Проект по изборна дисциплина
Разработка на клиент-сървър (fullstack) приложения с Node.js + Express.js + React.js, летен семестър 2020/2021

Travel Vlog

Изготвил: Вероника Валентинова

Фак.номер : 26520
Специалност : РСМТ

Assignment 1	Project Summary
Course	Fullstack Application Development with Node.js + Express.js + React.js - 2021

Project author		
FN№	Name	Email
26520	Veronika Valentinova	vgvalentinova@gmail.com

Project name	Travel vlog

1. Short project description (Business needs and system features)

Travel Log is a platform for those who document and capture the world around us, helping people everywhere can experience the cultural and natural wonders of our planet and can feel more connected.

The system will be developed as a *Single Page Application (SPA)* using ***React.js*** as front-end, and ***Node.js + Express*** as backend technologies. Each view will have a distinct URL, and the routing between pages will be done client side using ***React Router***. The backend will be implemented as a ***REST/JSON API*** using JSON data serialization.

The main user roles (actors in UML) are:

- Anonymous User
  - can view the stories
  - search story by specific destination
  - leave comments on a post by providing name and email
- Traveler (Registered User)
  - can bookmark stories
  - can follow channels
  - can add and manage stories
  - can receive emails for new interesting stories - Receive one story a day, at a time of your choosing
- Administrator (extends Registered User)
  - can manage (create, edit user data and delete) all registered Users and stories

2. Main Use Cases / Scenarios

Use case name	**Brief Descriptions**	**Actors Involved**
2.1. Browse stories		All users
2.2. Search story by destination		All users
2.3. Leave comments		All users
2.4. Bookmark stories	Bookmarked stories are saved in Registered user's profile	Registered User, Administrator
2.5. Follow channels		Registered User, Administrator
2.6. Register	Anonymous User can register in the system by providing a valid e-mail address, first and last name, username and choosing password. Administrator can register new by entering User Data	Anonymous User, Administrator
2.7. Change User Data	Registered User can view and edit her personal User Data Administrator can view and edit User Data of all Users	Registered User, Administrator
2.8. Manage Users	Administrator can choose a User to manage, and can manage the chosen User - edit (using Change User Data UC) or delete. Administrator can create a new user using Register UC.	Administrator
2.9. Manage Stories		Administrator, Registered User
2.10. Subscribe for one story a day, at fixed time		Registered User, Administrator

3. Main Views (SPA Frontend)

View name	Brief Descriptions	URI
3.1. Home		*/stories*
3.2. User Registration		*/register*
3.3. Login		*/login*
3.4. User Data		*/profile*
3.5. Users		*/users*
3.6. View story		*/{userName}/stories/{story}*
3.7. View channel		*/channels*
3.8. View channel stories		*/{channel}*

4. API Resources (Node.js Backend)

View name	Brief Descriptions	URI
4.1. Users	GET *User Data* for all users, and POST new *User Data* (Id is auto-filled by *OKTS* and modified entity is returned as result from POST request). Available only for *Administrators*.	*/api/users*
4.2. User	GET, PUT, DELETE *User Data* for *User* with specified *userId*, according to restrictions decribed in UCs.	*/api/users/{userId}*
4.3. Login	POST *User Credentials* (e-mail address and password) and receive a valid *Security Token* to use in subsequent API requests.	*/api/login*
4.4. Logout	POST a logout request for ending the active session with *OKTS,* and invalidating the issued *Security Token*.	*/api/logout*
4.5. Read story	GET/PUT/POST/DELETE	*/api/stories/{story}*
4.6. View list of channels	GET	*/api/channels*
4.7. View channel	GET/PUT/POST/DELETE	*/api/channel/{channel}*
4.8. Follow/Unfollow channel	POST/DELETE	*/api/channel/{channel}/follow*
4.9. Bookmark	POST/DELETE	*/api/stories/{story}/bookmark*

