Software Testing Portfolio Supporting Document 2022: S2084384

LO1 Requirements

Functional requirements:

R1 a) All users should be-able to create an account.

b) All users should register with a unique email which does not exist already, in the correct format. If email already exists, the user should be prompted to log in. Otherwise, the email is invalid, and the user should enter a different one.

c) Users must enter their name, password and address to create their account.

R2 a) All users should be able to log in to their account, by entering their email address and corresponding password.

b) Users should be able to view their orders and search up specific orders.

R3 a) Users should be able to a place order.

b) Users should be able to choose from box 1 or box 2.

c) Users should be able to update an existing order

d) Users should be able to delete an existing order.

R4 a) Admins should have elevated access which other users do not.

b) Only admins should be able add, view and delete orders and users.

c) Admins should not be able to delete other admins' profiles.

R5 The data base should update when a user is added, deleted, their details updated. It should also update when an order is added, deleted or updated.

Measurable Quality Attributes:

R6 All endpoints should be accessed by users and should handle traffic with a quick response rate.

Qualitative Requirements

R7 All functional requirements have been met

R8 The system is robust.

R9 The user details are stored in a secure database and are only used for their purpose.

R10 The system performs well.

Level of Requirements

R1: Unit testing for validating user input for creating an account, integration testing for checking if email is unique.

R2: Unit testing for validating user input for logging in, integration testing for checking if user can view their orders and update their details.

R3: Integration testing for checking if user can place, update, and delete orders.

R4: System testing for checking if admins have elevated access, integrated testing for checking if admins can view all users and orders and can add and delete orders and users.

R5: System testing to test that the database changing when a user or order is added, deleted, updated

R6: This is a system performance test, which will test that endpoints are accessible and should handle traffic with a quick response rate.

R7: System Test as this can only be completed once the main functionality of the system is correct, i.e., all functional requirements have been met.

R8: Testing how robust the system is can only be done towards the end of the testing approach, therefore this is a system test.

R9: This is a unit test as its checking that a user password is being stored in an encrypted format.

R10: This can be done as a system test, as it checks how different end points perform when multiple users are using at same time.

Test Approach, Details and Appropriateness

The chosen testing approach is reactive and will combine functional testing with unit testing. This will ensure it meets the specified requirements and will test the endpoints, input validation, error handling and data integrity. These, combined, will ensure that the data stored will be correct and verified according to the required constraints. There will also be multiple performance tests, with small medium and larger loads, these will show how the system long the system takes to respond to endpoint requests.

There are many positives to the testing approach, including that the unit tests catch earlier bugs, such that more code can be written with ease since the existing code will be functioning correctly. However, edge cases have not been planned, since the requirements don't include any specific details about the inputs, therefore the approach will not be extensive.

LO2 Test Plan

To ensure sufficient testing and analysis, it is important to follow the principles of Chapter 3:

- Sensitivity: better to fail every time than sometimes
- Redundancy: making intentions explicit
- Restriction: making the problem easier
- Partition: divide and conquer
- Visibility: making information accessible
- Feedback: tuning the development process

It was also important to consider real-life scenarios when planning the tests, which highlighted that multiple users will be accessing endpoints at same time, thus performance testing should reflect this.

Test Plan

The test plan has been written in order of the requirements, not the order they should be carried out.

R1 User must be able to create an account.

Since the function of the system is for users to create accounts and place orders, this is a high priority requirement and there for rigorous testing must be carried out to ensure that it is robust enough and performs well. The principles of Chapter 3 indicate we should consider at least two different testing and analysis approaches as the requirement has high priority. The partition principle suggests that to ensure the account is valid the requirement should be decomposed such that parts b, c and d will be unit tested individually. By the feedback principle, if one of the unit test fails, and as it is due to user error, there will be an error code 400 which will allow the user to locate their mistake and solve it. The integration test for b, all users should register with a unique, correct email which does not exist already, can be tested by trying to register with an existing email. The user should not be navigated through but should instead get an error message thrown stating that a user already exists with that email, and they should be prompted to login, or register with a different email. The unit tests for c, users must enter their name, password and address to create their account, will involve leaving name, password and address field empty one at a time. Any of these errors should throw error code 400, indicating a client error.

