**Exam Project Softwareudvikling 2. Semester**

**Test Strategy**

**Product**

Overview: A food delivery application MTOGO that employs delivery drivers to pick up orders from restaurants that have signed up to the application. The application should allow all users to:

Create account: Users should be able to create an account and choose their role (some roles are limited).

Login: Users can login to search for restaurants and/or accept orders.

Customers should be able to:

Search for restaurants: Customers can search for restaurants based on names, locations or categories.

View menus: Customers can view menus from restaurants.

Place orders: Customers can place orders.

Give feedback: Customers can give feedback once an order has been delivered.

Restaurants should be able to:

Accept orders: Restaurants can accept or decline incoming orders from customers.

Delivery drivers should be able to:

Accept orders: Delivery drivers can accept or decline orders that are ready to be delivered.

Admins should be able to:

View orders: Admins can see all the orders and their status.

View delivery drivers: Admins can see all delivery drivers and their status.

**Objective**

Ensure that the application meets all functional requirements.

Ensure the application is stable and reliable.

Confirm that the application performs well with typical user interactions.

**Scope**

Functional Testing: Test key features, including user registration, order placement, menu viewing, delivery and feedback.

Non-Functional Testing: Performance testing for scalability and security testing for data protection.

**Test Methodology**

This project is made following Agile principles and tests will be made while developing to test the code that is created. This means errors and bugs will be found early in development and will be easier to solve. Test Driven Development will be used for unit testing and ensuring code reliability early in the process.

It will also be a priority to setup an automated tests using GitHub Actions so that the tests will be run automatically when code is pushed to GitHub. The application will also be split into different services and each service will have tests that are isolated and only test that specific service. There will also be tests for the integration of the different services.

**Testing Approach**

Unit Testing:

Test individual components and functions using TDD to ensure the code is being tested early on.

Integration Testing:

Test the interaction between multiple components and functions.

System Testing:

Testing of the full application like the full process of order placement to delivery.

Acceptance Testing:

Tests to ensure that the system satisfies the acceptance criteria.

Load Testing:

Ensure that the system can handle an increase in load to ensure future growth.

Mutation Testing:

Changing the application to ensure that the tests can detect the change.

**Entry Criteria**

Development of feature or module is complete.

Unit tests for individual components have passed.

Test environment is set up and configured.

Test data is prepared for use across environments.

Requirements and user stories are clearly documented and understood

**Exit Criteria**

All unit tests, integration tests, system tests and acceptance tests passed successfully.

The application fulfills all functional requirements and user stories.

The load tests aren’t slower than 2 seconds.

No critical bugs or issues.

**Test Automation Strategy**

Automate unit tests and integration tests.

Having automated tests ensures that the code passes all tests.

**Risk Assessment**

Low test coverage can lead to problems with issues in the code going undetected.

Requirements changing can be an issue if I find that I want to change something in the application.

**Code coverage**

At least 65% code coverage on critical modules (order processing, bonus calculation, feedback collection).

Prioritize core business functionalities to maintain accuracy and reliability.