Recipe application

Sara Kitzing, Hanna Carlsson, Omar Oueidat, Tobias Lindgren

DAT076 - Web applications

1 Introduction

The purpose of the app is to find new recipes for your cooking and share your favorite recipes with others on the platform. In the app it is possible to browse, read and create recipes. When logged in you also get the possibility to save recipes for later and edit the recipes created by the user.

Link to git repo: `https://github.com/OmarOueidat/DAT076`

2 List of use cases

- Create a recipe
- Browse recipes
- Show recipes by category
- Read recipe
- Edit recipe
- Create user
- Edit user
- Delete user
- Log in
- Log out
- Save recipe for later
- Unsave recipe for later

3 User manual

There are two possible ways of using the app. The user can either use it without an account, with less functionality, or with an account in order to get all the features.

Without an account the user can browse, view and create recipes. The user can also search recipes by categories. To get more features the user can either log in if it already has an account, or create a new one.

When logged in the user gets all the functionality as without an account. However, the user also have the possibility to save recipes for later in its profile. In the users profile one can also edit the users password and name as well as find a list of all recipes created by the account. From the list of created and saved recipes in the profile it is possible to view the body of the recipes. For the recipes created by the account, one can edit the information of the recipe.

The user can delete its account from the edit user-view. When a user is deleted the recipes created by it will still be left in the app.

4 Design

In order to create a full stack application we have used MERN stack development. MERN is the acronym for MongoDB, Express, React and Node. MongoDB is the database system and Express and Node are back-end web application framework and runtime framework respectively. React works as a front-end library.

4.1 React

React is a JavaScript based library, used for building user interfaces. React makes it easy to pass data because of it being component-based [1], meaning you can encapsulate functionality and HTML in a component and easily use the component within the application, and through *props* you can send properties to children components to be able to pass information through the entire application. Also, everything is written in .js-files, and the react framework will convert the relevant code to HTML.

In our application, almost everything is it's own component which simplifies a lot of our implementation. For example, we have a component called *recipe-list* which shows the recipes according to the filtering from the sidebar. The recipe-list-component fetches all recipes from the database, and through some javascript code filters out all recipes that don't apply to the filter and then, for each recipe, creates a *recipe-miniature*-component. The functionality and HTML for that component is then encapsulated, which leads to more readable code.

4.2 Bootstrap

Bootstrap is a framework for the web containing design templates for HTML and CSS [2]. In the application, Bootstrap is used in order to design the components. Examples of components which we use bootstrap, is all forms (such as log in, create recipe and edit user info), the cards for showing miniature recipes and almost all buttons.

4.3 MongoDB

MongoDB is a document NoSQL database which stores data in JSON-like objects [3]. In the application it is used together with MongoDB Atlas, a free cloud service up to 512MB, in order to sync the data and avoid having local versions of the database. The web application also uses mongoose which is an library for MongoDB and Node.js. Mongoose is used to translate between the code-objects and the representation in the database [4]. For example, findByIdAndUpdate(id, update) is a mongoose function used in the application which uses mongodb's own function update(query, update, options). In the application mongoose is also used to create scheamas for the User, Recipe and Category which helps give a skeleton structure for the object being saved in the database.

4.4 Axios

Axios is a "Promise based HTTP client for the browser and node.js" [5]. In the application, axios is used to help with the CRUD-operations and make the code both simpler and easier to understand.

4.5 Express

Express is a web application based framework for Node.js [6]. Express is used for the server side of the application, it receives HTTP requests and handle these accordingly. express.Router() is also used to route the URIs and make the code cleaner and reduce the amount of code.

