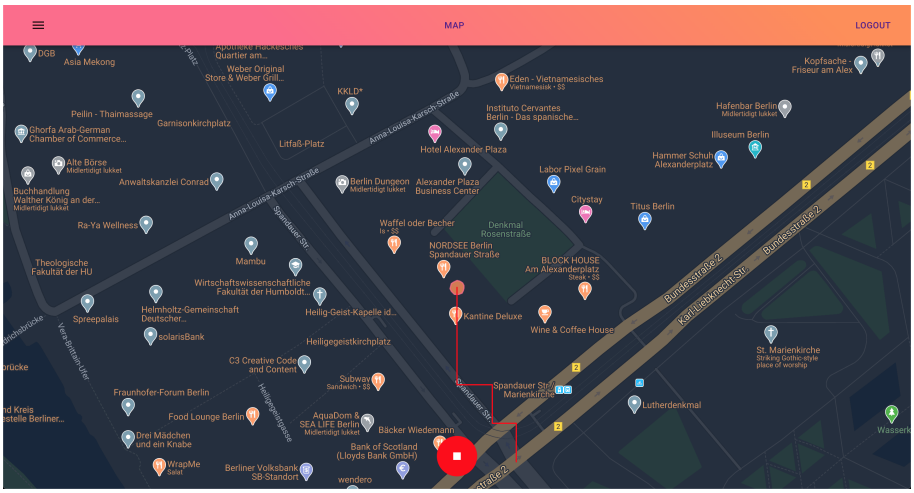

Figure 2: Map Stop

The map supports panning and zooming through a native-like touch movement.

2.2 Profile

Figure 3 shows a user's profile and consists of a tabbar with the tabs: *Stats*, *Activities*, *Friends*, and *All Users*.

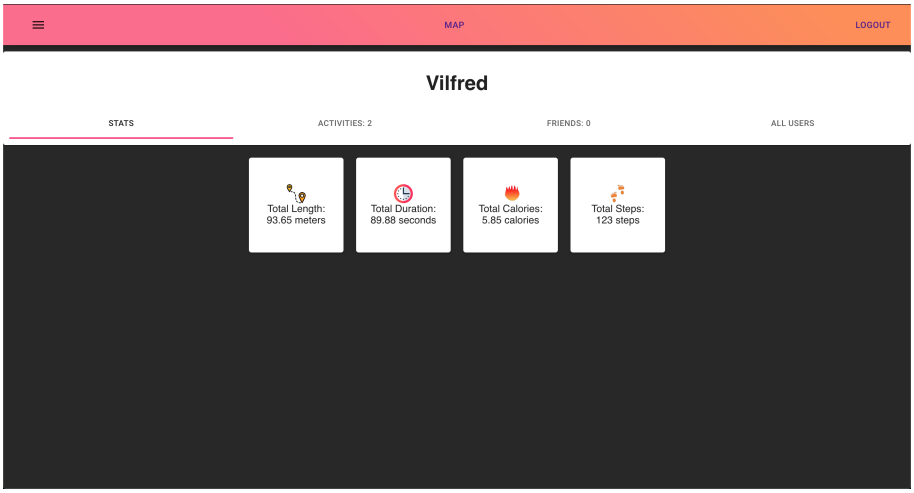

Figure 3: Profile

Stats shows the total length, total duration, total calories, and total steps, accumulated overall activities.

Figure 4 shows *Activities* which is a list of previous activities. The name of an activity is given by the date and time of creation. Each activity can expand by clicking on the name. Expanded activities show the details of the activity as well as an interactive map with the path.

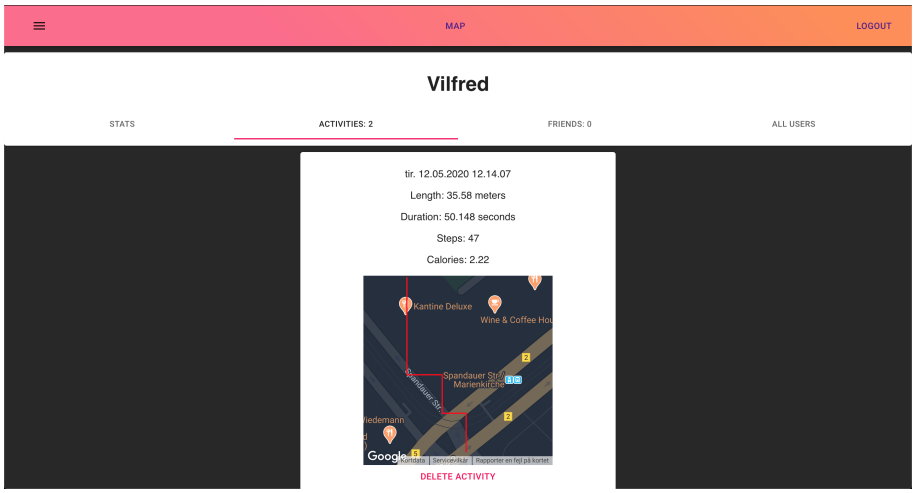

Figure 4: Activities

Figure 5 shows *Friends* which is a list of friends. A friend can be added by navigating to the *All Users* (see Figure 6) tab and pressing ADD USER. To see a friend's activities, the user can click SEE ACTIVITIES. A user can delete a friend by clicking REMOVE FRIEND