1. Technologies

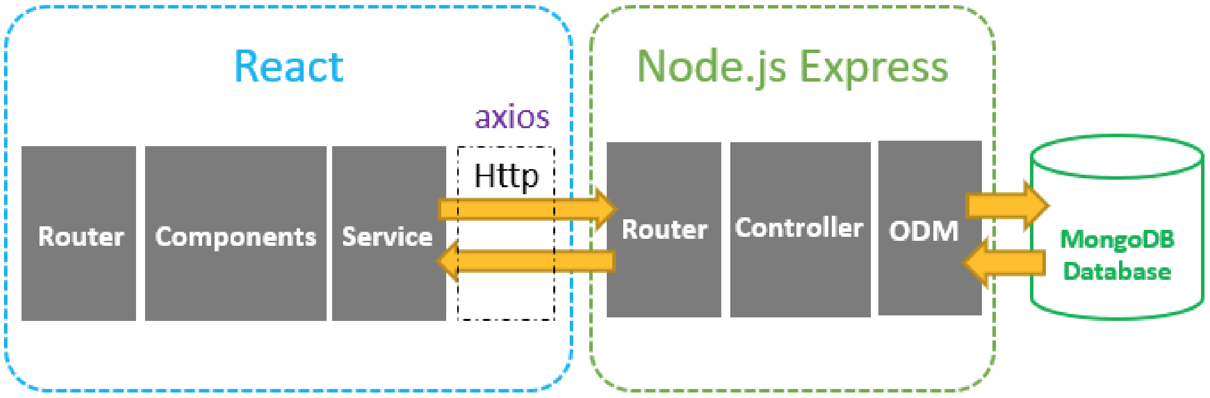
The back-end server uses Node.js + Express for REST APIs, Mongoose ODM

- Express 4.17.1
- bcryptjs 2.4.3
- jsonwebtoken 8.5.1
- mongoose 5.9.1
- MongoDB
- Cors 2.5.8

Front-end side is a React client with React Router, Axios & Bootstrap

- React 16
- react-router-dom 5.1.2
- axios 0.19.2
- bootstrap 4.4.1

2. Architecture



– Node.js Express exports REST APIs & interacts with MongoDB Database using Mongoose ODM.

– React Client sends HTTP Requests and retrieves HTTP Responses using *Axios*, consumes data on the components. React Router is used for navigating to pages.

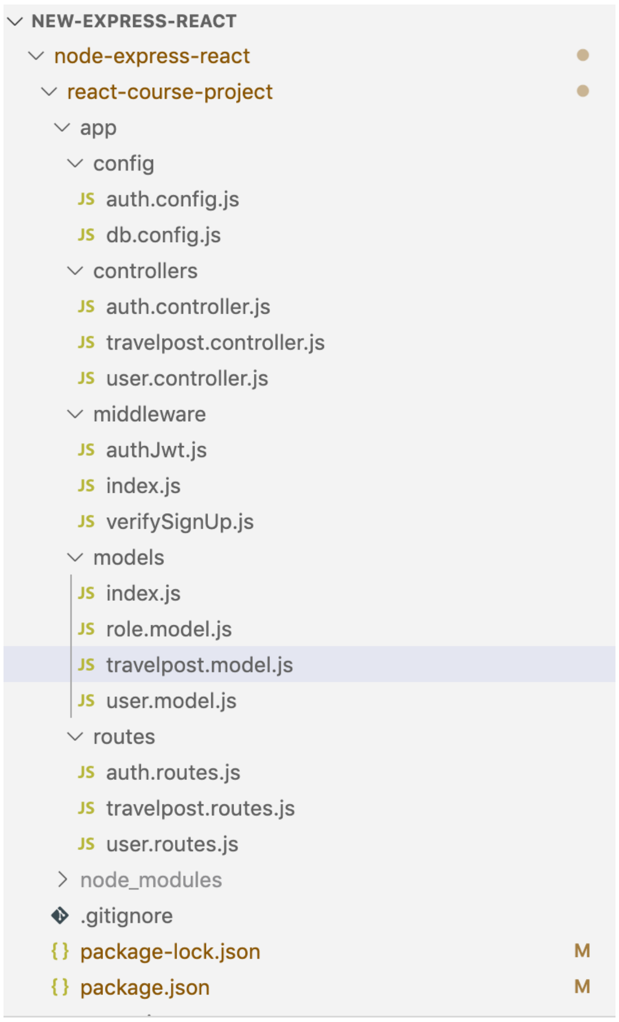
Via *Express* routes, HTTP request that matches a route will be checked by CORS Middleware before coming to Security layer.

Security layer includes:

- JWT Authentication Middleware: verify SignUp, verify token
- Authorization Middleware: check User’s roles with record in database

An error message will be sent as HTTP response to Client when the middlewares throw any error.

Controllers interact with MongoDB Database via *Mongoose* library and send HTTP response (token, user information, data based on roles…) to Client.



```
NEW-EXPRESS-REACT
  node-express-react                    ●
    react-course-project                ●
      app
        config
          auth.config.js
          db.config.js
        controllers
          auth.controller.js
          travelpost.controller.js
          user.controller.js
        middleware
          authJwt.js
          index.js
          verifySignUp.js
        models
          index.js
          role.model.js
          travelpost.model.js
          user.model.js
        routes
          auth.routes.js
          travelpost.routes.js
          user.routes.js
      node_modules
      .gitignore
      package-lock.json                 M
      package.json                      M
```

Inside **app/controllers** folder, *travelpost.controller.js* incudes these CRUD functions:

- create
- findAll
- findOne
- update
- delete
- deleteAll

- findAllPublished

models/*user.model.js*

```
const mongoose = require("mongoose");

const User = mongoose.model(
  "User",
  new mongoose.Schema({
    username: String,
    email: String,
    password: String,
    roles: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Role"
      }
    ]
  })
);

module.exports = User;
```

These Mongoose Models represents **users** & **roles** collections in MongoDB database.

