LO1 Supporting Document

1.1 Requirements specifications

Here are the requirements for our food box project. Note that many of these requirements are additional features that are not implemented, but are requirements that would eventually have to be met for a successful product.

Functional requirements:

- The software should allow the user to navigate through an UI to perform actions
- User registration: the system should be able to create new user accounts and store the provided personal information and password in the database.
- Authentication: the system should be able to verify the email and password provided by the user during login and grant or deny access accordingly.
- Order management:
    - The system should allow the user to place an order for a predefined food box and store it in the database
    - The system should allow the user to view all their orders
    - The system should allow the user to get a single order info
    - The system should allow the user to update an existing order
- User access levels: the system should be able to grant different levels of access to the application based on the user's role (admin or user).
- Payment processing: the system should be able to process payments for orders placed by users.
- Order tracking: the system should allow users to track the status of their orders.
- Inventory management: the system should allow administrators to track and manage the inventory of food boxes.
- Notifications: the system should be able to send notifications to users regarding their orders, such as confirmation of receipt and delivery status.

- Reporting: the system should provide administrators with detailed reports on system usage, orders, and inventory.
- User management: the system should allow administrators to manage user accounts and access levels.
- Error handling: the system should be able to handle errors and provide appropriate responses to users.
- Sales metrics: the ability to access and manage data related to the system, such as customer orders and sales metrics

Measurable quality attributes:

- Performance:
  - The system should be able to handle a high number of requests and process them in a timely manner.
  - Interactions with the server which require processing, such as login, should take less than 3 seconds
  - Over reasonably common internet connection speeds, the server should respond to client requests in less than one second
  - Querying the database should take less than one second
- Scalability: the system should be able to handle an increase in the number of users and orders without negatively impacting performance.
- Availability:
  - The system should be available to users at all times.
  - The system should be available to users in any location
- Recovery:
  - The system should have a data backup system in place to ensure that user data and order information is not lost in the event of a system failure.
  - The system should backup data every hour
  - The system should have a short recovery time (less than 1 minute)
- Security and privacy:

    - Users must be logged in in order to view their orders and place orders
    - The system should protect user's personal information and password using encryption.
    - Communication between the API and the database must be encrypted

Qualitative requirements:

- User-friendly interface: the system should be easy to use for users of all skill levels.
- Accessibility: the system should be accessible to users with disabilities.
- Customization: the system should allow users to personalize their experience within the limits of their access level.
- Maintainability: The code should be built modularly and common coding styles should be utilised.
- Help and support: the system should provide users with clear and accessible help and support resources.
- Privacy policy: the system should have a clearly defined privacy policy that outlines how user data is collected, stored, and used, having compliance with GDPR regulation.
- Multi-language support: the system should be able to support multiple languages for users.
- Mobile compatibility: the system should be optimized for mobile devices.
- Compliance: the system should comply with relevant regulations and standards.

1.2 Level of requirements, system, integration, unit

For this section I will only focus on a few requirement examples to keep the document short. Note that the strategies I will outline here can be applied for other similar requirements.

- ***System level requirements*** (high-level requirements that define the overall functionality and goals of the system). This includes:

- User registration: the system should be able to create new user accounts and store the provided personal information and password in the database.
    - ***System tests:***
        - Test that new user accounts can be created by submitting a registration form with valid information and that the system correctly stores the provided personal information and password in the database.1.
        - Test that the system prevents users from registering with an already existing email address, and that it sends an appropriate error message.
        - Test that the system sends an email to the user with a verification link, and that the user can activate the account by clicking on the link.
- Performance: The system should be able to handle a high number of requests and process them in a timely manner.
    - ***System tests:***
        - Test that the system can handle a large number of concurrent user requests, such as 100 users making requests at the same time, and that the response time for each request is within the acceptable range (less than 3 seconds).
        - Test the system's ability to handle a high number of requests over a prolonged period of time, such as 10,000 requests over the course of an hour, and ensure that response times remain within the acceptable range.