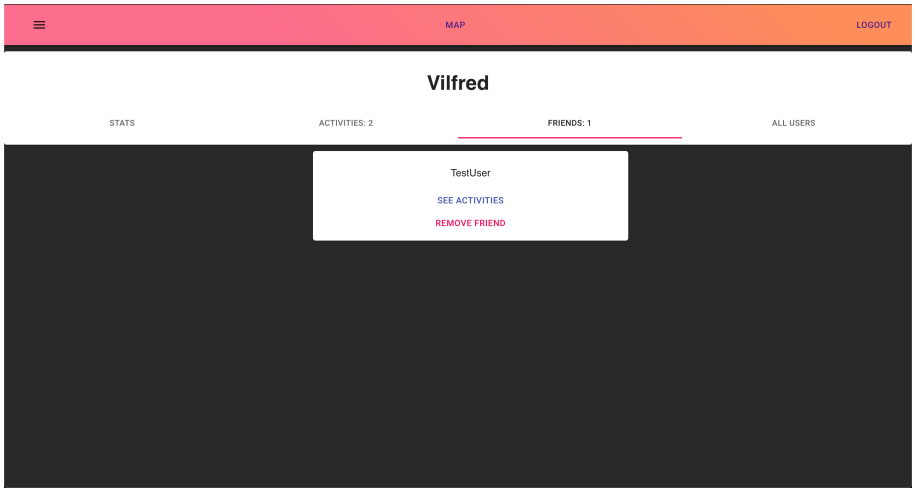

Figure 5: Friends

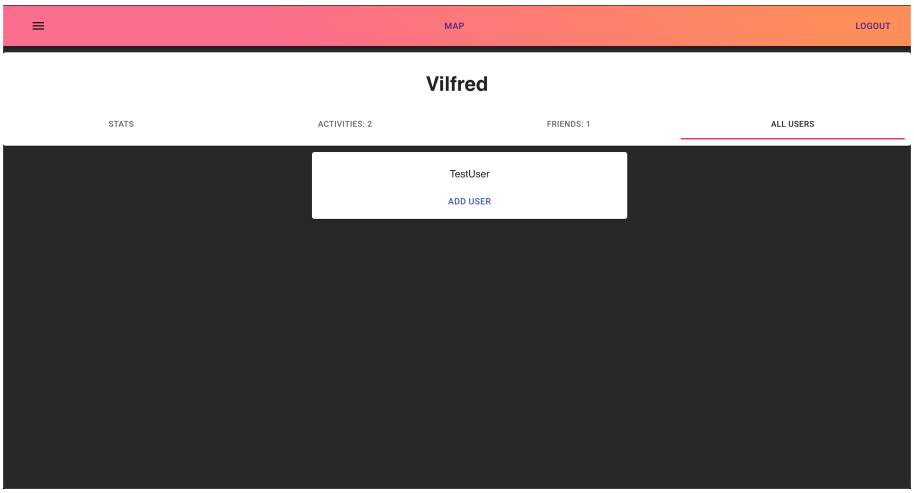

Figure 6: All Users

2.3 Registration - Sign Up

A user can register through the Sign Up page, shown in Figure 7.

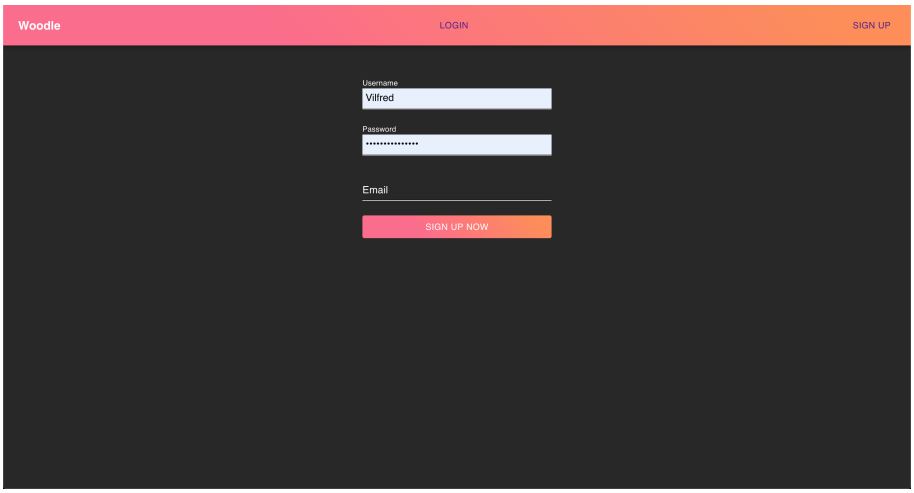

Figure 7: Sign Up

Sign up requires a valid email, a username with at least 4 characters and a password with at least 8 characters.

Clicking SIGN UP NOW sends a confirmation message to the email and redirects the user to the confirmation page (See Figure 8). The confirmation page requires a username input and a confirmation code.

