Recipie - Essential Pie Recipes

Application details

There are three important layers to the application - Database, Server and Client. The Database is on a normal MySQL server, and it's connected to the Server of the application via Sequelize. The Server uses Express routers to send JSON data - fetched from the Sequelize models - to the Client. The Client sends and receives data via the HTTP commands POST and GET, sent to the server as JSON requests. These JSON requests are then handled by the React framework to present it to the user.

The Database is created to simplify the data storage as much as possible, with restrictions and associations on the data built-in. A *User* has a unique username, and a password. A *Recipe* has a title, description, ingredient list and a link to a picture. There's nothing unique in a Recipe, so it has a column with recipe id. There's also *Tags*, which just consist of a unique tag name. A user can own a recipe, which is shown with an extra column *user* in the recipe table. Tags are used to describe recipes, and is a many-to-many relationship, thus creating an extra table with those associations. Sequelize handles the table structure behind the scenes for us, but the ER diagram is shown in Figure 1.

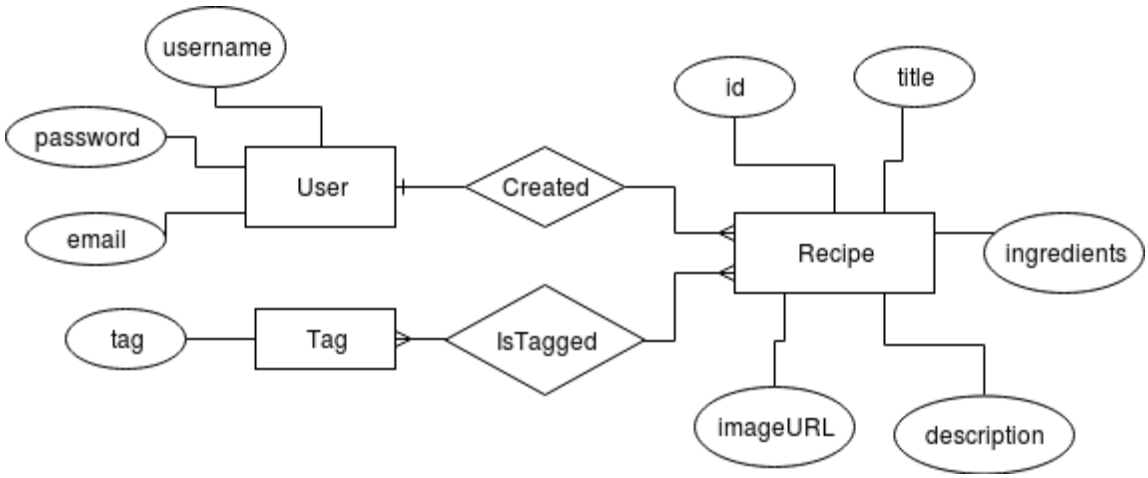

Figure 1. Relational table for the application. The attributes tag, username and id should be keys, but that disappeared somehow…

The Server has three important modules - migrations, models and routes. It begins with migrations, which contains the database entities. However, we cannot interact directly with these in the code, but must create a model for each entity. Located in the models module, those contain information about associations between the entities, and each model exports an object that contains methods used to handle the entities. The migrations along with the association information in the models is what Sequelize uses to automatically generate the correct tables in the database. The third module is the routes, and that is what the Client can use to interact with the database. The route classes use the models to set up functions that can GET and POST data via HTTP, on different addresses.

Client
The Client is built using the React.js framework. The client is responsible for presenting the data stored in the database. It does this by using the rest api located at the server. Therefore there is nearly no logic, concerning data modification in the client. It is all done on the server. The client communicates with the database (though the server) with jquery. More precisely GET and POST through ajax calls.

For session-handling we use react-cookie. After providing login info we check if the provided user exists in the database and then create a cookie which can be accessed throughout the whole application until the session is ended (when the user logs out).

For routing in the application we use react-router.

In the application we use the stylesheet library react-bootstrap. In most cases we only use the default bootstrap styling, hence there is a small amount of custom made css in the application.

The main module we have been implementing is the Src-package. In here we have the Components-package which holds all the components, for example the firstpage.js.

Flow - Creating a recipe

We assume that a user, Bob, is signed in on the site, and wants to create a new recipe. Bob can click the “Create Recipe” button, which takes him to a form, where he can fill in his ingredient list, instructions and title of the recipe, (along with a link for a picture). Bob now clicks “Save”. The save-button calls the POST function on *server-address/recipe/create*, and sends the recipe data with the call. This is where the server steps in. Realizing that a call has been made, the matching router will then use Sequelize to create a new entry in the recipe table (there is no need to define any SQL code, as this is handled automatically by Sequelize). After the recipe has been created, it is connected to the user who created it as a column in the recipe table. This connection must occur after the entry has been made, as both entities must exist in order for Sequelize to connect them. The same also applies for any tags that the user writes, but here, a tag doesn’t need to exist, so an entry in the tag table is created if it is a new tag.
When Bob goes back to his profile page, the updated list of recipes are fetched and he can now see his newly added recipe.