**13/2/2023**

**Authentication module**

In this app, a ful l registration module and authentication middleware are to be implemented.

**authorization**

This will be done via JSON web tokens, basically, a token is generated and sent to the client in a cookie, this token holds some data such as role and any necessary yet safe-to-share information. This token has a timeout after which it won’t be valid.

This token is generated whenever a user logs in. and destroyed either on logging out or on its timeout.

**Registration**

This is straightforward, just the simple plan that came across your mind, however, there is some stuff to bear in mind.

1. Don’t store passwords as plaintext in the database.
2. Don’t store passwords as plaintext in the database.
3. Don’t store passwords as plaintext in the database.

**How to do so?**

You can do so by encrypting (or hashing) the password, then storing it in the database, note that it’s best practice to use one-way hashing, actually in most scenarios, you don’t need to decrypt it at all, on signing in, you can hash the incoming password and compare it, but there’s a much better approach.

**Bcrypt library:**

This library can encrypt the password for you, and instead of encrypting the password provided on logging in it can compare the plain text password with the encrypted one (it does all the dirty work for you), read [the documentation](https://github.com/kelektiv/node.bcrypt.js/) for further info.

**Usecase 1 registering: Flow**

1. User enters username\password to register,
2. The parameters to the backend.
3. The username is tested against the database.
4. The password is hashed.
5. user is stored in the database and success is returned.

**Usecase 2 signing in**

1. The user enters email &password to register.
2. If the email exists in DB, then passwords are compared (remember, bcrypt)
3. If the passwords match, then a JWT token is generated and sent in a cookie to the site.
4. Now the user is authorized for the next x hours unless he/she logs out.

14/2/2023 11:00PM

**What’s been done?**

**Summary**

* User model {username, email, password, role}
* Functions of (register, login, logout)
* Authentication middleware functions (AuthAdmin, AuthBasic) => they check whether the user is signed in and authenticate his role

**Methods**

Register: takes username, email, and password, handles any errors, then creates a basic user in the database

Login: takes email, and password. Authenticates the user and sends necessary data as JWT in a cookie.

Logout: destroys the token, the user is now unauthenticated

**Middlewares**

AuthAdmin: a middleware function to protect admin routes, admin routes cannot be accessed except by an authorized admin user. Works by fetching the JWT token from the client, validating it, then checking the role

AuthBasic: a middleware function to protect basic routes. Basic routes cannot be accessed except by an authorized basic user. Works by fetching the JWT token from the client, validating it, then checking the role

14/2/2023 11:40PM

**Features to be added:**

1. **reset your password.**
2. **Update registration: Confirmation via email.**

**Details**

Users can reset their password via email.

How its done?

* On requesting a new password. Take the email, compare it across the database, and if found, send an email to it, with a JWT containing the request needed details, on the server, we will get this token and decode it to verify it’s the same user, if so, passwords submitted on the form which is attached to the email are verified, then updated.

To prevent multiple password changes, a single-use JWT can be used. However, there is a [clever way](https://www.jbspeakr.cc/howto-single-use-jwt/) to convert the normal JWT token to a single use without a need to store it in the DB for example, this way enhances the performance, scalability, and security of the DB.

15/2/2023 03:40AM

**What’s been done**

Registration updated, now you must confirm via email in order to create an account. But it needs some modifications, repeated requests lead to many emails with the same token, it’s useless. You can track confirmation emails, or find a better approach.

17/2/2023

* How to update?
  + U can add a “confirmed” field to the user documents, and create another collection with key-value pair where the key is the JWT hash and the value is the user id (you can add a timestamp for scalability(remove outdated requests after a specific time instead of remove on checking)), when a user clicks confirm on the mail, the confirm method checks the tokens collection. If its within its time limits user is confirmed and this document is deleted, otherwise its deleted from the database along with its user and the user is asked to re-register.

17/2/2023:

-Mail confirmation Done.

-password reset Done.