Web Applications Course Project Assignment

Microblogging app “ublog” by Lorenzo Coronati

The app is a microblogging service built on a MERN stack: the back-end is built with Node.js and Express, the storage is on MongoDB, and the front-end is made with React.

The web service allows the system users to create and post small social media messages (up to 100 characters) on their personal page and allows other users to view these blogs.

The source code is available on GitHub: https://github.com/lorecoro/ublog

Back-end

The back-end was bootstrapped with the *express-generator* tool, but some of the unused stuff was removed.

The libraries used by the Node.js back-end are:

- *express* (to manage HTTP requests)
- *http-errors* (to generate 404 erros)
- *mongodb* (for the database connection)
- *mongoose* (for the collection models and methods)

The main application file is *app.js*; it provides the connection to the database (which is hosted on Mongo Cloud) and defines two main routes:

- */api* (for the back-end endpoints)
- * (which serves the front-end at */fe/build/index.html*)

The available endpoints are defined in */routes/api.js*:

- *GET /api/users* (fetches the users in alphabetical order)
- *POST /api/user* (log in and sign in)
- *GET /api/posts* (fetches the latest 20 posts in chronological order)
- *POST /api/post* (creates a new post)

The corresponding controllers are written in */controllers/controller.js*, and return data in json format:

- *getUsers*

- *postUser*
- *getPosts*
- *postPost*

The mongoose models correspond to the two collections of the database and are stored in two files:

- */models/post.js* (collection *posts*)
- */models/user.js* (collection *users*)

BE Setup

After cloning the repository, run `npm install` from the root folder; then the back-end app can be started with `npm start` and express listens by default to port 8080.

Front-end

The front-end was bootstrapped with the *react-create-app* tool, and lives in the */fe* directory.

The libraries used by the React front-end are:

- *materialize-css* (for some front-end effects)
- *react*
- *react-dom*
- *react-scripts*

All the javascript files are located in the */src* folder.

The main application file is *app.js*; it initialises the state, sets an interval of 2 seconds for two XHR calls, one for the */api/users* endpoint, and the other for the */api/posts*; the fetch API is used for the purpose.

The render method returns a *div* with two columns (materializecss is used): the left column calls the Left component, and the right one calls the Right component; both components are passed some props: relevant state elements, relevant methods, and one of the props defines what child has to be render on each column.

The Left component can either render a LeftUsers component, which loops in the users array and renders a LeftUser component for each of them, or a LeftPosts component, which shows one LeftRow component for each post in the array.

Both are preceded by a PaneHeader component that defines what button text (“Show all posts” or “Show all users”) must be shown at the top of the column.

The users are identified in the users list and in the posts list with an avatar that is provided by a free web service: https://avatars.dicebear.com/api/human/

The Right component can either render a LogIn component (used to log in or sign up) or a Form component (used to submit a new post), depending on the loggedIn variable stored in the state.

The LogIn component is used both for the login of an existing user, and for the signup of a new one. When an existing user name is typed, the submit button text will switch automatically from “Sign up” to “Log in”.

In all cases, a call to */api/user* will be made, passing an object with the name and the password to the back-end, which will return 200 when the user is authenticated, 201 when the user is new, 403 when the authentication fails.

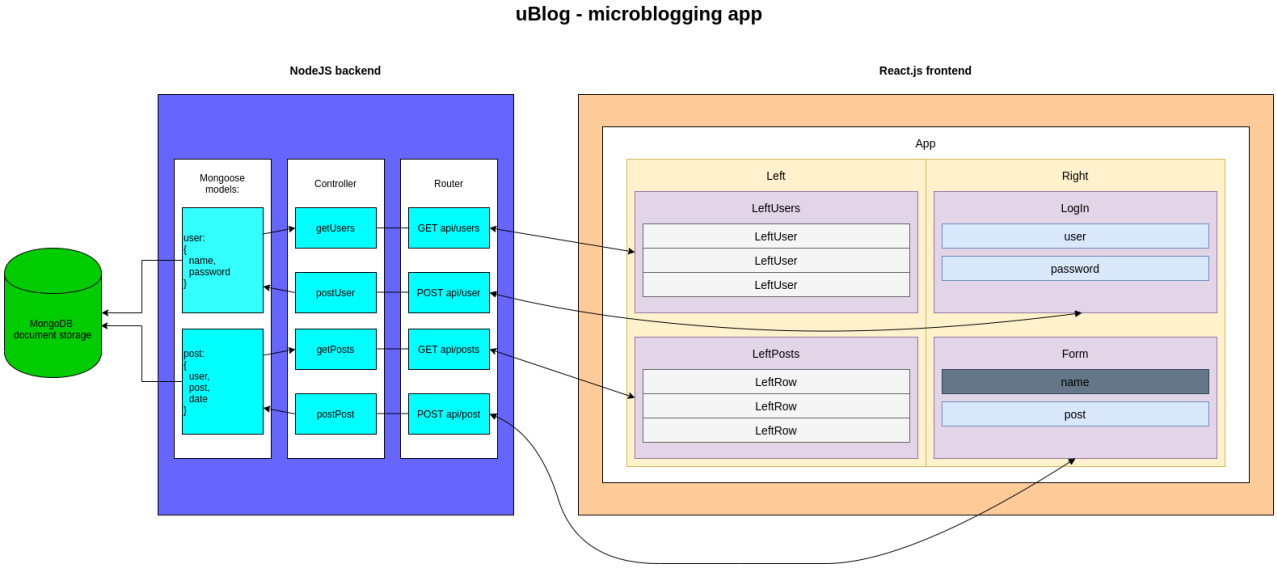
The logo in the header is a svg file taken from https://www.flaticon.com/authors/freepik

FE Setup

Enter the */fe* folder and run `npm install`; then the front-end app can be built with `npm run-script build` from the */fe* folder, and the generated files go into the */fe/build* folder.

Architecture

The program architecture is described in this figure:



Features and points

Required features:	Points:
Posting new posts	*
Support for multiple users	*
Viewing other users' posts	*
The app uses HTML and CSS	*
The system is implemented with Node.js back-end	*
The system uses *npm* package manager	*
The command that launches the Node app is `npm start`	*
The system stores data (posts and users)	*
The code is under Git version control: https://github.com/lorecoro/ublog	*
Minimum implementation points	30

Optional features:	Points:
The app uses React.js front-end	15
The app can run in a Docker container	5
The app can run in a Rahti pod	5
The app uses a MongoDB database	5
The app uses Mongoose models and methods	3
The front-end is responsive and uses Materialize	5
The front-end uses some animations (logo) and effects (character count and validation)	2
The app uses XHR calls (fetch API)	4
The app implements user registration, authentication, and password storage	5
Additional points	49

With the additional features, the total should exceed the maximum score of 60 points.