R2 User must be able to login to their account. Users should be able to view their orders and search up specific orders. Users should be able to update their personal details.

Once R1 has been met, R2 can partially be tested, and this can be tested by the partition principle and by unit and integrated tests. Firstly, using scaffolder code to create a user with existing orders, an integrated test should be implemented to test that a user can login and then view their orders. Secondly, this requirement can be checked by trying to login again with the same email address but an incorrect password. In this the user should not be able to login and a 401 error should be displayed. This determines that the users' details have been validated and the user can continue to

create an order. Unit tests should be carried out to check that the user is able to search up their own order, search up specific orders, and update their user details.

R3 Users should be able to place an order, select from two options, and should be able to update or delete existing orders.

This is a system test as, before this stage, the user must have created a valid account. To avoid redundancy in testing this requirement can be met within two integrated tests. The first test will add an order of box 1, then delete it. Then the user should search up all the previous orders to find that the one that was deleted, should now be removed from list of previous orders. The second test will be creating an order of box 2 and then changing it to box 1. Through searching up the list of previous orders, it will show that the order which previously had box 1 now has box 2. This tests all four aspects of the requirement and the scaffolding code needed will be that a user has been created and added to the database.

R4 Admins should have elevated access which other users do not. Admins should be able to add and delete orders and delete users. Admins should be able to view all users and orders, current and posted, whereas regular users should only have view of their own orders. Admins should not be able to delete other admins' profiles.

This is a high priority requirement which requires system testing therefore, by the principles of Chapter 3 indicate we should consider at least two different testing and analysis approaches. This also a safety requirement as admins have access to all users' personal data, if it was available to all users then this would be open to a potential data breach and be breaking the Data Protection Act 2018. By the partition principle, this testing of this requirement should be split into two parts. The first is testing if the admin can carry out each of these actions, and the second is testing if a regular user can carry out these actions. A series of unit tests should complete the testing of this requirement on the admins' behalf. These are detailed below. A test which checks if an admin can add an order. A test which checks if an admin can delete a general user. A test that checks that an admin cannot delete another admin. A test that checks admins can see all users. A test that checks admins can see all orders.

From the testing in R2 clearly user will be able to search up their orders, and update their details, so to avoid redundancy in testing, these will not be checked again. To test a user cannot do this for other users, there will be some unit tests, one which checks if a user can update another's details and one which check is a user can search the database for orders which are not their own. Another test to be carried out is that a user cannot view other users' details.

R5 The data base should update when a user is added, deleted or their details updated. It should also update when an order is added, deleted or updated.

This is a system level test, therefore a high priority test, and after carrying out the above tests, we know that these should all be passing, since we know that users who have created an account, can login into it, can add, search, update and delete orders. The testing and analysis for this test must be carried out in such a way that there is 100% certainty that this requirement is met. By the principle of redundancy, based on the priority of this requirement, there will be explicit tests carried out to ensure there are no issues with the functionality of the database itself, and if there is to isolate these specifically.

The first step of this requirement test would be to add two users, a regular user and an admin. Then search the database using the user ID, if a user exists then the test has passed.

Next would be to test if a user was being deleted, by first adding a user, then deleting using the user ID, searching the database to see if a user existed with that user ID and if there is no matching user, the test has passed.

To test if an users' details can be update, by first adding a new user, then changing the name, role, password and address. It is important to test if each of these details can be updated, since a user may want to do this. Then by searching the database and checking if the specific field is equal to the change. By the sensitivity principle, if one detail cannot be updated, this test will fail, and further inspection will be needed.

To test if an order can be added, firstly create an order, then search the database using the order ID to see if one exists.

To test if an order can be deleted, firstly add an order, then delete it, then search the database by the order ID to see if it still exists.