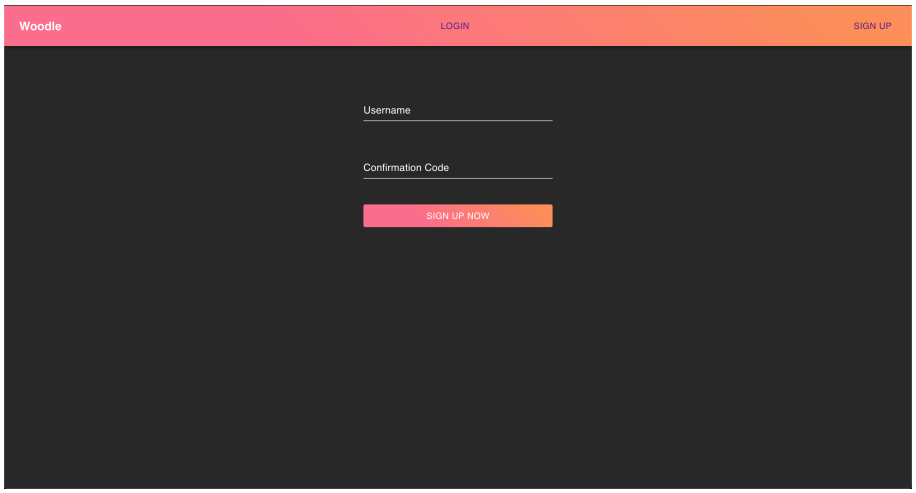

Figure 8: Confirm Sign Up

3 Technical Description

This section covers the technical details of the application. They are ranging from programming languages to permissions, data persistence, and security.

3.1 Progressive Web App

Woodle is a Progressive Web App (PWA)[1], which is an application run on the web, but with enhancements to create a native-like application. Progressive Web Apps bring native-like capabilities, such as geolocation, push-notifications, offline use, and more to web-platforms.

Some of these native-like functionalities, such as offline use, are possible because of service-workers. Service workers are scripts running in the background and act as a network proxy between the application (See figure 9).

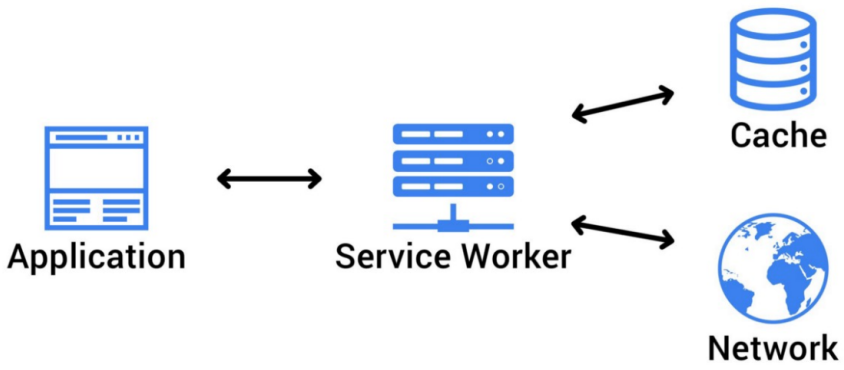

Figure 9: Service Workers

Service workers can intercept network requests, cache files, push notifications, and access the cache. It is required to run the application on https for service workers to function.

Woodle is hosted on https, has a manifest.json (a JSON file describing the app name and icons) file and service workers, and therefore meets the chrome criterias[2] to be installable.

3.2 Amazon Web Services

Woodle uses a variety of services from AWS to host the website, database storage, store users, and manage security and permissions.

[1] What are Progressive Web Apps, web.dev, `https://web.dev/what-are-pwas/`
[2] What does it take to be installable, web.dev, `https://web.dev/install-criteria/`

3.2.1 Route 53, CloudFront & S3 Buckets

Amazon Cloudfront is a Content delivery network[3] service that delivers data, applications, and API's securely to users.

Amazon S3 buckets are storage containers hosted by Amazon. They can be used to store any data, ranging from website to server backups. Woodle uses S3 buckets to store website build bundles.

Amazon Route 53 is Amazon's Domain Name System (DNS)[4] and connects the browser to the rest of Amazon Web Services.

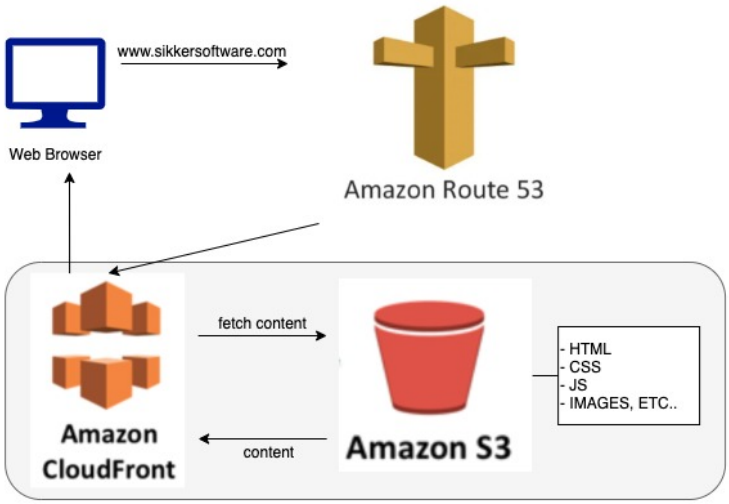

Figure 10: Web flow

Figure 10 shows the flow of a browser entering www.sikkersoftware.dk. Amazon Route 53, receives the URL and connects it to the Amazon CloudFront service. Amazon CloudFront receives the content from S3 bucket and delivers it to the user.