4.6 Node.js

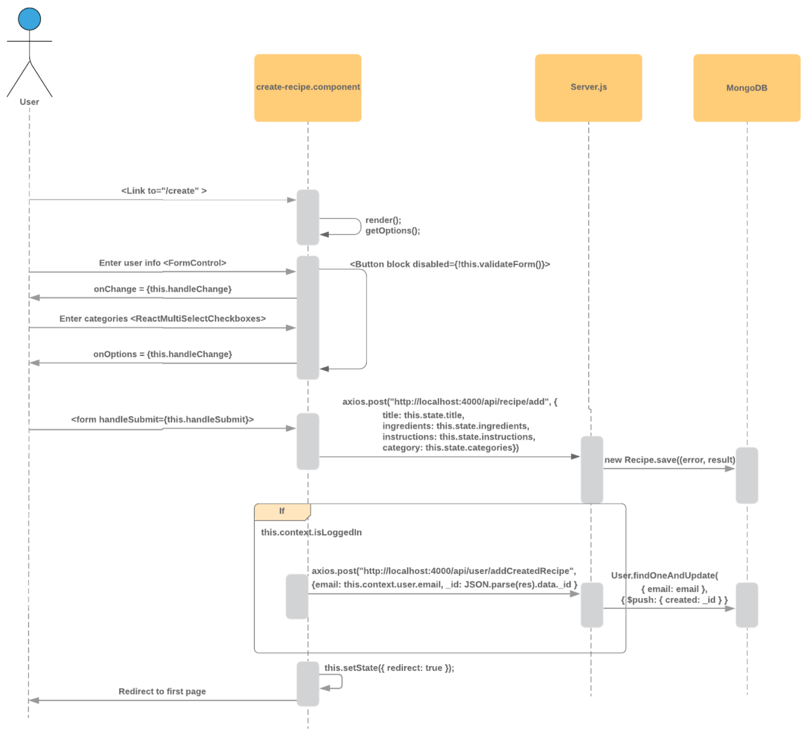
Node.js is a runtime built JavaScript environment [7]. It basically includes everything we need in order to run the code. It is light-weight, non-blocking I/O event driven which makes it more efficient. Node is great when the application should handle several requests at the same time. Node also gives the opportunity to use the Node package manager (npm) which is used to install different packages created by the community or by the Node team themselves.

We used Node to start the server and the backend by using npm, we also install every single package within the app by using npm.

5 Application flow

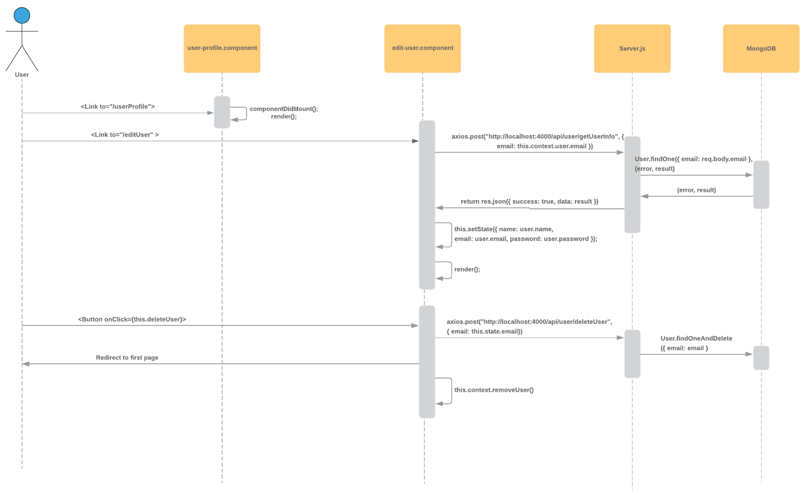
The use cases below are chosen in order to show how we interact with all the component as well as the database for the different functionality of the application. Note: we have decided to only show the calls and results which we actually use later on to simplify the sequence diagrams. We have also simplified some of the calls to make it easier to follow the call.

5.1 Create recipe


User
create-recipe.component
Server.js
MongoDB
<Link to="/create" >
render();
getOptions();
Enter user info <FormControl>
<Button block disabled={!this.validateForm()}>
onChange = {this.handleChange}
Enter categories <ReactMultiSelectCheckboxes>
onOptions = {this.handleChange}
<form handleSubmit={this.handleSubmit}>
axios.post("http://localhost:4000/api/recipe/add", {
title: this.state.title,
ingredients: this.state.ingredients,
instructions: this.state.instructions,
category: this.state.categories})
new Recipe.save((error, result)
If
this.context.isLoggedIn
axios.post("http://localhost:4000/api/user/addCreatedRecipe",
{email: this.context.user.email, _id: JSON.parse(res).data._id }
User.findOneAndUpdate(
{ email: email },
{ $push: { created: _id } }
this.setState({ redirect: true });
Redirect to first page

