CS 458 Programmer's Manual

Planet Rocket: Alec Levin, William Doudna, Eric Mott, Levi Pole

Table Of Contents

1.0 - Introduction:

This web application is built using a MERN stack (MongoDB-Express.js-React.js-Node.js). What is important to remember about this type of stack is that the front end and the back end run as seperate applications. The way that the front end receives and stores data in the back end is via sending HTTP requests to endpoints that are setup in our backend.

1.1 - Backend Introduction:

Here is an example of a url for a simple get request:

planetrocket.com:3001/idea/hs83SJQK/user/steven808/oneIdea/ship

The front end is requesting data at port 3001 because that is the port that our backend server is running on.

'/idea/' indicates that the user will be fetching an idea object.

- For the most part our routes align with CRUD operations
- This means that each type of HTTP request symbolizes an action on an object in our backend
- POST - Create an object
- GET - Read an object
- PUT - Update an object
- DELETE - Delete an object

The Next sequence on characters ‘hs83SJQK’ is an example of an api key that a user received upon login and will use to authenticate all of their api requests.

- api keys are used to authenticate a users request and ensure that other users do not get access to data that they are not supposed to see

The ‘user’ portion of the url indicates that the next part of the url will be the username of the user that is making the request, ‘steven808’ is the user making the request, ‘/oneIdea/’ indicates thatthe next portion of the url will indicate the ID of the idea to be read to the front end, and ‘ship’ is the ID of the idea object to passed back to the frontend

1.2 - Frontend Introduction:

React is a frontend technology that utilizes what it calls components to manage it’s contents. These components each have their own life cycles that keep track of values stored client side and allow react to modify itself without making a request to a server.

You can see this in effect in our login system. After a user logs in successfully the user is given a fresh api key and it is stored within a state by the main component (App.js). Upon receiving the api key the login screen is switched out for the dashboard. This happens without any requests made, instead App.js is checks to see if there is an api key stored within it’s state and if there is it knows to switch to the dashboard.

To communicate with the backend React needs to use a library called axios. This allows it to send a HTTP request to the backend. Through axios we are able to manipulate Reacts state control capabilities to send specific data through an HTTP request, then react upon the results of that request. For example, if we wanted to grab a users ideas:

- Grab the user’s login info such as their username from a component’s state
- Send that info through an HTTP request to the backend
- On receiving the response of ideas the user is involved in, we could store the results within a state and then output the results to a list

2.0 - File Hierarchy Structure and Details

Below is a detailed list describing the file hierarchy and structure of the software including basic details of relevant folders and files.

2.1 - Backend Files:

apiFunctions -- contains the functions for all of the HTTP endpoints that return and modify our MongoDB object.

keyVerify --A folder within the apiFunctions folder that contains a function that authenticates api keys

bin -- contains the scripts that start the backend both locally and on production

config -- contains a config file that connects mongoose (ORM) to our Mongo database.

models -- the folder containing the models for our MongoDB objects

public -- ignore, auto generated by express generator

routes -- api.js is the file where all of the api routes are declared and where the associated functions are attached to the endpoints

views -- ignore, auto generated by express generator

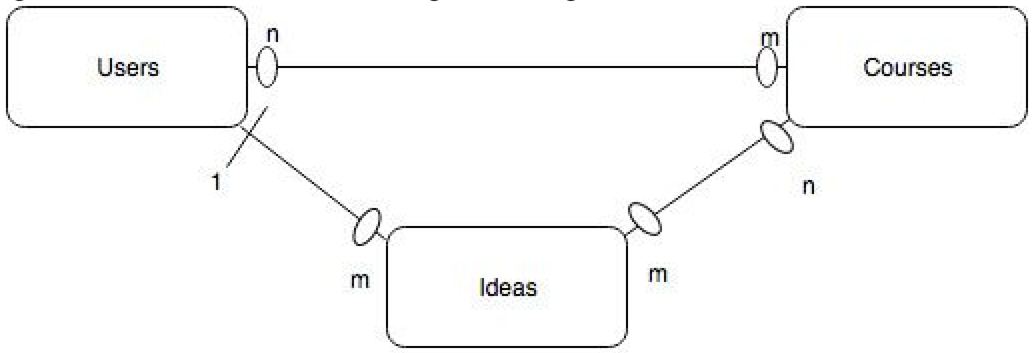
app.js -- This file contains all middleware and connects our routers to the server

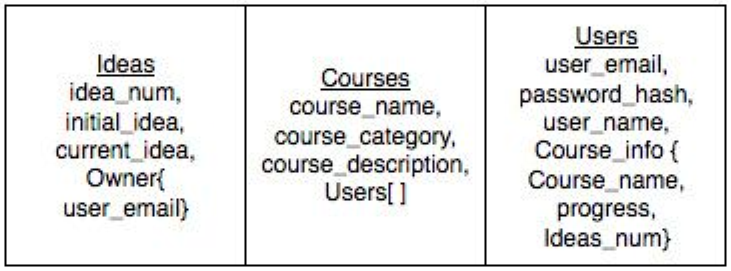
package-lock.json -- list of all npm node modules

package.json -- list of dependencies and scripts to start the project locally

2.1.1 - Database Schema

Our MongoDB makes use of the following data design.





