LO_1

Require document ---------- Page 2

1.1 notes---------- Page 4

1.2 notes---------- Page 6

1.3 notes---------- Page 8

1.4 notes---------- Page 9

For 1.1, I covered functional/non-functional requirements in the ***requirement document*** and explained in detail considering the different stakeholders' perspective in the ***notes***.

For 1.2, I covered the system/integration/unit level requirements in the ***requirement document*** and explained in detail in the 1.2 ***notes***.

For 1.3, in the 1.3 ***notes***, I covered a wide range of testing methods and chose some that I think is the best fit for the project and discussed how to implement the chosen methods. The testing requirements in the ***requirement document*** also gave some examples what some of the testing techniques are.

For 1.4, I analysed the pros and cons of the chosen techniques and wrote a summary how the chosen techniques can ensure adequate testing in the 1.4 ***notes***.

Requirement Document

API for User Registration, Authentication, and Order Placement

Introduction

This requirement document outlines the specific requirements for the API for User Registration, Authentication, and Order Placement. The API will allow users to register with the system, authenticate with the system, and place orders for predefined boxes at an imaginary food shop. The API will be built using JavaScript and the Node.js engine and will use the MongoDB database and Mongoose package for database operations. The system will utilize JWT for authentication and Bcrypt for password encryption. The API will be tested using a combination of manual and automated testing approaches, and will be evaluated for performance, scalability, security, usability, and other non-functional requirements. This requirement document is intended to provide a clear understanding of the project scope and goals, and to guide the development and testing of the API.

Functional Requirements:

The API should allow users to register with the system by providing their email address, password, and other necessary information.

The API should allow users to authenticate with the system using their email address and password.

The API should allow users to place orders for predefined boxes (Box1 and Box2).

The API should allow users to view their order history.

The system should have two user access levels (Admin and User), with the administrator having greater permissions and the regular user having restrictions.

Detailed requirements from Users:

The ability to register and authenticate users

The ability to place orders for boxes

Information of orders (confirmation, track order, rough time estimate)

Privacy

Manage personal information

Low resource usage (memory, CPU power)

Detailed requirements from Administrators:

View/ have access to all users and orders that have been placed

Be able to delete user or order

Update the status of the orders

Low resource usage (memory, CPU power)

Non-Functional Requirements:

Measurable quality attributes:

The API should be secure, with password encryption and measures to prevent unauthorized access; the system should also be able to handle injection attacks.

The API should be scalable, able to handle increasing workloads without a decrease in performance.

The API should be reliable, with a long mean time between failure.

The API should be available, with a high proportion of the time with the system operating properly.

The API should be repairability, with short mean time to repair so the service is running again.

The API should be supportable, the system should be able to accommodate new products and product lines without major reengineering.

Qualitative requirements:

The API should be user-friendly, easy to use and intuitive for users.

The system should be elegant, well-designed, and aesthetically pleasing.

The system should be simple and straightforward, with a minimal number of features and options.

The system should be flexible and adaptable, allowing for customization and the addition of new features or functionality.

Level of requirements

System requirements

System requirements refer to the hardware, software, and other resources that are needed to run the system. For this project, the system requirements include a computer with the Node.js engine and the MongoDB database installed, as well as any other dependencies or tools specified (such as the Mongoose, JWT, Bcrypt, Express, Jest, and Artillery packages).

Integration requirements

Integration requirements refer to the requirements for integrating the system with other systems or components. For this project, the integration requirements include the need to integrate the system with the MongoDB database and any other external systems or components that the system may interact with.

Unit requirements.

Unit requirements refer to the requirements for individual units or components of the system. Unit testing for this project would involve testing the individual units or components of the system to ensure that they function correctly and meet the specified requirements. This include testing the API endpoints, the database schema and operations, and the authentication and authorization mechanisms.

Testing Requirements:

The system should be thoroughly tested using a combination of manual and automated testing approaches.

The system should be tested for performance and scalability using load testing tools such as Artillery.

The system should be tested for security vulnerabilities and measures to prevent unauthorized access.

The system should be tested for usability and user experience.

The system should be tested for functionality and the correct implementation of all required features.

Learning outcome 1 notes

1.1

Range of requirements:

The range of requirements for this project includes all the functional and non-functional requirements specified for the API. In order to be as complete as possible, we can first analyse the stakeholders and generate requirements from different perspectives.

Stakeholders:

Users

Usability (nice UI)
The ability to register and authenticate users.
The ability to place orders for boxes.
Information of orders (confirmation, track order, rough time estimate)
Privacy
Manage personal information.
Low latency when using the app (little wating time)
Low resource usage (memory, CPU power)
Ensure that the app can perform the functions required.

Administrators

Usability (nice UI)
View/ have access to all users and orders that have been placed
Be able to delete user or order
update the status of the orders
Privacy
Security
Low latency when using the app (little wating time)
Low resource usage (memory, CPU power)

The software developers

The software is able to deal with high demand.
App is easy to maintain and add new features.