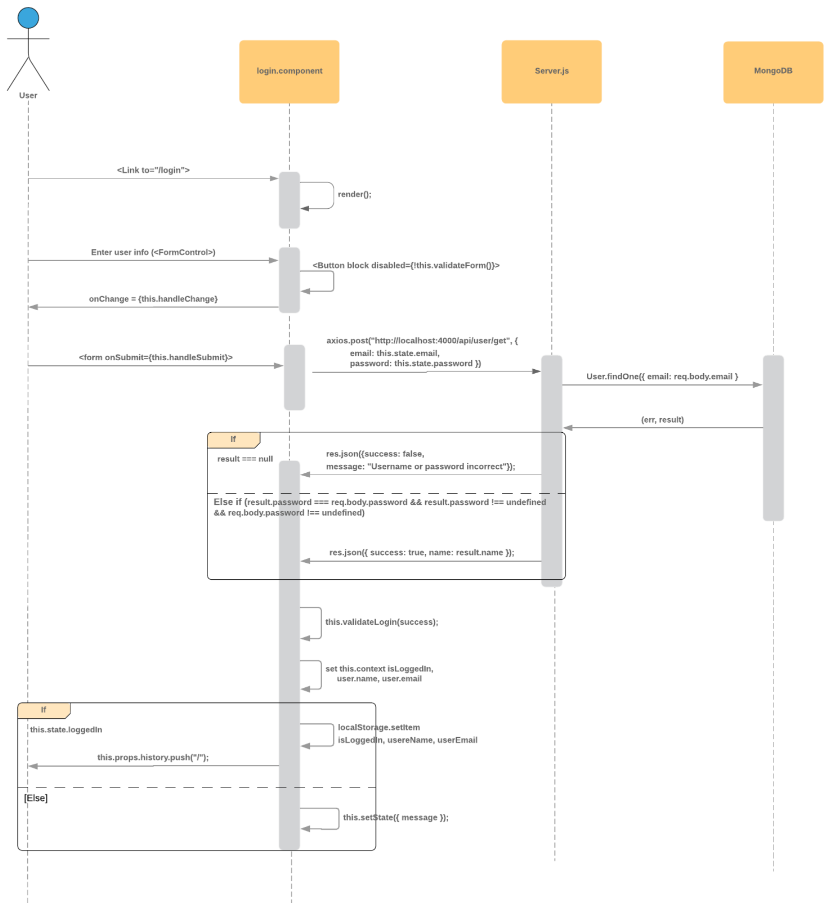
To create a recipe, the user first has to press the Create Recipe button, which will take the user to the create recipe page. The user then has to complete the form, while the fields are empty the user can not press the submit button. The component will change its state when the user changes the text or the multiselect checkboxes. When submit is clicked there will be a call to the database which adds the recipe. If the user is logged in the recipe id will also be added to the accounts created-field to make it possible to keep track of which account that created which recipe. The user then gets redirected to the home page.

5.2 Delete user


User
user-profile.component
edit-user.component
Server.js
MongoDB
<Link to="/userProfile">
componentDidMount();
render();
<Link to="/editUser" >
axios.post("http://localhost:4000/api/user/getUserInfo", {
email: this.context.user.email })
User.findOne({ email: req.body.email },
(error, result)
(error, result)
return res.json({ success: true, data: result })
this.setState({ name: user.name,
email: user.email, password: user.password });
render();
<Button onClick={this.deleteUser}>
axios.post("http://localhost:4000/api/user/deleteUser",
{ email: this.state.email})
User.findOneAndDelete
({ email: email }
Redirect to first page
this.context.removeUser()