3.2.2 Data Persistence

Woodle uses Amazon DynamoDB databases to persist data. Amazon DynamoDB is a NoSQL database that supports key-value and document data structures. DynamoDB table items have attributes, which means that we can avoid null values because we choose what attributes an item has. In contrast to SQL, tables have defined columns they have to fill. Amazon DynamoDB tables are required to have a primary key, and each item has to contain this.

As seen in Figure 11, Woodle has four DynamoDB tables: User-, Friend-, FriendConnector- and Activity-table.

[3]CDN, Wikipedia, `https://en.wikipedia.org/wiki/Content_delivery_network`
[4]Amazon Route 53, aws.amazon, `https://aws.amazon.com/route53/`

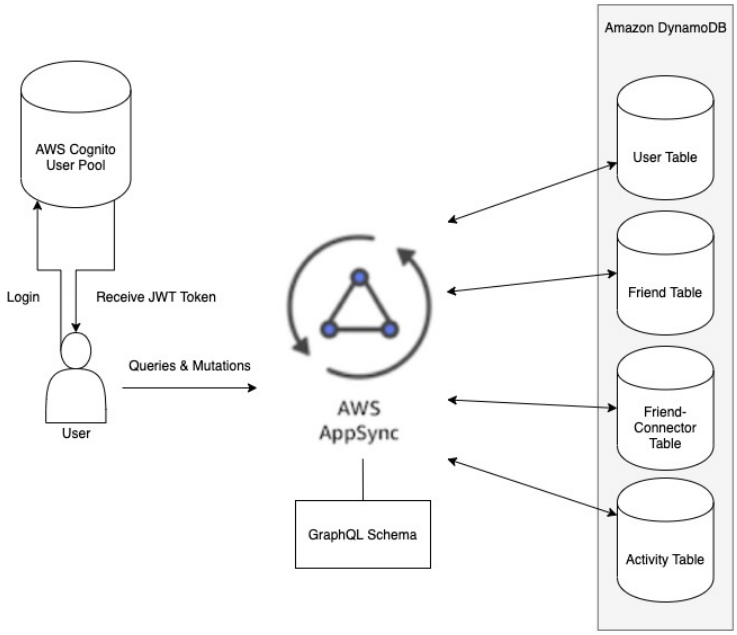

Figure 11: Client Flow

The User table contains an *ID*, a *username*, a *friendConnector*, and a list of *activities*.

The Friend table contains an *ID*, a *friendName*, a *friendConnector*, and a list of *activities*. The Friend model is basically a reference to a user.

The FriendConnector table contains an *ID*, a *friendID*, a *connectorID*, a *connector* (the user), and a *friend*.

The Activity table contains an *ID*, a *userID*, a *name*, a *duration* in seconds, a *length* in meters, a number of *calories*, a number of *steps*, and a list of *coordinates*. The coordinate is of type:

```
type Coordinate {
  lat: Float
  lng: Float
}
```

User registration creates a user and a friend with the same id and username. The user and friend model both have a 1-to-many relationship with the Friend-Connector model, which results in a many-to-many relationship between User and Friend. Every DynamoDB table is queried and mutated through GraphQL (See Section 3.3).

3.2.3 Security and permissions

All users register through Amazon's Cognito User Pool system. A register requires a username, a valid email, and a password and has to be confirmed with the confirmation code sent to the email (See Section 2.3). Each user is stored safely, with every information inaccessible from the command tools, as seen in Figure 12.

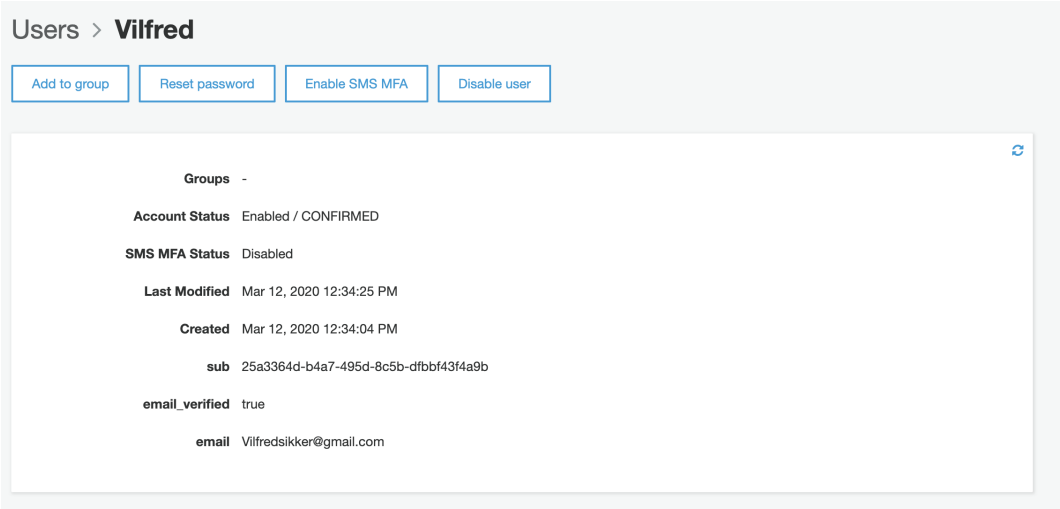

Figure 12: User information

On login, a user receives a JWT token, which is valid for the current session an used to query and mutate the DynamoDB tables, as explained in Figure 11.

