Forums - Application Report

Advanced JavaScript Continuous Assignment 2

George Blanaru

N00153498

[GitHub Link](https://github.com/georgeBl/forums)

Table of Contents

[1. Introduction 2](#_Toc5671904)

[2. Implementation 2](#_Toc5671905)

[MongoDB 2](#_Toc5671906)

[Express 2](#_Toc5671907)

[React 3](#_Toc5671908)

[Node.js 5](#_Toc5671909)

[3. User stories 6](#_Toc5671910)

[4. Reflections on the project 6](#_Toc5671911)

# Introduction

This report will walk trough most of the key features of the project. The objective of the project is to create a usable MERN application. The theme of the project is a forum-like application where users can log in, create threads and post comments in those threads. The application should be able to respond to all the user request. It also needs to be modular and the size of the data should not affect the performance.

# Implementation

The project is composed of two elements, a React App (client side) and a Node.js server (server side), that run concurrently to create a MERN application. The term MERN refers to MongoDB, Express, React and Node.js.

## MongoDB

MongoDB is a non-relational database that stores the data in JSON-like documents. In this project, the database called “react-ca2” contains four collections:

1. “users” - stores all the user information (name, email, password, etc.);
2. “subforums” – stores all the subforum information (name, logo);
3. “threads” – stores all the thread information (title, subtitle, body, image, etc.), but also a reference to the subforum it belongs;
4. “comments” – stores all the comments and a reference to the thread and the user it belongs.

## Express

Express is a Node.js web application that provides a set of features for a web or mobile application. One of the features provided by Express is creating a REST API. In simple terms a Representational State Transfer (REST) API allows a developer to do different types of requests (e.g. GET, PUT, DELETE, POST) over the internet without installing any additional libraries. The submitted application defines the following API routes:

* GET “/api/user” – this API returns from the database the user that is logged in. If the user is not logged in (no token is provided), nothing will be returned;
* GET “/api/user/:id” – returns the name of a user depending on the parameter “id”. This route does not require any authentication token;
* POST “/api/register” and POST “/api/authenticate” – returns an authentication token in case the user submits the right requests (the right email, a valid password, etc.);
* GET “/api/checkToken” – returns a status code of 200 if the user provides a token and a status code of 401 if it doesn’t;
* GET “/api/logout” – erases the token provided by the user by setting the token variable empty;
* GET “/api/subforums” – returns all the subforums from the database
* GET “/api/subforums/:id” – returns a subforum by id
* POST “/api/comment” – inserts a comment into the database
* GET “/api/comment/:threadId” – returns all the comments from a thread
* GET “/api/threads/:name” – returns all the threads that relate to the name of a subforum (e.g. if the name is “games” returns all the threads in the games subforum)
* GET “/api/thread/:id” – returns a thread by id
* POST “/api/thread/:name” – creates a new thread in the database for the named subforum
* DELETE “/api/thread/:id” – deletes a thread by id
* PUT “/api/thread/:id” – updates a thread by id
* PUT “/api/ratethread/:id” – updates the rate value in a thread
* GET “/api/checkVote/:threadId – returns a status code of 200 and a message containing the vote in case it exists, if the user is logged in

Express is also used to serve the index.html file in the client-side folder.

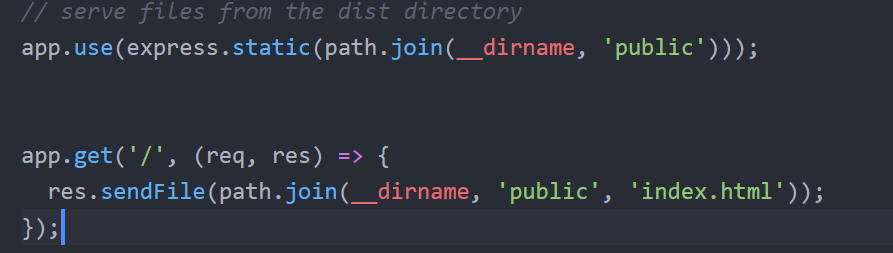


Figure 1 Express used to serve html document

## React

The client side of the project and everything the user sees in the browser is created using React. React is a JavaScript UI library created by Facebook that allow developers to create HTML components. Using JSX, React Components can create and manipulate HTML DOM elements in real time. In this project, components are rendered conditionally depending on multiple factors (e.g. user is logged in, browser link). The following components were used in the app:

* App.js – Like most react applications this is the first component rendered in the browser. This component stores in its state if the user is logged and a list of all the subforums available in the database. It also handles the login and logout functionality. All the possible routes are also set in this component inside a Switch tag imported from the react-router-dom library, as shown in Figure 2.