Government

Protect Privacy of civilians
Make sure this software is not used for illegal purposes.

General public

Doesn't spread virus.

Advertisers

The amount of people using the app
How often do they use the app

Functional requirements:

The functional requirements are the specific features and functionality that the API is expected to provide.

The functional requirements in this project are as follows:

For users:

Register
Login
Manage personal information
Make an order
Access personal order
Delete personal order.

For administrators:

Register
Login
Access all user's information
Get information of a specific user
Delete a user
Get users' order information
Add an order for him/herself
Update order for everyone
Delete order

Measurable quality attributes:

Measurable quality attributes are characteristics of the system that can be quantitatively measured and evaluated.

Measurable quality attributes for this project might include:

API's Performance: The performance of the system, such as the response time of the API and the throughput of the system.

Scalability: The ability of the system to handle increasing workloads without a decrease in performance.

Security: The security of the system, including measures such as password encryption and the prevention of unauthorized access. Resistance to injection attacks is also a part of it.

Reliability: The reliability of the system, including the frequency and severity of defects and failures. (e.g., uptime; mean time between failure).

Availability: The system should have a high proportion of the time with the system operating properly.

Repairability: The system should have short mean time to repair so the service is running again [MTTR].

Qualitative requirements:

Qualitative requirements are non-functional requirements that describe the desired characteristics of the system, but which are more subjective and cannot be easily quantified.

Qualitative requirements for this project include:

User-friendliness: The system should be easy to use and intuitive for users (user experience).

Elegance: The system should be well-designed and aesthetically pleasing.

Simplicity: The system should be simple and straightforward, with a minimal number of features and options.

Flexibility: The system should be flexible and adaptable, allowing for customization and the addition of new features or functionality.

1.2

Level of requirements

There are several levels of requirements that can be identified for a project, such as system requirements, integration requirements, and unit requirements.

Unit requirements.

Unit requirements refer to the requirements for individual units or components of the system. Unit testing for this project would involve testing the individual units or components of the system to ensure that they function correctly and meet the specified requirements. This include testing the API endpoints, the database schema and operations, and the authentication and authorization mechanisms.

Unit testing for this project would involve testing the individual units or components of the system to ensure that they function correctly and meet the specified requirements. This include testing the API endpoints, the database schema and operations, and the authentication and authorization mechanisms.

Integration requirements

Integration requirements refer to the requirements for integrating the system with other systems or components. For this project, it would include ensuring that the system integrates with the MongoDB database, that the API integrates with the database and any other external systems or components, and that the authentication and authorization mechanisms integrate with the API.

Integration testing for this project involve testing the integration of the various components and systems that make up the project. This include testing the integration of the API with the database, the integration of the authentication and authorization mechanisms with the API, and the integration of any external systems or components with the project.

System requirements

System requirements refer to the hardware, software, and other resources that are needed to run the system. For this project, the system requirements include a computer with the Node.js engine and the MongoDB database installed, as well as any other dependencies or tools specified (such as the Mongoose, JWT, Bcrypt, Express, Jest, and Artillery packages). It should also include compatibility with specific operating systems, reasonable memory and storage usage.

System testing for this project would involve testing the entire system as a whole to ensure that it functions correctly and meets the specified requirements. This include testing the API, authentication and authorization mechanisms, database operations, and any other functionality of the system.

System level requirements also include performance requirements, security requirements and robustness requirements.

Performance requirements

Performance requirements refer to the requirements for the system to perform well in terms of speed and efficiency. Performance requirements for this project include quick response times for API calls, efficient database operations, and efficient message parsing. For this project, performance testing would entail putting the system through various loads and stress situations to confirm that it can manage the estimated volume of users and transactions in a production environment. This is possible with tools like Artillery.

Security requirements

It is the requirement for the system's security and protection against potential security vulnerabilities. Security requirements for this project include the need for the system to be safeguarded against injection attacks, unauthorised access, and data breaches. For this project, security testing would entail scanning the system for potential security vulnerabilities with tools like OWASP ZAP and implementing test cases focused on discovering and addressing security concerns.

Robustness requirements

The requirements for robustness refer to the system's ability to withstand failure. Robustness requirements for this project include the system's need to continue functioning correctly in the case of component or service failures. For this project, robustness testing would entail testing the system under various failure scenarios, such as simulating connection failures with MongoDB, to ensure that it can manage unexpected conditions and continue to work correctly.

1.3

Identifying test approach for chosen attributes

There are many different test approaches that can be used, depending on the needs and goals of the project. Some common test approaches include:

1.Manual testing: Manual testing involves executing test cases manually, without the use of automation tools. This approach is useful for testing the usability and functionality of the system, and for testing scenarios that are difficult to automate.

2.Automated testing: Automated testing involves using tools or scripts to execute test cases automatically. This approach is useful for increasing the efficiency and reliability of the testing process, and for executing a large number of test cases.

