**Stephen Bradley C15801268**

**Enterprise Application Development**

**Worksheet 1**

**Q1. Implement a users table having a username and hashed password fields. Use the postgresql crypt() and gen\_salt() functions to implement the password hashing**

**Implement a protected resource table (e.g. a “products” table) to which you can use to demonstrate your authentication features**

**A screenshot of a cell phone

Description automatically generated**

Create database called ‘lab2 ‘in psql shell and login.

A close up of a sign

Description automatically generated

Create a ‘users’ table as shown and a products table with columns for product name and price.

A screen shot of a social media post

Description automatically generated

Insert a row into the ‘users’ table with a username and a hashed password as shown.

A close up of a screen

Description automatically generated

Run the command SELECT \* FROM USERS; to make sure that the password has been hashed.

A close up of a black background

Description automatically generated

Insert some values into the products table that you created.

**Q2.** **Implement a JWT-secured version of the API based on the users table from the previous step. Your solution will implement the following API extensions**

**A (pre-authentication) login API call which accepts a username and password and returns (if successful) a JWT with a set of claims. The claims should include, minimally, the user id and an expiry timestamp; the token should be set to expire no later than 24 hours**

**A mechanism to verify client tokens as bearer tokens in a HTTP Authorization header field**

**Authentication should be applied, minimally, to any API calls which update any tables; Token validation should be performed on all API calls**

**Assume the client has a priori knowledge of the user password**

**Use asynchronous crypto in your solution**

**Demonstrate your JWT authentication on a protected resource**

**If authenticated or validated, the API return code should be in the 2xx range, otherwise 401.**

**A screenshot of a cell phone

Description automatically generated**

I used massive to access and query my database and configured it as shown.

A screenshot of a cell phone

Description automatically generated

I then made the /authenticate endpoint for users to login. It requires that the user enter a username and password that are within the users table in my lab2 database. If the wrong credentials are entered a message saying “error user not in database is returned. If the correct credentials are entered a java web token is returned as shown below.

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Create the /products endpoint so that if the user enters the correct login details, they can view all of the fruit in the products table. If the user enters the incorrect username or password return the message ‘incorrect username/password”.

Correct login details for protected products table:

A screenshot of a cell phone

Description automatically generated

Incorrect login details for protected products table:

A screenshot of a cell phone

Description automatically generated