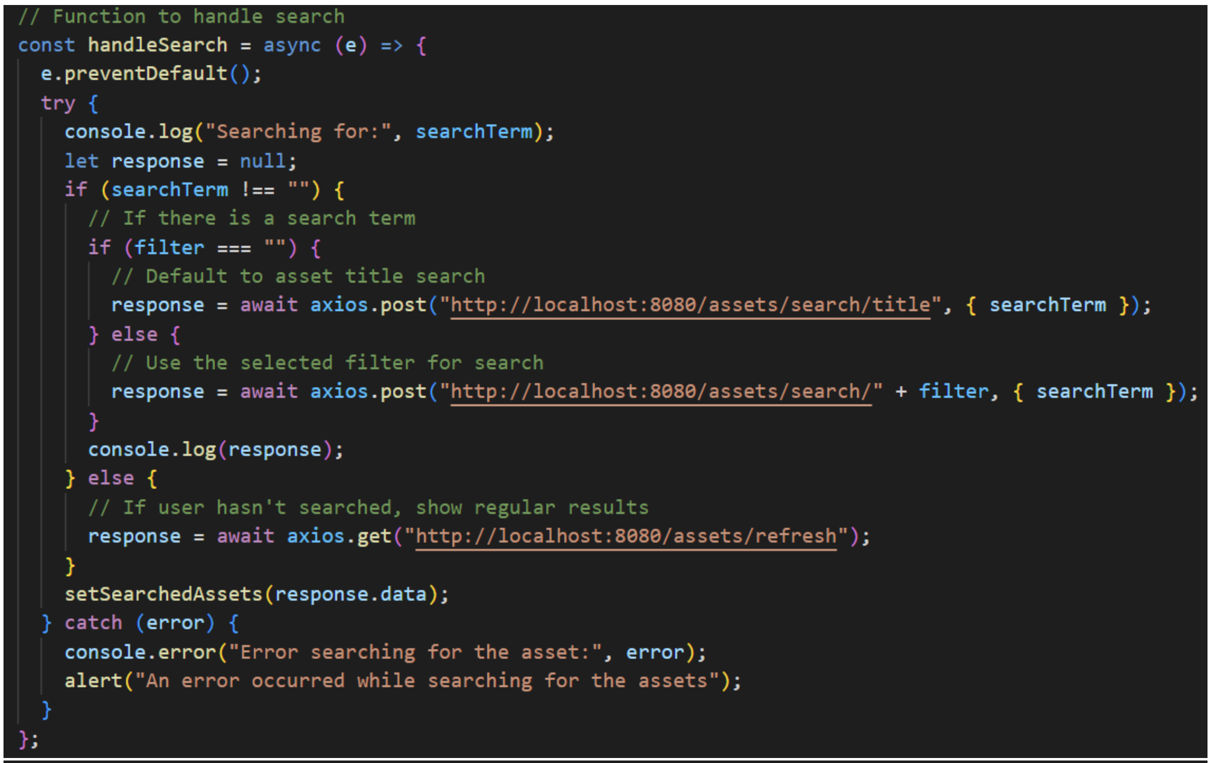
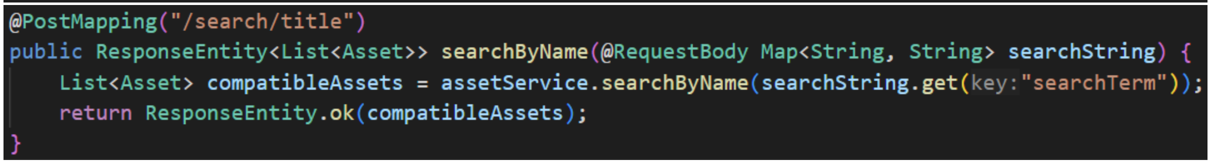
CS2815 (Team Project) | Individual Report

