|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Finished | Task | Grade 3 | Grade 4 | Grade 5 |
| ✓ | Create git repository for the lab | ✓ | ✓ | ✓ |
| ✓ | Have incremental commits in repository | ✓ | ✓ | ✓ |
| ✓ | Create a database with a table named “Users” | ✓ | ✓ | ✓ |
| ✓ | Add 4 users to the table | ✓ | ✓ | ✓ |
| ✓ | Create the path /admin | ✓ | ✓ | ✓ |
| ✓ | Add Roles for users and admin | ✓ | ✓ | ✓ |
| ✓ | Add roles for student 1, student 2, teacher |  | ✓ | ✓ |
| ✓ | Create path /student1, /student2, /teacher |  | ✓ | ✓ |
| ✓ | Render list of users in admin.ejs |  |  | ✓ |
| ✓ | Register new users with bcrypt encryption |  |  | ✓ |
| ✓ | Create dynamic route for user profiles |  |  | ✓ |
| ✓ | Add readme.md file |  |  | ✓ |

Lab 4 Login and Access Control

1. Database Related Files
   1. Database File (users.db)

Database Creating Statements

|  |
| --- |
| CREATE TABLE Users (  userID TEXT PRIMARY KEY  UNIQUE  NOT NULL  CHECK ( (userID != "") ),  name TEXT UNIQUE ON CONFLICT FAIL  CHECK ( (name != "") )  NOT NULL ON CONFLICT FAIL,  role TEXT CHECK ( (role != "") )  NOT NULL,  password TEXT NOT NULL ON CONFLICT FAIL  CHECK ( (password != "") )  ); |

* 1. Database Connection File (knex.js)

In this lab, I used a different database connection library called “knex”, to make the syntax of database manipulation code more concise and readable.

The database type (sqlite3) and path of the database file have been specified in this file.

* 1. Database Accessing / Manipulation File (database.js)

There are 5 database functions inside this file, all of which are exported into the module.

The 5 functions are:

1. getUsers()
2. getUser(username)
3. addUser(user)
4. deleteUser(username) (Not actually be used in this lab)
5. updateUser(username, user) (Not actually be used in this lab)
6. Server.js
   1. Middleware

* authenticateToken()
  + - 1. Check if currentKey (token) is an empty string, if so, redirect user to /identify with a 401 status code.
      2. Check if currentKey is corrupted (using jwt.verify()), if so, redirect user to /identify with a 401 status code.
      3. Store the decoded information into the request (req.user).
      4. Call next().
* authenticateRole(page)

1. Check if the role of current user is included in permitted roles for this page.
2. If current role is student, check if the username is correct. (user1/user2)
3. If either of the conditions wasn’t fulfilled, redirect user to /identify with a 403 status code.
4. If both of the conditions are fulfilled, call next().
   1. Endpoints

* /

Listen to GET requests and redirect the client to /identify.

* /identify

Listen to GET requests and POST requests.

Render static file identify.ejs after receiving GET requests.

When receiving POST requests:

1. Check whether the user exists (using getUser() function), if not, return the respond with status code 400 and render fail.ejs with an error message “There's no such user! Please register first!”.
2. Check whether the password is correct (using bcrypt.compare()), if not, return the respond with status code 400 and render fail.ejs with an error message “Incorrect Password!”.
3. If everything above is alright, generate a JSON web token using sign(), encrypt userID, name, role with the secret key.
4. Redirect to user’s personal profile.

* /register

Listen to GET requests and POST requests.

Render static file register.ejs after receiving GET requests.

When receiving POST requests:

1. Check whether either of username,password is empty string, if so, return the respond with status code 400 and render fail.ejs with an error message “The username and password shouldn't be empty!”.
2. Encrypt the password with bcrypt.hash().
3. Insert the user into the database using db.addUser().
4. If operation finished successfully, redirect to /identify. If not, return the respond with status code 409 and render fail.ejs with an error message "There's already a user with this username!"

* /admin

Render the admin.ejs after authenticate the token and current role.

Only admin users can access this route.

Show users list with all the information stored in the database.

* /student1

Render the student1.ejs after authenticate the token and current role.

Student1 him/herself, teachers, admins can access this route.

* /student2

Render the student2.ejs after authenticate the token and current role.

Student2 him/herself, teachers, admins can access this route.

* /teacher

Render the teacher.ejs after authenticate the token and current role.

Teachers, admins can access this route.

* /users/:userId

Check if there is not such a user, if not, return the respond with status code 404 and render fail.ejs with an error message “No Such User!”.

Check if current userID is identical as the parameter, if not, redirect user to /identify with a 403 status code.

Render the profile.ejs after authenticate the token.

* 1. Technical Problems & Solutions
* Forget to add return at the end of the if branches, the code keep running after it should be terminated.
* redirect() doesn’t work with status(), have to pass the status code as the first parameter.