`User` object will have a `roles` array that contains ids in **roles** collection as reference.

In server.js initially 3 roles are created: TRAVELER, ADMIN, MODERATOR

middlewares/*verifySignUp.js*

– check duplications for `username` and `email`

– check if `roles` in the request is legal or not

***middlewares**/authJwt.js*

- check if `token` is provided, legal or not. We get token from **x-access-token** of HTTP headers, then use **jsonwebtoken**'s `verify()` function

- check if `roles` of the user contains required role or not

Controller for Authentication

There are 2 main functions for Authentication:

- `signup`: create new User in database (role is user if not specifying role)

- `signin`:

- find `username` of the request in database, if it exists
- compare `password` with `password` in database using bcrypt, if it is correct
- generate a token using jsonwebtoken
- return user information & access Token

React project structure

– **package.json** contains 4 main modules: `react`, `react-router-dom`, `axios` & `bootstrap`.

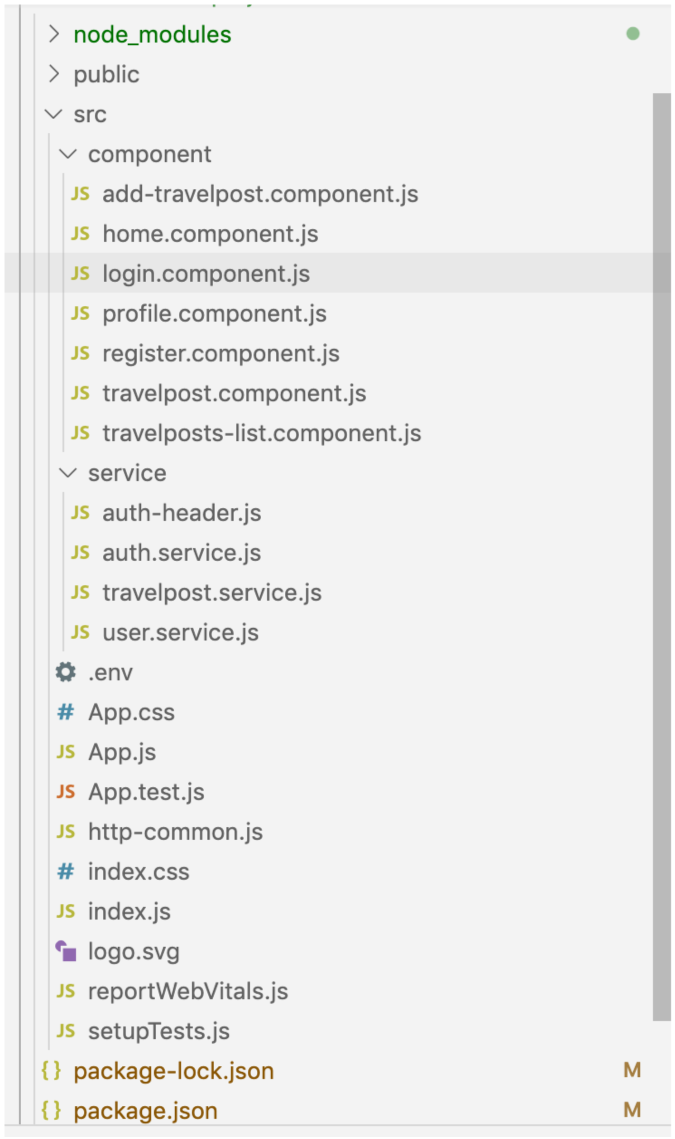
– `App` is the container that has `Router` & navbar.

– There are components for Travel Posts management, user login an registration.

– **http-common.js** initializes axios with HTTP base Url and headers.

– `TravelPostDataService` has methods for sending HTTP requests to the Apis.

– **.env** configures *port* for this React CRUD App.



```
> node_modules
> public
v src
  v component
     add-travelpost.component.js
     home.component.js
     login.component.js
     profile.component.js
     register.component.js
     travelpost.component.js
     travelposts-list.component.js
  v service
     auth-header.js
     auth.service.js
     travelpost.service.js
     user.service.js
   .env
   App.css
   App.js
   App.test.js
   http-common.js
   index.css
   index.js
   logo.svg
   reportWebVitals.js
   setupTests.js
 package-lock.json                                M
 package.json                                     M
```