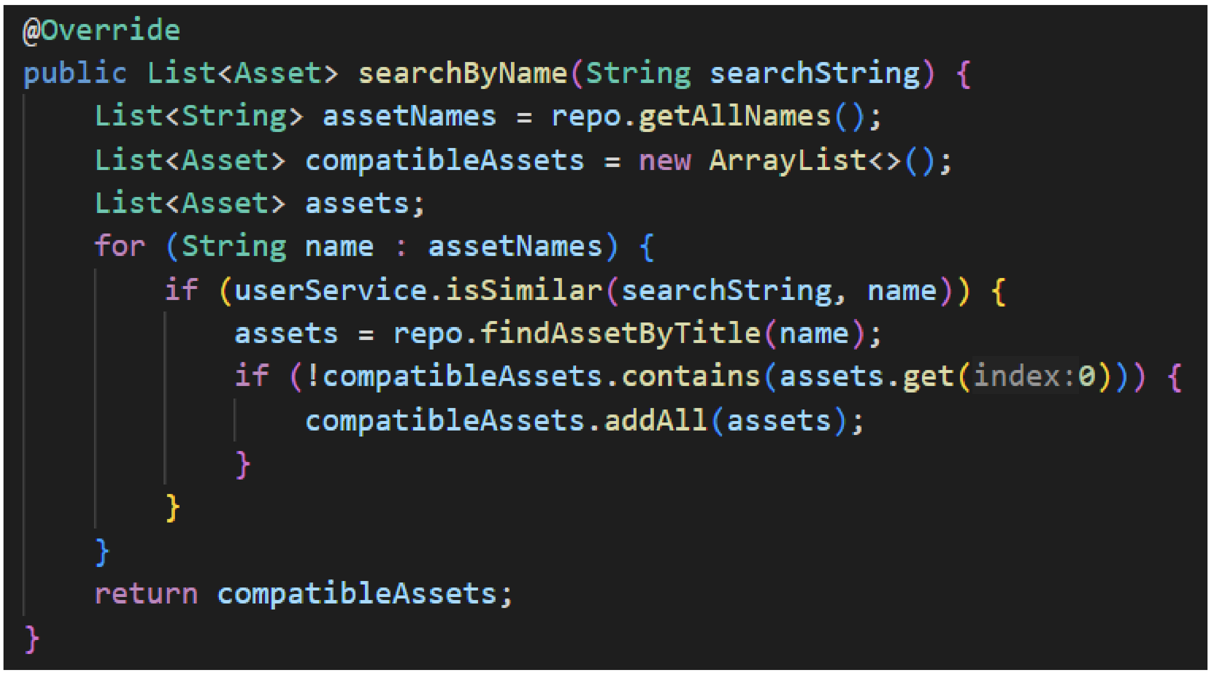
1.



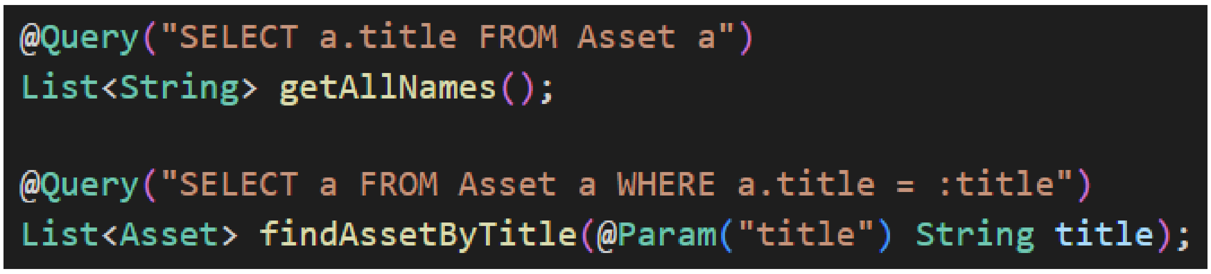
```
// Function to handle search
const handleSearch = async (e) => {
  e.preventDefault();
  try {
    console.log("Searching for:", searchTerm);
    let response = null;
    if (searchTerm !== "") {
      // If there is a search term
      if (filter === "") {
        // Default to asset title search
        response = await axios.post("http://localhost:8080/assets/search/title", { searchTerm });
      } else {
        // Use the selected filter for search
        response = await axios.post("http://localhost:8080/assets/search/" + filter, { searchTerm });
      }
      console.log(response);
    } else {
      // If user hasn't searched, show regular results
      response = await axios.get("http://localhost:8080/assets/refresh");
    }
    setSearchedAssets(response.data);
  } catch (error) {
    console.error("Error searching for the asset:", error);
    alert("An error occurred while searching for the assets");
  }
};
```



```
@PostMapping("/search/title")
public ResponseEntity<List<Asset>> searchByName(@RequestBody Map<String, String> searchString) {
    List<Asset> compatibleAssets = assetService.searchByName(searchString.get(key:"searchTerm"));
    return ResponseEntity.ok(compatibleAssets);
}
```



```
@Override
public List<Asset> searchByName(String searchString) {
    List<String> assetNames = repo.getAllNames();
    List<Asset> compatibleAssets = new ArrayList<>();
    List<Asset> assets;
    for (String name : assetNames) {
        if (userService.isSimilar(searchString, name)) {
            assets = repo.findAssetByTitle(name);
            if (!compatibleAssets.contains(assets.get(index:0))) {
                compatibleAssets.addAll(assets);
            }
        }
    }
    return compatibleAssets;
}
```



```java
@Query("SELECT a.title FROM Asset a")
List<String> getAllNames();

@Query("SELECT a FROM Asset a WHERE a.title = :title")
List<Asset> findAssetByTitle(@Param("title") String title);
```

How it works:

Using React, the frontend monitors for input into the search bar, selection of a filter type and clicking of the search button. Doing this will run the handleSearch command (image 1), which sends a command to the controller, depending on the filter type selected. The controller (image 2) communicates with the relevant service implementation function. The service implementation function (image 3) gathers data from the database, hosted on pgAdmin 4. The repo (image 4) gets the data through the use of the Spring Data JPA framework, which runs SQL commands to the database. The implementation then iterates through all the assets in the database, and checks if the name is similar to the search name. If it is, it adds it to the list and returns the list at the end.