3.2.4 AWS Amplify

AWS Amplify[5] is the development platform used to build and access the AWS Cognito User Pool, Authentication, and API. The Amplify Command Line Interface creates and manages the AWS services as well as providing auto-generated code that can be modified.

3.3 GraphQL

GraphQL is a query language for APIs. It pairs well with document-structured databases such as Amazon DynamoDB. GraphQL is designed around "Ask for what you need and get exactly that", which means that you can get more resources in a single request. REST APIs often require multiple requests to different URLs to obtain the same amount of resources.

[5] AWS Amplify, aws.amazon, `https://aws.amazon.com/amplify/`

Schema.graphql defines and creates the four types: User, Friend, FriendConnector, and Activity. Each type can have the following annotations:
@model: converts the type to a model, which creates a table for the type. Activity has the @model annotation, and the Coordinate type does not. Coordinates is a simple type and used within models, e.g., an Activity has a path variable, which is an array of Coordinates.
@key: gives the model a key, which is defined by a name and the unique combination of fields.
@connection: creates a relation between models. Connections are made to keys in other models. The User has a connection to Activity through the "by-Activity" key.

3.3.1 Permissions & Authentication

The *schema.graphql* file also declares the query and mutation permissions. The @auth annotation describes the authentication rules for a model. The User model has create, update, delete, and read permissions by the owner. The owner refers to the Cognito User (see Section 3.2.3), and a User can be read by private, which is anyone within the User Cognito Pool. Permissions can also be public, which everyone can access. This is not used.

3.3.2 Queries and Mutations

Queries (read data), as mentioned, are designed to return "what is asked for" in a JSON format, which makes it easy to handle in our client.

Mutations requires specific input. As an example, *createUser* (see Section 5.2.2), requires a username, but ID is optional. If the ID isn't given, a random ID is generated.

3.4 React

This section covers the language and tools used to build the client.

Woodle uses React and Typescript. React is a component-based, Javascript library/framework, which makes state handling and passing of data through the application easier. Typescript extends Javascript with type-based variables, which makes the code more reliable.

3.4.1 Lifecycle

After React 16.8, there are two ways to use lifecycles. The standard React class components which extend *React.Component* or functional components with React hooks (introduced in React 16.8). Both have three phases: mounting, updating, and unmounting. Figure 13 shows the React Class Component Lifecycle.

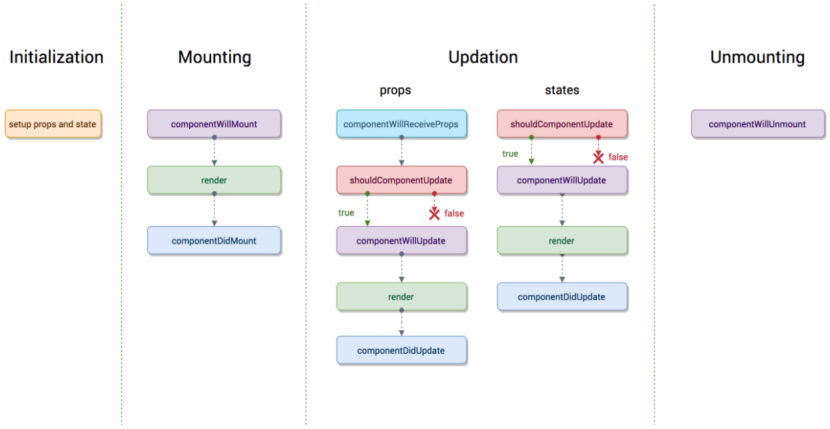

Figure 13: Component Lifecycle[6]

Woodle uses functional components, which changes the lifecycle a bit as shown in Figure 14. *useEffect* replaces *componentWillMount*, *componentDidUpdate*, and *componentWillUnmount* and we can handle state through the *useState* hook. Functional components are normal javascript functions, but with the new React hooks, we can get state and other class component features.