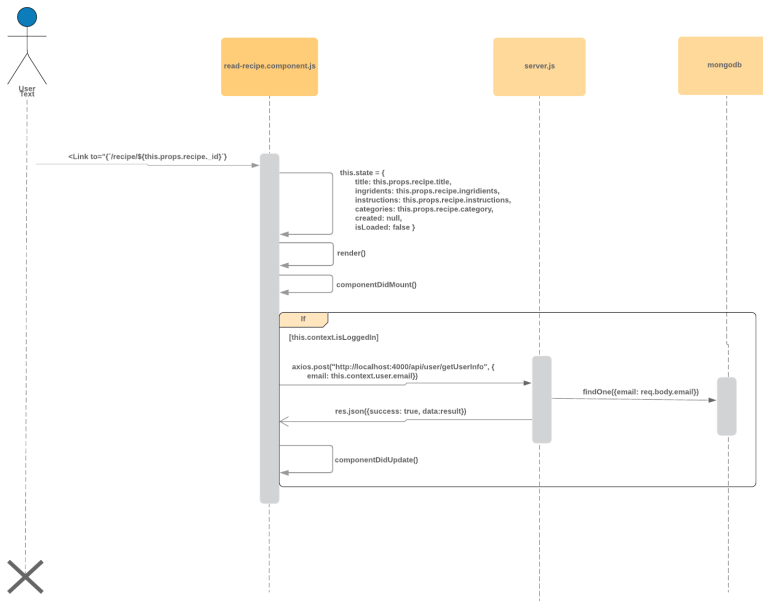
To delete an account the user has to press the user name in order to render the user profile page. From the user profile the user can press the edit user button which renders the edit user page and gets the user information from the database. When the delete-button in edit user is clicked there will be a call to the database to delete the account and redirect the user to the home page. Note that the recipes created by the user wont be deleted when the user is deleted.

5.3 Log in


User
login.component
Server.js
MongoDB
<Link to="/login">
render();
Enter user info (<FormControl>)
<Button block disabled={!this.validateForm()}>
onChange = {this.handleChange}
<form onSubmit={this.handleSubmit}>
axios.post("http://localhost:4000/api/user/get", { email: this.state.email, password: this.state.password })
User.findOne({ email: req.body.email }
(err, result)
If
result === null
res.json({success: false, message: "Username or password incorrect"});
Else if (result.password === req.body.password && result.password !== undefined && req.body.password !== undefined)
res.json({ success: true, name: result.name });
this.validateLogin(success);
set this.context isLoggedIn, user.name, user.email
If
this.state.loggedIn
localStorage.setItem isLoggedIn, usereName, userEmail
this.props.history.push("/");
[Else]
this.setState({ message });

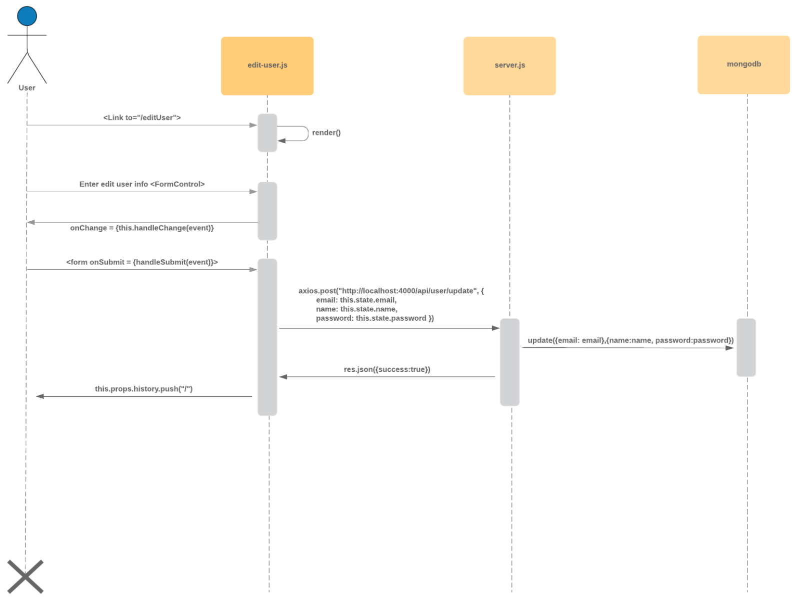
The login proccess starts with the user pressing the login button which then links them to the login page that renders. If the user wants to log in they have to complete the form and while the form is empty the user can not press the submit button. However, on changing the text the component will change their state and when submitting the form there will be a call to the database which retrieves the information and checks if the user name exists or if password is correct, then it redirects the user to the homepage or tells them that the password or email is wrong.