Figure 2 List of routes

Finally, when all the routes are set, the component renders the footer of the page.

* Navbar.js – This is the second component rendered and it contains link to different places in the app. On the left side there are two links available, from which one is hidden/inaccessible if the user is not logged in. The “Home” link will redirect the user to the root of the website, where the SubforumList component is loaded. The next link, “View Details” redirects the user to the Details component where all its details are displayed. In the right side of the Navbar are the login, register and logout links. Those links will redirect the user to their specific component apart from logout which calls the logout property and redirects the user to the root.

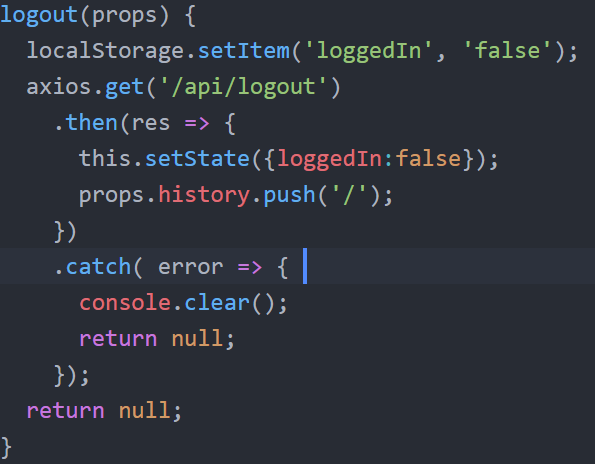


Figure 3 Logout function

* SubforumList.js – this component retrieves all the subforums from the database using an axios request and generates a list of Subforum objects passing all the key features obtained from the database (id, name, logo);
* Subforum.js – this is a stateless component which only displays in the browser the props given when instantiated in a bootstrap card;
* ThreadList.js – same as SubforumList, this component retrieves from the database a list. This time the list will contain threads that relate to the subforum the user is in. For example, if the user is in the Games subforum, only threads related to Games will be retrieved from the database. For each thread a ForumThread object is created and the data related to the specific thread is bound to it. If the user is logged in, an additional button will render in the browser allowing the user to access the CreateThread component.
* ForumThread.js – this component renders a bootstrap card containing some of the properties bound to it and a button that act as a link that redirects to a different route which will render the ThreadExpanend component. In case the user is logged in and has ever created a Thread, two extra buttons (Edit and Delete) will render on the threads created by him. The edit button will render the EditThread component which is very similar to CreateThread and the delete button will erase the thread.
* CreateThread.js and EditThread.js – those two components allow the user to create or edit a thread. The difference between those two components is minor, EditThread having its input fields already populated with the existing information. Another difference is that CreateThread does a POST request while EditThread does a PUT request.
* ThreadExpanded.js – this component renders all the information available in the thread. It also renders all the comments related to it, each comment being a different component. In case the user is logged in, a comment box and a rating system is also rendered. The comment box is an instance of PostComment and the rating system consist of two buttons (Upvote and Downvote) which will update the rating of the thread when pressed.
* PostComment.js – is the component that renders the textarea input which allows the user, in case is logged in to post a comment
* Details.js and EditUser.js – The second link next to Home is View Details which redirects the user to “/details”. This route renders the Details component displaying some user information. In order to edit those details the user can press the Edit button which will render the EditUser. Similar to the other Edit component, it displays a form list with the details the user can edit. It is worth noting that not all details are editable.
* withAuth.js – this middleware function prevents some of the components being rendered in case the user is not logged in. An axios request that checks if the browser has a token in its cookies is fired whenever a component is wrapped in this function. In case the request fails (status code is not 200), the user is redirected to the login page.

