**Web Programming EGCI427 (1) - Food recipe website to minimize food waste**

**Description:** The website will suggest recipes based on the ingredients available, as well as describe its implementation. More information and tips on cooking and preventing food waste can also be found.

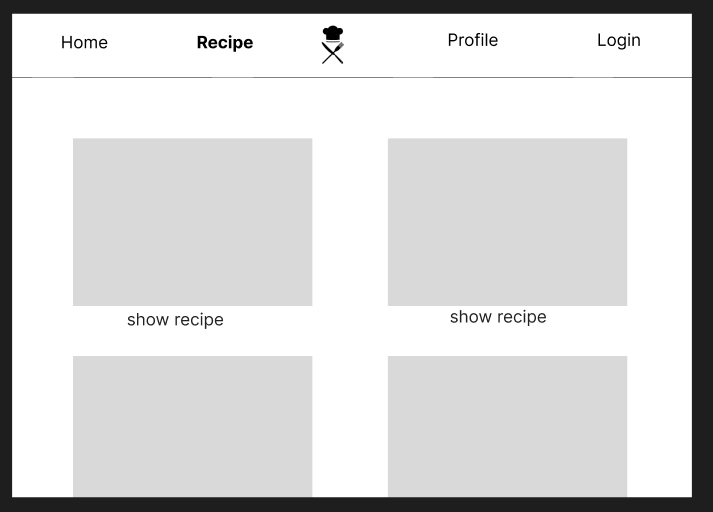
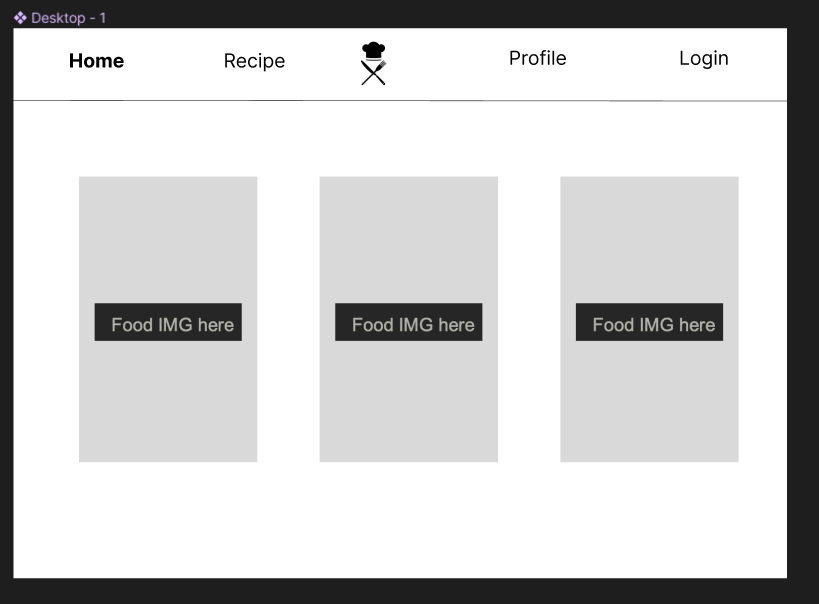
**List of Members:**

1. Robin Maurice Harbort, ID:6580735
2. Celina Müller, ID:6580738
3. Kevin Schulz, ID:6580728
4. Till Haid, ID:6580725
5. Magnus Freisens, ID:6580724

**Features of the application:**

* Get food inspiration from the *Home* website
* Search for recipes based on your available ingredients. The website will suggest recipes for you, which contains the ingredients you have. If you select multiple ingredients, you will see all recipes, which contains at least one of your ingredients.
* Get tips how to reduce food waste

**Website structure**



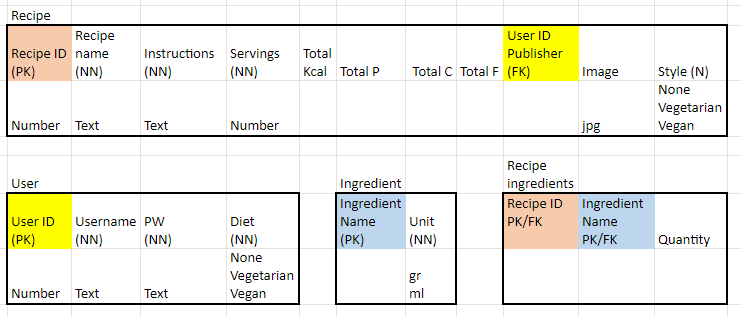
We use a *Home* site to show a few pictures of food, which you can cook with the help of our instructions. Also, we have the *Recipe* site to show all our recipes.

If you click on our logo, you will be forwarded to input your available ingredients to look for recipes.

*1/2*

**Data structure**

To store, add and access the data for our website, we designed a database structure, which contains four data sets. The data sets are normalized (stage 3) and linked with primary and foreign keys.



As shown in the picture above, we use the data sets *Recipe*, *User*, *Ingredient* and *Recipe ingredients*.

**Functionality of our files**

**Store.js**

This script creates and exports a Vuex store, which is a state management pattern and library for Vue.js applications. The store contains a state object with an initial state of an empty object containing the userInfo object. The mutations object contains a setUserInfo mutation, which is used to modify the state by setting the userInfo object to the payload passed in. This mutation logs the payload and the new value of userData to the console.