5.4 Read recipe


User
Text
read-recipe.component.js
server.js
mongodb
<Link to="{´/recipe/${this.props.recipe._id}´}
this.state = {
title: this.props.recipe.title,
ingridents: this.props.recipe.ingridients,
instructions: this.props.recipe.instructions,
categories: this.props.recipe.category,
created: null,
isLoaded: false }
render()
componentDidMount()
If
[this.context.isLoggedIn]
axios.post("http://localhost:4000/api/user/getUserInfo", {
email: this.context.user.email})
findOne({email: req.body.email})
res.json({success: true, data:result})
componentDidUpdate()

When pressing the show recipe button, the user gets redirected to the read recipe page. The recipe props gets sent to the recipe as well making it possible to set all the component's states. The component then render, adding the content of the states to each part of the recipe body. If the user is logged in, the recipes created by the user are retrieved in order to make it possible for the user to enter the edit recipe page from this view.

5.5 Edit user


User
edit-user.js
server.js
mongodb
<Link to="/editUser">
render()
Enter edit user info <FormControl>
onChange = {this.handleChange(event)}
<form onSubmit = {handleSubmit(event)}>
axios.post("http://localhost:4000/api/user/update", { email: this.state.email, name: this.state.name, password: this.state.password })
update({email: email},{name:name, password:password})
res.json({success:true})
this.props.history.push("/")

To edit a user the user has to press the edit user button in the user profile page to be directed to the edit user page. The edit user page renders and the fields fills with the user info. When the user changes the user info, the states of the component is updated. Submitting the form will call the database which updates the users information. The user will be redirected to the user profile page.

6 Responsibilities

All of the group have been active in the discussions of deciding the main design of the app. All members have also been responsible for proof-reading the report and checking the functionality of the application.

Hanna has been responsible for setting up the database and the server side of the application together with Tobias. Creating the update recipe and edit user components, save/unsave recipes to a user and save created recipes. Worked together with Sara to connect the user profile to the database and Omar with setting up context, along with some minor fixes throughout the application.

Tobias has been responsible for setting up the database together with Hanna, creating the component for read-recipe, sidebar (including filtering), setting up and connecting the category-database to components, some routing, also worked together with Sara when connecting the recipe list to the database and some minor fixes throughout the application.

Omar has been responsible for making sure the navigation works. He also worked on state-handling with the context API with Hanna, so that there could be a way to pass the user from component to component. Omar created the Login component and made it possible to login and also logout. He has also done minor fixes.

Sara has been responsible for creating the components for the user profile, create recipe, recipe cards and recipe list. She has worked together with Tobias when connecting the recipe list to the database and together with Hanna when connecting the user profile to the database. She has also made some minor fixes to the application.

References

[1] React. ”React”.[Online]. Available: `https://reactjs.org`, retrieved: 2019-03-13.

[2] Wikipedia. ”Bootstrap (front-end framework)”. [Online]. Available: `https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)`, retrieved: 2019-03-13.

[3] MongoDB. ”What is MongoDB?”. [Online]. Available: `https://www.mongodb.com/what-is-mongodb`, retrieved: 2019-03-13.

[4] Karnik, Nick. ”Introduction to Mongoose for MongoDB”, 2018. [Online]. Available: `https://medium.freecodecamp.org/introduction-to-mongoose-for-mongodb-d2a7aa593c57`, retrieved: 2019-03-13.

[5] Axios. ”Read Me”. [Online]. Available: `https://github.com/axios/axios`, retrieved: 2019-03-13.

[6] Express. ”Express”. [Online]. Available: `https://expressjs.com`, retrieved: 2019-03-13.

[7] Node.js. ”Home”. [Online]. Available: `https://nodejs.org/en/`, retrieved: 2019-03-13.