Team Name: Code Ninjas

Github User ID of submitted assignment: oliviercm

Members:

Name	**SFSU ID**	**Github User ID**
Olivier Chan Sion Moy	913202698	oliviercm
Jasneil Nat	921273467	sikorskyhawk
Jinal Shah	913318697	Jinal-Shah-Design
Jingxing Luo	921266941	JingxingLuo

Description of work:

Updating User preferences (Jinal):
I worked on posting user preferences for email subscription choices. I found the backend portion of this class the most difficult. It took me a long time to get my environment up and running and ran into many challenges while trying to test my code. This was the hardest portion of building the website for me. Oliver was very helpful, I did the best I could but struggled with understanding the intricacies of backend development. Connecting the front end to the backend and all the layers in between were difficult for me to understand but I did learn a lot through this process.

Automatic Environment Variable Configuration (Olivier):

To be able to handle different deployment environments, the application retrieves MySQL credentials and the server's secret key for JWT signing from environment variables.

To aid in setting these environment variables, I created a handler which reads from a .env file and sets the included variables within that file.

For security reasons, the repository does not contain a .env file with sensitive information, but instead includes a .env.example file with the configurable keys, and non-sensitive example values.

MySQL table schemas (Olivier):

To aid in deployment, I created a script to create the necessary tables. I did not have the application set up the database because in a production environment, the application should not have permission to create or drop databases and tables.
Some notable choices of the table schemas are:

- User passwords (hashed) are stored in their own table. This doesn't provide much extra protection by itself when compared to simply storing passwords within the main user table, but this could be expanded with different application MySQL accounts and table read permissions in order to achieve more separation, making a potential attack just a bit harder.
- Some tables are not as normalized as they could be, for example, some columns in our tables are of the JSON type. I went with this route because it would make implementation easier as less changes would be needed. Also, since the JSON type is natively supported by MySQL, we still retain good functionality of our table since we can still query the contents of the JSON columns in a native manner.

JWTs (Olivier):

For authentication purposes, the application uses JWTs. Helper methods were created to aid in this implementation. The application generates RFC 7519 compliant JWTs, and can also handle JWT expirations - the example value in the environment file has the expiration time as 1 month.

Database Handler (Olivier):

To interact with the database, a helper class is used. Credentials are retrieved from environment variables.

In order to combat potential attacks, the application uses prepared statements. Since the prepared statements themselves are not dynamically generated and do not use dynamic column names, only field values, the application should be safe from injection.

For operations that require multiple queries (such as processing orders, creating new users), transactions are used to maintain consistency - transactions will be correctly rolled back in case of errors.

User Registration (Olivier):

For user registration, the backend will validate requests for missing/invalid fields, and disallow duplicate emails being registered.

In order to store user passwords, the password is first encrypted with bcrypt, then stored. This was relatively easy to pull off since PHP has native methods for password hashing/verifying.

User Authentication/Authorization (Olivier):

For user login, the backend will verify the sent email and password against the database - passwords are verified by comparing the stored hash.

User authentication is stateless - upon successful verification, we issue a JWT and CSRF token to the client. The JWT contains both the user's ID and their CSRF token, and is signed with our server's secret key. The JWT is sent as an HTTP-only cookie; because the JWT is stored as a cookie, in order to prevent CSRF, we also send the client a CSRF token in a custom header during login. Thereafter, the client must send all authenticated requests with both the user's JWT as a cookie and the user's CSRF token as a custom header - the backend will check for the existence of both of these, check the signature of the JWT, and compare the JWT's stored CSRF token with the CSRF token sent as a custom header.

To summarize, our authentication system is stateless, based off of JWTs, and uses the "Synchronized Token" pattern in order to mitigate CSRF attacks.

User-specific endpoints such as fetching cart, changing account details and making orders MUST be authenticated.

Database Product Storage (Olivier):

To store products in the database, the product schema was first normalized, then a script was written to insert products into the DB.

To fetch products, an endpoint was created which receives requests and responds with products - requests can either request all products by not specifying a product ID, or requests can request a specific product by specifying an ID.

Frontend implementation was relatively painless, as the initial implementation kept the eventual backend implementation in mind.

One change that did need to be made was the switch from prices being floats to prices being integers. In order to avoid imprecision in the database, prices are stored as integers (cents). This was relatively easy to update on the frontend.

Database Cart Storage (Olivier):

_____The application now stores the user's cart within the database, so that it can be restored when logging in. The frontend keeps the state of the users cart stored locally as a sort of cache, but also refreshes the stored state when loading a page so as to keep the frontend and backend synchronized as best as possible. Logging out will clear the cached cart.

Database Order Storage (Olivier):

_____When checking out, the application now stores that order in the database, which is viewable by users on a new "Order History" page.

Updating User Name, Email, Password, Preferences (Olivier/Jasneil):

_____The user preferences pages now send requests to the backend, and result in changes to the database. Just like all other user-specific endpoints such as fetching user data, order history, etc., these update requests must be authenticated.

Requests are validated for missing and invalid fields, invalid examples such as duplicate emails, passwords being too short, passwords not matching, etc.

(Trial)Updating User Name/Couple icons on checkout page(Jingxing):

I had tried to update the user name singly but it was too difficult to deal with the frontend and backend. Eventually, I got a little work to add some of the payment icons on the checkout page.

Problems Encountered and Solutions:

- (Olivier) This was my first time ever writing PHP, so I had initial hiccups when learning how to access properties and debug code. I found it particularly clumsy to debug code compared to Node.js since I could not figure out how to log and display output in real time, so my solution was to echo or print_r data and use Postman (curl) to make requests and view output.
- (Olivier) Initially, user authentication had a bug where some JWTs would not verify correctly. I eventually tracked this down to the URL safe base64 encoding that was being done on the JWTs - there was a bug which sometimes caused the encoded string to not decode back to the same one.
- (Jasneil) This part of the class was the most difficult. Setting up the environment to test the changes took several hours. Postman was useful to see if there was output. I learned alot about the relationship between php, MySQL and how the backend works. I worked on updating the user name but was off base alot. Olivier was extremely helpful in teaching me, helping me debug, and making sure the project turned out great.
- (Jingxing) The backend process was the most difficult part. I used Wampserver to run a server on windows and its tool to look at the database. It was hard because developing the backend needed to handle different applications to work. Olivier was great as he gave much information on how to deal with the connection between mySQL and the php.