The actions object contains an async function called getUserData, which takes an email as an argument. This function retrieves user data from a Firestore database using the getFirestore, collection, query, and getDocs functions imported from the Firebase library. It then maps the returned document data to the userData array and logs it to the console. Finally, it commits the setUserInfo mutation with the first element of userData as the payload.

Overall, this script sets up a Vuex store with initial state and mutations to modify the state. It also defines an async action to fetch user data from Firestore and commit mutations to update the store with the fetched data**.**

**SignUp.vue**

The function signUp() is called when the user clicks on the signup button. It uses Firebase Authentication to create a user with their email and password. It also redirects the user to the Recipes page after successful registration. If there's an error during the registration process, an alert message with the error code is displayed.

The function addUser() is used to add the user information to Firestore without the password. This way, the password stays protected while the user information like diet and username are saved.

The HTML template of the signup page contains input fields for the user's email, password, username, and a dropdown menu to select their diet orientation. There is also a button to submit the form and a router link to redirect to the sign-in page if the user already has an account.

**SignIn.vue**

The **signIn** method is a component method in a Vue.js component called SignIn. This method is responsible for authenticating the user by using their email and password to sign in.

To accomplish this, the signIn method first retrieves the Firebase Authentication instance by calling the getAuth() function from the firebase/auth library. It then calls the signInWithEmailAndPassword() function with the email and password provided by the user as arguments to authenticate the user.

If the user authentication is successful, the method logs the user email to the console and dispatches a Vuex action to get the user data from the store using the getUserData() method. It then commits the user information to the store using the setUserInfo() method.

Finally, the signIn method redirects the user to the profile page by calling $router.replace('/profile'). If there is an error during the authentication process, the method displays an alert with the error message.

**RecipeList.Vue**

This script defines a Vue.js component called RecipeList that retrieves data from a Firebase Firestore database called "Recipe" and displays it as a list of recipes in the HTML template. The component fetches the data from Firestore and stores it in the recipes object. It then iterates over this object and displays each recipe's image, name, and a router link to the recipe details. Additionally, there is a "logout" button that allows the user to log out of the application, which redirects the user to the "signin" page. The CSS styling is scoped to this component only.

**FindRecipes.Vue**

The component displays three dropdown menus where users can select their ingredients from a pre-defined list of options. After selecting their desired ingredients, the user clicks the "Submit" button, which triggers the findRecipes() function.

The findRecipes() function presumably sends a request to an API or backend server that returns a list of recipes that can be made with the selected ingredients. The component then displays the recipe names, images, and a "See Recipe" button for each recipe that was returned.

When the "See Recipe" button is clicked, the user is directed to another page (presumably using Vue Router) where they can view the full recipe details.

The script also includes a "theme-selector" component that allows users to change the color scheme of the page by clicking a paint bucket icon.

**Info.vue**

This script is a Vue.js component that retrieves and displays information about a specific recipe from a Firestore database. It queries the database using the recipe name as a parameter, then retrieves and displays various information about the recipe, such as its name, style, servings, instructions, and ingredients. It also accesses the details of each ingredient, such as its unit, and displays them accordingly. The template section of the code is an HTML document that displays the retrieved information using Vue.js syntax. The script also includes a function that changes the theme of the page when a user clicks on the corresponding button

**Profile.vue**

This file is a Vue.js single-file component that displays the profile page of a user in a web application. The component consists of a template section and a script section

The template section contains HTML markup and defines the structure of the profile page. It includes a header section, a container section that displays user information such as the user's name, email, and diet preference. The template also includes a footer section that displays links to other pages in the web application and a logout button.

The script section contains the logic for retrieving user information from the Firebase Firestore database and displaying it on the page. It imports Firebase and Vuex functions, initializes Firebase authentication instance and an array to store user data retrieved from Firestore. The computed properties map the Vuex state to the component's data properties.

The script section also includes a logout function that logs the user out of the application when the logout button is clicked. The function uses the Firebase authentication signOut method to sign out the user.

**Index.js**

This file is a router file for a Vue.js web application. It defines the routes for the different pages of the application, as well as the authentication requirements for each page.

The file starts by importing the necessary modules and components, including createRouter and createWebHistory from the vue-router module, as well as various Vue components for different pages of the application. It also imports getAuth and onAuthStateChanged from the firebase/auth module.

Next, the router is created using the createRouter function. The history property is set to use createWebHistory with a base URL taken from the import.meta.env.BASE\_URL property. The routes property is an array of objects representing the different routes in the application.

The first route redirects the user from the root URL to the home page. The second route redirects any unknown URL to the sign-in page. The rest of the routes represent specific pages in the application, such as the recipe list, recipe details, and user profile. The meta property on each route is used to indicate whether authentication is required to access the page.

The getCurrentUser function uses Firebase authentication to return a Promise that resolves with the current user object or rejects with an error.

The router.beforeEach function is used to intercept every navigation request and check whether authentication is required for the destination page. If authentication is required, it checks whether the user is authenticated using getCurrentUser, and either allows or denies access to the page accordingly. If authentication is not required, it simply allows navigation to proceed.

Finally, the router is exported as the default export of the module. This allows other parts of the application to import the router and use it to navigate between pages.

*2/2*