- **Unit level requirements** (requirements that specify the individual components or sub-systems of the system.). Below are some examples with some unit tests:
    - User registration: test cases should be created to test the submission of the registration form, the validation of user input, and the storage of user information in the database.
    - Order placement: Test cases should be created to test the submission of the order form, and the storage of order information in the database, to test that the placing an order of a box that doesn’t exist returns an error.
- **Integration level requirements** (requirements that specify how different sub-systems of the system should work together). Below are some integration tests

examples:

- User registration and authentication: Test cases should be created to test the interaction between the registration and authentication systems, such as the creation of a new user account adding the user to the database, and the subsequent login.
- Payment processing and order tracking: Test cases should be created to test the interaction between the payment processing and order tracking systems, such as the confirmation of payment and the updating of order status.

1.3 Identifying test approach for chosen attributes

There are various testing approaches that we could use to test our requirements. Note that we could use a combination of different testing approaches for 1 requirement. We would perform white box and black box testing. White box testing would focus on the internal logic of the system and the implementation of the code, such as unit testing. On the other hand, black box testing is a method where the tester only has access to the external interface of the system, which could include functional testing or acceptance testing.

Here is a more precise breakdown of testing approaches we could use:

- **Unit testing**: This type of testing is focused on testing individual units or components of the system, such as a single function or class. An example of a unit test for the "User registration" requirement might be to test that the function responsible for creating new user accounts correctly stores the provided personal information and password in the database.
- **Integration testing**: This type of testing is focused on testing how different units or components of the system work together. An example of an integration test for the "Order management" requirement might be to test that the functions responsible for placing, viewing, updating, and getting information about an order all work correctly together and with the database.
- **System testing**: This type of testing is focused on testing the entire system as a whole, including all its functional and non-functional requirements. An example of a

system test for the "Performance" requirement might be to test that the system can handle a high number of concurrent requests and process them in a timely manner by simulating a large number of users making requests simultaneously.

- **Acceptance testing**: This type of testing is focused on testing whether the system meets the acceptance criteria defined by the stakeholders. An example of an acceptance test for the "Security and privacy" requirement might be to test that the system properly encrypts user's personal information and password and that communication between the API and the database is also encrypted.
- **Usability testing**: This type of testing is focused on testing the user interface of the system and how easy it is for users to complete tasks. An example of a usability test for the "User-friendly interface" requirement might be to test that users of all skill levels are able to complete common tasks, such as placing an order or updating their personal information, without difficulty.
- **Security testing**: This type of testing is focused on testing the security of the system, such as testing for vulnerabilities or compliance with security standards. An example of a security test for the "Recovery" requirement might be to test that the system's data backup system is working correctly and can recover from a failure within the specified time frame.
- **Performance testing**: This type of testing is focused on testing the performance of the system, such as response time and resource usage. An example of a performance test for the "Scalability" requirement might be to test that the system can handle an increase in the number of users and orders without negatively impacting performance.

1.4 Assess the appropriateness of your chosen testing approach

Here are some limitations that can arise when using our testing approaches:

- ***Functional testing:***
    - Complexity: Testing a complex system can be difficult and time-consuming, especially when trying to cover all possible scenarios.
    - Limited ability to find defects: Functional testing is mainly used to check that the system behaves as expected, but it may not be able to find defects that are not

related to the system's functionality.

  - Maintenance: Functional testing requires maintenance as the requirements, system and the testing scripts changes over time.

- ***Performance testing:***
  - It may not be able to test all the different scenarios that a real-world system might encounter, such as sudden spikes in traffic or unexpected usage patterns.
  - It may not be able to test the system for a long enough period of time to identify any performance issues that might only occur after extended usage.
  - It can be resource-intensive and may be difficult to execute under realistic loads without access to a large amount of data or a realistic test environment.
- ***Usability testing:***
  - It is often done manually and can be time-consuming, and it can be difficult to get a representative sample of users to test the system.
  - Results of usability testing are often subjective, and it can be hard to get a consistent evaluation from different testers.