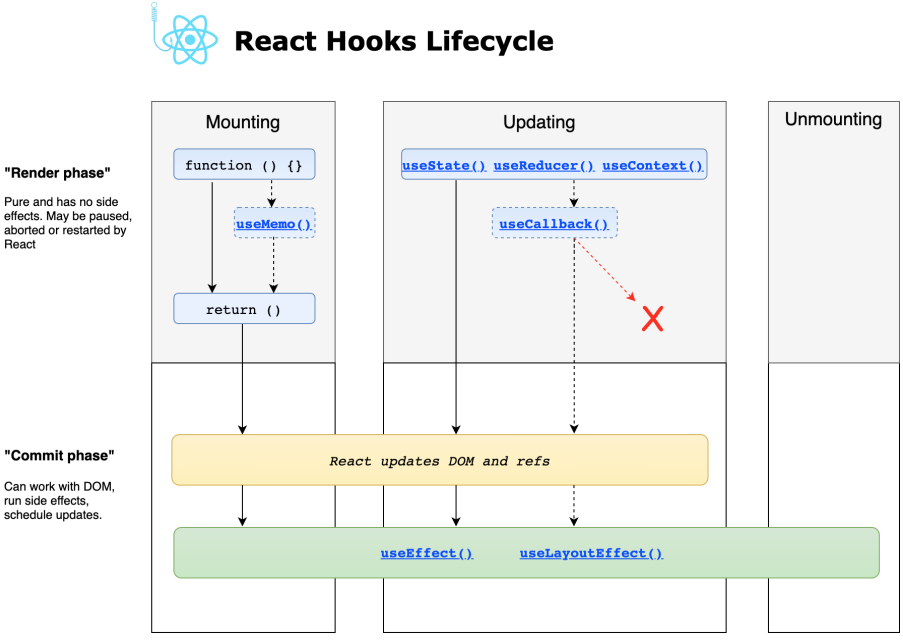

Figure 14: React Hooks Lifecycle[7]

By default, useEffect run after each render. useEffect takes a dependency array as an optional second argument. The dependency array specifies when to run the useEffect.

```
const [count, setCount] = useState(0);
useEffect(() => {
    document.title = `You clicked ${count} times`;
}, [count])
```

The dependency array [count] decides that the useEffect runs whenever count changes. A functional component can have multiple useEffects, and each useEffect can implement a clean-up function. Clean-up functions runs when the component unmounts, but will also run when the useEffect is called[8]. As a rule of thumb, the cleanup function runs before the actual useEffect.

3.4.2 App Context

Woodle has an App Context component, which is responsible for sharing data throughout the application. The app-context component holds a state of the current user, language, theme, and jwt-token and provided through the *AppContextProvider*, where every child of the provider has access to the state and can modify it as shown in the App.tsx file.

3.4.3 Routing

Woodle is using react-router[9]. App.tsx shows how the Router defines each possible route in the application with the *Route* component. The *Route* component takes a url and the component which should be rendered. The *Map* and *Profile* component is restricted for logged in users. Section 3.4.4 describes restrictions.

3.4.4 Higher Order Components - Authenticate

The higher-order component[10] (HOC) authenticate-route.js wraps the components in *Route* components that requires a login. When the *authenticate-route* component is called, the HOC will validate if the user is logged in. If not, it redirects the user to the login page.

3.4.5 Layout

The layout component, *layout.tsx*, is responsible for rendering the navbar, the component (e.g., *FormikSignIn.tsx*), and the container for Toasts. The layout component is placed inside the App.tsx file as the child of the *AppContextProvider*. The child of the layout depends on the current route, determined by the *Router*.

[8] Reactjs.org, use-effects, `https://reactjs.org/docs/hooks-effect.html`

[9] React Training, react-router, `https://reacttraining.com/react-router/web/guides/quick-start`

[10] Reactjs.org, higher-order component, `https://reactjs.org/docs/higher-order-components.html`

3.4.6 Formik

Woodle uses Formik [11] to handle form elements. Formik, together with yup[12] validation, makes form handling a lot simpler. Formik handles the form element values and the action when submitted. Yup handles the validation of the input. *useFormik* initiates the form values, validationSchema, and the onSubmit action. All there is left to do is specify the render function.

3.4.7 Toasts

Woodle uses Toasts from react-tosts[13] to show user confirmation and error messages. Toasts lives tithin the *ToastContainer* in the *Layout.tsx* component. To use the toasts, simply use the *ToastsStore.success()* or *ToastsStore.error()* functions.

3.4.8 Map

The map component uses Google Maps API[14] together with react-google-maps/api[15] npm-package to present the map.

Render map:
The render function in *google-maps.tsx* use the *LoadScript* component with the API-key to access the API and the *GoogleMap* component as a child component. The *GoogleMap* component takes multiple properties: a center location, a zoom-value, component styling, an id, and options. Furthermore, *GoogleMap* can have children, such as the *PolyLine*, which displays a path and the *Circle* that displays the player.

The *GoogleMap* component's useEffect (componentDidMount, componentDidUpdate, componentWillUnmount) sets up the geolocation tracking through the *navigator.geolocation.watchPosition()* function to find the user's location. If the runTracker-boolean is true, we track and show the user's path; otherwise, we show the user's location.

An activity starts by clicking the green playbutton (See Figure 1) and stopped by clicking the red stop button (See Figure 2).