Other functionality includes:

* Login state persists on refresh due to the use of localStorage

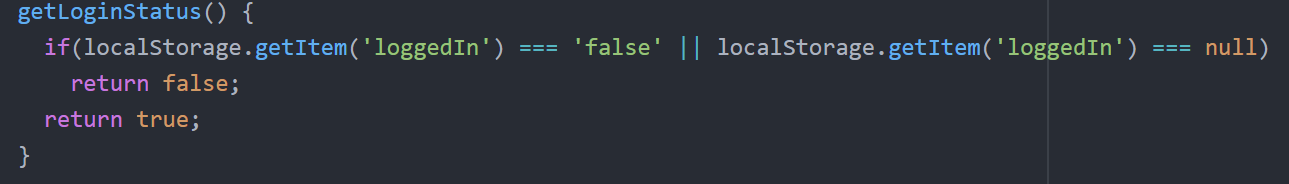


Figure 4 Usage of localStorage

When the page is refreshed the loggedIn value inside the state is instantiated with the result of this function.

* Some components use the props inherited when created in order to reduce the number of requests to the API. On page refresh, the props are erased and an axios request is done that will refresh the whole component. Without an axios request the component wouldn’t be able to display the props and will either throw errors or render nothing.

Most of the components where defined in this application, but in order to set up Routes and Links that redirects the user and render the exact component in the browser, the ‘react-router-dom’ library was used. From this library the project uses the Link (which manipulates the link in the browse), Route (which renders a component depending on the browser link), Switch (which helps with the transition between routes), BrowerRouter (which has to wrap all the other components provided in this library) and Redirect (which redirects to specific links).

Even if here are some custom CSS classes written in the app.css file (e.g. removing link underling), the styling of the project is done using the bootstrap framework. The lists would render simple cards containing the props of the component. Images are styled to be contained inside the cards using in-line styling. On different screens, the images used might look different, as the in-line style is not made to be responsive. The rest of the components are responsive, and the application should be usable on most devices.

## Node.js

Node.js uses JavaScript server-side code to create different application, mostly in the Web environment. This application uses Node.js to serve the Express API and to connect to the database using the “mongooses” library. When the server starts, a connection to the database is established using the link provided by the database host. The project database is hosted on Atlas and the project code on Heroku. Even if the server loads multiple libraries when it starts, the ones worth mentioning are:

* jwt – which creates a token composed of some user information (in this case, the email), and a custom string as a secret to help the server differentiate logged users from simple users;
* bodyParser – allows the use of req.body/params/send/status, etc.;
* bcrypt – encrypts a string using a salt value of type int; the bigger the salt value, the better the encryption; although the bigger the salt value, the longer it takes to encrypt. This library is used to encrypt and decrypt the user password;

# User stories

The following stories present a reason of why this application is viable:

1. As a not registered user, I want to be able to see the rating of the thread so that I can get a general idea of the authenticity of the information provided.
2. As a logged user, I want to be able to post comments in a thread section, so I can give my opinion on the subject.
3. As a logged user, I want to be able to edit my post, so I can fix any possible mistake.
4. As a logged user, I want to be able to change my password, so I can keep myself protected in case I shared it by mistake.

# 4. Reflections on the project

While developing this project I faced a lot of challenges, which mostly, I overpassed. In terms of coding skills, I improved my overall knowledge on MERN applications, I learned how to create a REST API, how to correctly call the API (using the right request), how to use mongoose and MongoDB, and how to use an authentication token. I also learned more about state management in react, how to use routes and how to protect components.

Because of the lack of knowledge I had at the beginning of the project, the code is not persistent and some functions even if they do the exact same thing, they are written differently. There is also a lot of code repetition which could have been avoided if I would have given the project more time.

There are some bugs I could not fix, and there might be some bugs I don’t even know about, as the application wasn’t tested properly.

In the end, I realised that my designing skills are not as strong as my coding skills and I still need to work on that.