To see if an order can be updated, again using the sensitivity principle such that if either the box type or the description cannot be updated, the test will fail, firstly add an order, make the changes and then search the database by order ID and ensure the fields equal the new details.

R6 All endpoints are accessible and should handle traffic with a quick response rate.

An integrated test can be written which calculates the success rate of endpoints returning a response status of 200, signalling that each end points does what it is supposed to do, e.g., logging in, creating user etc, in an appropriate time.

R7 All functional requirements have been met

If the tests for requirements R1 – R5 have been carried out, passed or corrected, then this requirement can be considered as met.

R8 The system is robust.

R9 The user details are stored in a secure database.

The users' passwords should be stored securely, by storing a hashed version of the password. To test that this is secure, a test should be carried out comparing the login password to the hashed password, and using the 'bcrypt' library, will pass the unit test if they are the same. If the password is valid the user should continue to the login process otherwise the 401 error will be returned because the user is invalid.

R10 The system performs well.

Whether this requirement should be based on both the clients' feedback and the performance tests, such as detailed above in R6. This can also be done by load testing, by testing how the system handles different amounts of users at once and completing actions, such as logging in, within an appropriate time limit.

Process and Risk

In this section you think about where to place the tasks you have identified in the lifecycle process. To do this you really need to estimate how long the task will take (we won't go into that here – it is a big topic).

Since time is a limited factor as hours have been budgeted for the testing of this project, I have chosen to follow a waterfall life cycle to ensure that as any aspects of the code are tested as possible within the time constraint. Firstly, unit tests will be instrumented to ensure that the system-depending functions, such as creating a user, are isolated and working well. Security testing is included in the unit tests, then integration testing can be carried out, followed by system testing and finally performance testing. These tests will check what actions the simple user can carry out and what actions the administrators can carry out, according to the requirements above. The performance testing will be the last stage of the process since testing how the entire system performs is more informative than isolated performance testing.

Risks is that if data was stolen from database, the passwords could be stored with associated emails, and used for fraudulent purposes. Therefore, it is imperative to carry out security tests that will ensure passwords are encrypted in the database.

Another risk is that simple users will have the admin only accessible features, therefore this should also be tested extensively. This ensures that only approved admins have access to all information stored in database, and simple users only have access to their own.

Furthermore, another risk is that to maintain an equal level of control among admins, it is important that admins can't delete other admin accounts. This ensures that no admins with an alternative agenda put the company at risk by taking 'sole control' of the API and deleting other admins.

One risk is that synthetic data is not truly representative of real-life data, for example some addresses can be long, therefore the design may perform poorly upon creation of the API.

Evaluation of Instrumentation

It is worth noting that all the tests compare the status code for each request with the expected code, and not the actual response data. While the testing is accurate, if the system was to grow, it would be worth developing the tests such that this is not the case.

Table displaying expected error codes and corresponding reasoning.

Error Code	Example Reasons	Error Message
200	When a user has entered the correct login, when a user or order has been deleted.	"OK" "Login Successful"
201	When a user or order has been added or updated successfully.	There is no message as request has been successful.
400	If the email given is not in correct format.	"Email in bad form" "No User Found"

		"Bad Request" "No user associated with order"
401	This is shown when a password has been entered on login that does not match the password saved.	"Invalid Password"
403	If an admin attempts to delete another admin.	"Unauthorised Access"
404	When logging in or searching up an order, if the record doesn't exist in database.	"User Not Found" "No Order Found"
409	When registering a new account with an email that already exists in the database.	"Bad Request" "User Exists"

LO3 Testing Techniques

Range of Techniques

Unit testing was appropriate to locate small errors within the code, earlier on, so that only bigger errors were in the system tests. Integration testing worked well to ensure that elements of the code which interact, were able to do successfully, for example when creating and updating orders. After the unit and integration testing were returning successful results, it meant that system testing was able to test how all different elements of the code were interacting and running successfully with each other. The security testing was also needed as it ensured that the user's passwords were being stored securely. Finally, performance testing was necessary test how well the system was performing.