Calculations:
The length calculation: is based on the Haversine Formula[16]. The haversine formula is accurate in most cases, but using a spherical model over earth gives errors up to 0.3%. The haversine formula (see Figure 15) calculates the distance

[11] jaredpalmer.com, Formik, `https://jaredpalmer.com/formik/docs/overview`
[12] npmjs.com, Yup, `https://www.npmjs.com/package/yup`
[13] npmjs.com, react-toasts, `https://www.npmjs.com/package/react-toasts`
[14] Google, Google Maps Platform, https://cloud.google.com/maps-platform/
[15] github.com, react-google-maps/api, `https://github.com/JustFly1984/react-google-maps-api/tree/master/packages/react-google-maps-api`
[16] movable-type, calculate distance between latitude/longitude points, https://www.movable-type.co.uk/

$$a = sin^2(\Delta\varphi/2) + cos(\varphi 1) * cos(\varphi 2) * sin^2(\Delta\lambda/2)$$

$$d = R * c$$

Figure 15: Haversine Formula[17]

'as the a crow flies', which means it doesn't consider mountains, hills, and other elevations.

```
function measureTwoCoordinates(
    latitude1: number,
    longitude1: number,
    latitude2: number,
    longitude2: number
){
    const R = 6371e3 // metres
    // latitude and longitude in radians
    const lat1 = (latitude1 * Math.PI) / 180
    const lat2 = (latitude2 * Math.PI) / 180
    const dLat = ((lat2 - lat1) * Math.PI) / 180
    const dLng = ((longitude2 - longitude1) * Math.PI) / 180

    const a =
      Math.sin(dLat / 2) * Math.sin(dLat / 2) +
      Math.cos(lat1) * Math.cos(lat2) * Math.sin(dLng / 2) * Math.sin(dLng / 2)

    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))

    const d = R * c // in metres
    return d
}
```

Figure 16: Haversine Conversion to Javascript[18]

The amount of steps is calculated based on the assumption that the average person has a stride length of 70cm. Figure 17 shows the calculation for the number of steps, where d = distance in meters.

$$(d/70) * 100$$

Figure 17: Number of steps calculation

The amount of calories is a rather complicated calculation because it requires so many factors to get right. Figure 18 shows the calculation for the number of calories, where d = distance in meters.

$$d * 0,05$$

Figure 18: Calories calculation

3.5 Testing & Continuous integration

Woodle tests with user-testing (see Section 5.1) on the essential functionalities explained in Section 2. Graphql testing covers the queries and mutations used in the application. The test queries and mutations runs on the AWS AppSync Console (see Section 5.2).

Woodle use Github Actions[19] as continuous integration. The pipeline begins when master receives an 'on push' event (see Section **??**) if it succeeds the build files will be transferred to the AWS S3 bucket(see Section3.2)

4 Limitations & Improvements

4.1 Tracking

Activities are only saved in the database once a user clicks the stop button (see Figure 2), which means an active activity will be lost if the user navigates to another page (e.g., Profile). The reason it's lost is that the active activity persists in the map components state, which will reset after a refresh. To improve this, every other page could open as a modal/popup, which won't refresh the maps state or active rides could save in localStorage[20].

4.2 Refresh problems

Woodle is struggling with refreshing problems. Currently, users will log out when refreshing the page, which is because the current session doesn't persist

[19] Github, Actions, https://help.github.com/en/actions/building-and-testing-code-with-continuous-integration/about-continuous-integration

[20] developer.mozilla, local storage, `https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Local_storage`

on page refresh. According to Github issues[21] this has been an issue. It is closed, but I didn't get around to fix it.

4.3 Development Environment and Tests

Woodle only has a live database environment, which means that I tested queries and mutations on 'live' servers, which is not preferred. It was only possible because one developer worked on the project. Furthermore, It would've been great to use cypress[22] to test front-end code coverage as well using it in the continuous integration pipeline.

[21] github.com/aws-amplify, How to persist login across page refresh 2480, `https://github.com/aws-amplify/amplify-js/issues/2480`

[22] cypress.io, Cypress, `https://www.cypress.io/`

5 Appendix

5.1 User Testing

5.1.1 Sign Up

Steps	Result
Click 'Sign Up'	redirect to /sign-up
Input: TestUser, 12345678, vilfredsikker@gmail.com	redirect to /confirm-sign-up
Receive email confirmation code	code: 463802
Input: TestUser, 463802	success: redirect to /login

5.1.2 Login

Steps	Result
Login input: TestUser, 12345678	Logged in, redirect to /app/map

5.1.3 Track Path, Create activity (Requires login)

Steps	Result
Click on 'Map'	redirect to /app/map
Make sure to allow location	See current location
Click Green Start Button (see REF HERE)	Track starts, Green Button becomes red
Walk a bit	A path is created
Click Red Stop Button (see REF HERE)	Track stops, Red Button becomes green

5.1.4 See stats, previous activities (Requires login)

Click 'Burger Menu' (see Ref)	Dropdown appears
Click 'Profile'	redirect to /app/profile
Navigate tab menu to 'Stats'	See stats
Navigate tab menu to 'Activities'	See activities

5.1.5 Add Friend, see users (Requires login)

Click 'Burger Menu' (see Ref)	Dropdown appears
Click 'Profile'	redirect to /app/profile
Navigate tab menu to 'All Users'	See users
Click 'ADD FRIEND'	Toast appears
Navigate tab menu to 'Friends'	See friends

5.1.6 Delete Friend (Requires login)

Click 'Burger Menu' (see Ref)	Dropdown appears
Click 'Profile'	redirect to /app/profile
Navigate tab menu to 'Friends'	See friends
Click 'DELETE FRIEND'	Toast appears

5.1.7 Delete Activity (Requires login)

Click 'Burger Menu' (see Ref)	Dropdown appears
Click 'Profile'	redirect to /app/profile
Navigate tab menu to 'Activities'	See activities
Click on an activity	
Click 'DELETE ACTIVITY'	see toast