3.Acceptance testing: Acceptance testing involves evaluating the system to determine whether it meets the acceptance criteria for the project. This approach is often used to determine whether the system is ready for release.

4.Performance testing: Performance testing involves evaluating the performance and scalability of the system under different load conditions. This approach is useful for identifying performance bottlenecks and ensuring that the system can handle the expected workload.

5.Security testing: Security testing involves evaluating the security of the system to identify vulnerabilities and ensure that it is secure. This approach is important for protecting sensitive data and preventing unauthorized access to the system.

Among them, we may choose the following method to test this API.

Manual testing: We can test the API with different user roles or testing the security of the system.

Automated testing: This involve using tools such as Jest to automate the execution of unit and integration tests.

Performance testing: This involve using a tool such as Artillery to simulate different levels of load on the system and measure the response time and resource utilization.

With my learning of the course, we can also use:

Functional testing: testing the system's functionality by providing inputs and comparing the outputs to the functional requirements. It ensures that the system meets the specifications outlined in the requirements document.

Structural testing: testing the internal structure of the code and ensuring that all parts of the code are executed during testing. The goal is to achieve high code coverage by testing all possible code paths.

Model-based testing: testing that generates test cases based on a model of the system's behaviour. It can be used to test the system's behaviour in various states and transitions, as well as its performance in edge cases and exceptional states. It is frequently used to test complex systems, such as state machines and flow graphs.

1.4

Assess the appropriateness of your chosen testing approach

Manual testing:

Pros:

Flexibility: Manual testing allows for a high level of flexibility, as testers can adjust the test cases and scenarios on the fly and test the system in an ad-hoc manner.

Usability: Manual testing can be useful for evaluating the usability of the system, as it allows testers to use the system in the same way that end users would.

Functionality: Manual testing can be effective for testing the functionality of the system, as it allows testers to interact with the system and test different scenarios.

Cons:

Time and cost: Manual testing can be time-consuming and costly, as it requires manual effort and may not be as efficient as automated testing.

Reliability: Manual testing can be less reliable than automated testing, as it is subject to human error and variability.

Coverage: Manual testing may not provide as comprehensive coverage of the system as automated testing, as it is limited by the time and resources available for testing.

Automated testing:

Pros:

Efficiency: Automated testing can be more efficient than manual testing, as it allows test cases to be executed quickly and without the need for manual effort.

Reliability: Automated testing can be more reliable than manual testing, as it is not subject to human error and variability.

Coverage: Automated testing can provide comprehensive coverage of the system, as it allows a large number of test cases to be executed.

Repeatability: Automated testing allows test cases to be easily repeated, which can be useful for verifying the stability of the system.

Cons:

Cost: Automated testing can be costly, as it requires the development and maintenance of automation scripts and tools.

Complexity: Automated testing can be complex, as it requires the development of test cases and the integration of automation tools into the testing process.

Limited scope: Automated testing may not be suitable for testing certain scenarios or aspects of the system that are difficult to automate.

Performance testing:

Pros:

Identifying performance issues: Performance testing can help to identify performance bottlenecks.

Evaluating scalability: Performance testing can help to evaluate the scalability of the system and determine its ability to handle increasing workloads.

Improving performance: Performance testing can help to identify opportunities for improving the performance of the system.

Predicting system behaviour: Performance testing can help to predict the behaviour of the system under different load conditions and identify potential issues before they occur.

Cons:

Cost: Performance testing can be costly, as it requires the development and execution of specialized test cases and the use of specialized tools

Complexity: Performance testing can be complex, as it requires the development of test cases and the setup of load-testing environments.

Limited scope: It may be that lack of data makes it difficult to do performance testing under realistic loads.

Functional testing:

Pros:

Helps to ensure that the system meets the specified requirements and functions as intended.

Can be used to test the system's behaviour in various scenarios and edge cases.

Cons:

May not be able to detect errors in the code but doesn't cover functions in the requirements.

Can be time-consuming and resource-intensive if many test cases need to be executed.

Structural testing:

Pros:

Can achieve high code coverage.

Relatively easy to design and implement.

Can be used to detect defects in the internal structure of the code.

Cons:

May not be able to detect certain types of defects, such as usability or performance issues

Model-based testing:

Pros:

Can be used to test the system's behaviour in various states and transitions, as well as its performance in edge cases and exceptional states.

Cons:

Can be challenging to create accurate and comprehensive models of the system's behaviour.

Overall, these test approaches can be used to ensure that the API meets the functional, performance, and security requirements, and that it is robust and can handle different scenarios and edge cases. By combining manual, automated, and performance testing, we can ensure a thorough testing of the system and increase the confidence in its ability to handle real-world scenarios before it is released to production. Additionally, utilizing functional, structural and model-based testing will ensure that the system's behaviour and its internal structure are thoroughly tested, and that all possible code paths are executed during testing.