Evaluation Criteria

The criteria that will be used to evaluate the testing is

- Test coverage: This will define how much of the code is being checked when the tests are being run, normally a higher test coverage alludes to a higher test completeness, however, does not imply code is bug-free.
- Test case design and completeness: This refers to the writing of tests such that they check for how the system deals with all inputs and outputs including edge cases, relevant scenarios and unexpected data.
- Test case execution and test case pass rate: This criterion refers to running the test and comparing expected results to the actual results, and what percentage of tests pass or fail.
- Test efficiency: This refers to how quickly the tests can be run, where a higher efficiency is beneficial as it allows for fewer delays in the development process.
- Test maintainability: This is the measure of how maintainable the tests are and how easy it will be in the future to alter the code, where the more maintainable the tests are, the better.

- Test reliability: The test should be reliable and therefore return the same results every time.

Results and Evaluation

We can conclude that users will create validated accounts, where their passwords will be stored securely in the database by being encrypted, and can log in to their accounts, they can also update their name. Users can create, update and delete orders successfully. Admins have elevated access, but all share the same level of control and cannot delete other admin accounts or create orders for simple users. Admins can update and delete users and orders. Furthermore, users can search up their own orders only, while admins can search up all orders.

The performance test that was carried out for R6, while the results show that the system performs well under 100 get requests to all endpoints in rapid succession, with a timeout of 100ms, it is worth noting that the test is checking the success rate and not the actual time it takes to get a response. As aforementioned, it would help to improve the performance testing to carry out load testing, such as detailed in the test plan for R10.

The test coverage is 100%, as seen below, which suggests that while the program may not be bug-free, every line is being executed and since all the tests are passing and are reliable, suggests that the system is functioning correctly. However, it is notable that this conclusion can only be drawn if the test case design, execution and pass rate are of a high level. The testing lacks the checking of edge cases and ideally there would be many tests were there is user input, to ensure that any unexpected inputs are caught. It can be argued, however, that more information on the user interface is needed before assuming what can be expected. The test efficiency is very high, with the test suites running within 5 seconds. While the maintainability of the tests is very high as the tests make use of helper JavaScript files, which flood the database with several users such that the tests written further can simply test using these details.

```
----------|---------|----------|---------|---------|-------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------
All files |     100 |      100 |     100 |     100 |
 order.js |     100 |      100 |     100 |     100 |
 user.js  |     100 |      100 |     100 |     100 |
----------|---------|----------|---------|---------|-------------------

Test Suites: 6 passed, 6 total
Tests:       143 passed, 143 total
Snapshots:   0 total
Time:        5 s
```

File		Statements		Branches		Functions		Lines	
order.js		100%	4/4	100%	0/0	100%	0/0	100%	4/4
user.js		100%	5/5	100%	0/0	100%	1/1	100%	5/5

LO4 Limitations

Targert Performance and Security Levels

The performance level of the system should be moderately high, the test implemented aims for a 95% rate of the system dealing with 100 users at endpoints within 100ms. Performance tests are used to measure the responsiveness, stability and scalability of a system.

While implementing the performance test, I have realised that mostly the delay time is below 10ms, so it may be beneficial to firstly calculate the average delay time, then set an expected rate thereafter. Since performance targets are normally set based in expected usage, such as the typical number of users, this was not implemented as detailed as would have hoped.

The level of security testing needed depends greatly on the data that will be stored. In this system personal details are stored and for ensuring the safety of users there is no target level, although extensive testing should be carried out. Security testing is typically used to ensure that the system is protected against attacks. The security testing implemented in this test suite is unit level and is very minimal when it comes to identifying security issues.

When testing the security, the target is that the system is protected against attacks. The security test written only ensures that the users are protected if the system is attacked, using the 'bcrypt' library ensures that if the database is breached the attacker won't have access to users' real passwords, only encrypted versions. This is weak point of the testing and further security testing would be beneficial to ensure that the system is also protected against attacks