5.2 GraphQL tests

5.2.1 listUsers

Query:

```
query listUsers {
  listUsers {
    items {
      id
      username
      activities {
        items {
          name
        }
      }
      friends {
        items {
          friend {
            friendName
          }
        }
      }
    }
  }
}
```

Result:

```
{
  "data": {
    "listUsers": {
      "items": [
        {
          "id": "623de62d-999d-46c6-8743-9a7dc71c41b9",
          "username": "TestUser",
          "activities": {
            "items": []
          },
          "friends": {
```

```
            "items": [
              {
                "id": "e172d41a-574d-47f0-85b8-e016125c4ae5",
                "friendID": "cdef38f4-2f28-4ae6-af76-579dd2466855",
                "connectorID": "623de62d-999d-46c6-8743-9a7dc71c41b9",
                "friend": {
                  "friendName": "Vilfred"
                }
              }
            ]
          }
        },
        {
          "id": "cdef38f4-2f28-4ae6-af76-579dd2466855",
          "username": "Vilfred",
          "activities": {
            "items": [
              {
                "name": "tor. 7.05.2020 10.50.14"
              }
            ]
          },
          "friends": {
            "items": []
          }
        }
      ]
    }
  }
}
```


5.2.2 createUser

Query:

```
mutation createUser {
  createUser(input: {
    username: "TestUser"
  }){
    id
    username
  }
}
```

Result:


```
{
```

```
  "data": {
    "createUser": {
      "id": "623de62d-999d-46c6-8743-9a7dc71c41b9",
      "username": "TestUser"
    }
  }
}
```

5.2.3 createFriend

Query:

```
mutation createFriend {
  createFriend(input: {
    id: "623de62d-999d-46c6-8743-9a7dc71c41b9"
    friendName: "TestUser"
  }){
    id
    friendName
  }
}
```

Result:

```
{
  "data": {
    "createFriend": {
      "id": "623de62d-999d-46c6-8743-9a7dc71c41b9",
      "friendName": "TestUser"
    }
  }
}
```

5.2.4 createFriendConnector

Query:

```
mutation createFriendConnector {
  createFriendConnector(input: {
    connectorID: "cdef38f4-2f28-4ae6-af76-579dd2466855"
    friendID: "623de62d-999d-46c6-8743-9a7dc71c41b9"
  }){
    friend {
      friendName
    }
  }
}
```

Result:

```
{
  "data": {
    "createFriendConnector": {
      "friend": {
        "friendName": "TestUser"
      }
    }
  }
}
```

5.2.5 createActivity

Query:

```
mutation createActivity {
  createActivity(input: {
    userID: "623de62d-999d-46c6-8743-9a7dc71c41b9"
    name: "TestActivity"
    length: 5200
    duration: 1200
    calories: 85
    steps: 6000
  }) {
    id
    name
    length
    duration
    calories
    steps
  }
}
```

Result:

```
{
  "data": {
    "createActivity": {
      "id": "d58f32cf-edb8-468d-b918-74983e8cdba5",
      "name": "TestActivity",
      "length": 5200,
      "duration": 1200,
      "calories": 85,
      "steps": 6000
    }
  }
}
```

5.2.6 getTestUser

Query:

```
query getTestUser {
  getUser(id: "623de62d-999d-46c6-8743-9a7dc71c41b9"){
    id
    username
    friends {
      items {
        friend {
          friendName
        }
      }
    }
    activities {
      items {
        id
        name
      }
    }
  }
}
```

Result:

```
{
  "data": {
    "getUser": {
      "id": "623de62d-999d-46c6-8743-9a7dc71c41b9",
      "username": "TestUser",
      "friends": {
        "items": [
          {
            "friend": {
              "friendName": "Vilfred"
            }
          }
        ]
      },
      "activities": {
        "items": [
          {
            "id": "d58f32cf-edb8-468d-b918-74983e8cdba5",
            "name": "TestActivity"
          }
        ]
      }
```

```
    }
  }
}
```

5.2.7 deleteActivity

Query:

```
mutation deleteActivity {
  deleteActivity(input:{
    id: "d58f32cf-edb8-468d-b918-74983e8cdba5"
  }){
    id
    name
    length
    duration
    steps
    calories
  }
}
```

Result:

```
{
  "data": {
    "deleteActivity": {
      "id": "d58f32cf-edb8-468d-b918-74983e8cdba5",
      "name": "TestActivity",
      "length": 5200,
      "duration": 1200,
      "steps": 6000,
      "calories": 85
    }
  }
}
```

5.2.8 deleteFriend

Query:

```
mutation deleteFriend {
  deleteFriendConnector(input:{
    id: "e172d41a-574d-47f0-85b8-e016125c4ae5"
  }){
    id
    friend {
      id
```

```
      friendName
    }
  }
}
```

Result:

```
{
  "data": {
    "deleteFriendConnector": {
      "id": "e172d41a-574d-47f0-85b8-e016125c4ae5",
      "friend": {
        "id": "cdef38f4-2f28-4ae6-af76-579dd2466855",
        "friendName": "Vilfred"
      }
    }
  }
}
```