How it contributes to the success of the project:

It provides functionality to the following requirement in the initial spec: “The system should support search, both keyword based and “filter-based” search that would retrieve all relevant information for an asset which is stored in the repository”. I am proud of this code as it elegantly, simply and efficiently provides this functionality, enabling a flexible asset management search system whilst keeping a log of the results.

With most search functions of the backend, they have been worked on by many people, however my contribution was helping Jay with the logic of the implementation, refining the implementation and enabling the front-end search capabilities.

2.

		Complexity (L/A/H)	**Function Point value**
EI	Story 1	L	3
EO	Story 2	L	3
	Story 3	A	5
EQ	N/A	N/A	N/A
ILF	Client Table Order Status Table Order Table	L L L	7 7 7
EIF	N/A	N/A	N/A
			UFC = 32

EI:

Story 1:

- FTRs: 1 (client table)
- DETs: 5 (name, date of birth, address, phone number and email address)

EO:

Story 2:

- FTRs: 1 (client table)
- DETs: 2 (date of birth, amount spent in the restaurant since registering the details)

Story 3:

- FTRs: 2 (order status and order tables)
- DETs: 6 (order ID, table number, time placed, time of last change, notes and corresponding order_status name)

EQ:

N/A – There is nothing of the theme of requesting information from external systems.

ILF:

As there is no 0 column in the RETs, I am working on the assumption that a table is an RET of itself.

Client:

- RETs: 1 (client)
- DETs: 6 (name, date of birth, address, phone number, email address, and amount spent in the restaurant since registering the details)

Order_Status:

- RETs: 1 (order_status)
- DETs: 3 (ID, name and description)

Order:

- RETs: 2 (order, order_status)
- DETs: 6 (order ID, table number, time placed, time of last change, notes and corresponding order_status name)

EIF:

N/A – There is nothing of the theme of machine readable interfaces to other systems

3.

I have learnt that the client plays a crucial role in the development of software. They must be kept up to date with relevant information regarding the functionality of the software (as opposed to behind the scenes which does not concern them). Our role as software engineers is to meet the client's requirements in a way that makes them the most satisfied with the final product, whilst also ensuring robust implementation along the way, as demonstrates by the Scrum process of regular review meetings.

I have also learnt about the efficiency of communication and organisation when working with a team. You must keep each other informed of only relevant information at all times, as to not step on anyone's toes and to stay streamlined in terms of progress for each version every sprint. Ensuring that the client's requirements are met on schedule. It is crucial to be concise and precise during communication, as shown throughout the project with the Scrum meetings and review meeting. The methodologies we adopted of regular review meetings with the client, and the use of Trello to transparently report progress on and status of the project according to the plan set out at the start of each sprint and the backlog, directly corresponds to section 3.1 of the BCS Code of Practice.

Finally, I have learnt to embrace criticism and alternative viewpoints of my teammates, to produce the best product for our client possible. This directly corresponds to section 2e of the BCS Code of Conduct.

4.

Software security plays a vital role in a successful and robust full stack application. There are five notable features that provide security for our asset management system:

1. We use RBAC (role based access) – Viewers, users and admins have access to different features, dependent on the necessity of features. For example, a viewer create assets and users cannot access the admin tab, which is home to various functionalities like seeing the log table.

2. We have prevented a bug which allows users to backtrack using the back arrow on their web browser after logging out. Also, some pages will prevent access through this method of navigation. This prevents exploitation by users who perform unexpected behaviour.

3. By hashing passwords and sending them back to the database hashed, we can maintain confidentiality and integrity of passwords. By using the Springboot password encoder, we ensure that any interception of the passwords will be useless, as it would be computationally infeasible to reverse engineer the raw password. Furthermore, any tampering and modification will be noticed as upon login, we compare the expected value and the actual value of the password received. If a user believes that their details are compromised, admins can reset their password in the admin menu.

4. Whilst typing passwords, they are hidden to ensure confidentiality from anyone screen grabbing or watching in real life.

5. We have protection against SQL injection via the use of JPA repositories provided by Springboot, which automatically use prepared statements to prevent these forms of attacks.