Ideas	Courses	Users
idea_num,	course_name,	user_email,
initial_idea,	course_category,	password_hash,
current_idea,	course_description,	user_name,
Owner{	Users[ ]	Course_info {
user_email}		Course_name,
		progress,
		Ideas_num}

2.2 - Frontend Files:

Package.json -- list of dependencies and scripts to start the project locally

Package-lock.json -- list of all npm node modules

public -- This folder is the landing page of React if the React App has not been properly started.

src/Components -- This folder contains all our components for React. Components hold our Frontend's look and functionality.

config/ipAddress.js -- This holds the ip address of our backend database.

config/promptText.js -- This holds the text displayed throughout the bmcCourse

redux -- This folder contains the files to implement redux. However they are not currently in use.

test -- This holds a text file named testing.txt. Not in use.

App.css -- This file holds the CSS styling for App.js

App.js -- This is the main file of the React App. All components are maintained here.

App.test.js -- This file is not in use.

Index.css -- This file holds the CSS styling for Index.js

Index.js -- This file tells React what our main file is and directs it there.

serviceWorker.js -- This file is currently not in use.

3.0 - Downloading, Installing, and Running Software on Production Server:

This walkthrough will enable you to install the software onto a server making the following assumptions:git is installed, yum or similar download client is accessible, and a server ready computer is in use. If these assumptions are satisfied:

1. Install and start Nginx, MongoDB (on port 27017), node.js, npm, and pm2. Most of these can be installed using your preferred service (yum, apt-get, pip, brew, etc.).
2. Clone the repository available at https://github.com/emott10/PlanetRocket2.0 anywhere on your server.
3. Change the ipAddress listed at frontend/src/config/ipAddress.js to the base domain you are using, for example *const ipAddress = 'https://dev.arlevin.org';*
4. Run *npm install* from each the backend and frontend folders.

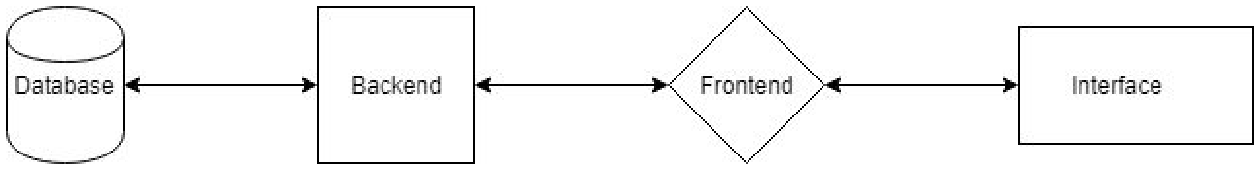
5. From the backend run the command *pm2 start bin/www-prodo* This command tells the node process manager (pm2) to start the node backend located at bin/www-prodo.
    a. Note that the www-prodo starts an HTTPS node backend, this should only be used if you also have an SSL certificate for your frontend running similarly on HTTPS.
    b. Note that pm2 leaves your backend running in the background, a very helpful tool for being able to exit the command line and leave your process (backend) running.
6. From the front end, run *npm run build* This command builds a production level build of your front end and places it in the frontend/build folder.
7. Move the build folder to /usr/share/nginx/
8. Nginx out of the box serves will serve from /usr/share/nginx/html folder. Change this by editing the following line of /etc/nginx/nginx.conf from */usr/share/nginx/html* to */usr/share/nginx/build*
9. You now have nginx serving your frontend as a static server running on your ip address or domain and pm2 running your backend.

4.0 - Downloading, Installing, and Running Software Locally:

1. Clone the repository to a folder on your computer
2. Make sure that node.js and npm (node package manager) are both installed
3. Make sure that MongoDB is installed and running on localhost:27017
    a. If it isn't, change to the folder where it is installed and run the mongo.exe file that is within the 'bin' folder
4. From within each the 'backend' and 'frontend' folder run the command 'npm install'
5. From the 'backend' folder, type the following command into your terminal 'npm run dev'
6. You can now use the website by accessing localhost:3000. Refer to the User Manual for usage instructions.

5.0 - Frontend/Backend Interfacing:

This diagram represents the flow of data of our app



6.0 - Managing Session State and Passing Data:

We manage the session states with React states. For example, to keep track of how the web application knows if a user is logged in we pass an API key to each component which states if the user is logged in:

```
<RegisterBox newKey={this.saveApiKey.bind(this)}
     checkLogin={this.checkLogin.bind(this)} />
```

This code snippet is found in App.js on line 47. As you can see you need to pass the login API key as well as the the function to check if the user is logged in to each component when they are called. The function sets the state of the component to the respected value, such as true if the user is logged in or false if the user is not. Also, within these states we pass user data. This data holds the username, login state, api key, and the users score.