TEST PLAN FOR

<<CRAVE>>

Note that you can refine your testing plan as the project development goes. Keep the change log as follow:

ChangeLog

Version	Change Date	By	Description
1.0.0	Oct. 18, 2023	All Members	Initial Testing Plan for sprint 2

1 Introduction

1.1 Scope

Scope defines the features, functional, or non-functional requirements of the software that **will be** tested.

This is our testing scope for **Sprint 2**:

1. Account
    a. User registration
2. Dashboards
    a. Restaurants dashboard
    b. Restaurants menus

c. Shopping cart

1.2 Roles and Responsibilities

Detailed description of the Roles and responsibilities of different team members like. Note you only need to list the role you have in your team. There are some example roles.

- QA Analyst
- Test Manager
- Configuration Manager
- Developers
- Installation Team

TODO: Update here (assign other roles? Check net Ids)

Name	Net ID	GitHub username	Role
Jared Castillo	castil13	jablue-12	Full Stack Developer
Koye Fatoki	fatokioo	koy-e	Backend Developer
Eddie Wat	wate	ediwat	Frontend Developer
Xiaoran Meng	mengx5	xiaoran-MENG	Frontend Developer

Role Details:

1. Frontend Developer
    - A developer whose primarily focus is on creating components that make up the UI/UX of the website.
2. Backend Developer
    - A developer whose focus is on the server side of web development, working with databases, server logic and APIs.
3. Full Stack Developer
    - A developer who is both a frontend and a backend developer

2 Test Methodology

2.1 Test Levels

Unit Tests:

Core Feature: Account

1. Ensure that the Account object is created properly.
2. Ensure that the Login object is created properly.
3. Ensure that the Login Service can retrieve the appropriate account ID
4. Ensure that the account service can get the available accounts in the database.
5. Ensure that the account service can create new accounts in the database.
6. Ensure that the account service can update account info in the database.

7. Ensure that the account service can find an account when given an id.
8. Ensure that the account service can delete an account from the database.
9. Ensure that the account repository can do a GET request for getting all the accounts.
10. Ensure that the account repository can do a POST request for adding a new account.
11. Ensure that the account repository can do a POST request for updating an existing account.
12. Ensure that the account repository can do a GET request for getting a specific account by specifying an ID.

Core Feature: Dashboard

1. Ensure that Restaurant and RestaurantItem objects are created properly.
2. Ensure that the restaurant service is able to get the available restaurants in the database.
3. Ensure that the restaurant service is able to create new restaurants in the database.
4. Ensure that the restaurant service is able to update restaurant info in the database.
5. Ensure that the restaurant service is able to find a restaurant when given an id.
6. Ensure that the restaurant service is able to delete a restaurant from the database.
7. Ensure that the Restaurant repository is able to do a GET request for getting all the restaurants.
8. Ensure that the Restaurant repository is able to do a POST request for adding a new restaurant.
9. Ensure that the Restaurant repository is able to do a POST request for updating an existing restaurant.
10. Ensure that the Restaurant repository is able to do a GET request for getting a specific restaurant by specifying an ID.

Core feature: Comments and ratings

1. Ensure that the comment object is created properly.
2. Ensure that the comment service can get the available comments in the database.
3. Ensure that the comment service can create new comments in the database.
4. Ensure that the comment service can update comment info in the database.
5. Ensure that the comment service can find a comment when given an id.
6. Ensure that the comment service can delete a comment from the database.
7. Ensure that the comment repository can do a GET request for getting all the comments.
8. Ensure that the comment repository can do a POST request for adding a new comment.
9. Ensure that the comment repository can do a POST request for updating an existing comment.
10. Ensure that the comment repository can do a GET request for getting a specific comment by specifying an ID.

Core feature: Ordering

1. Ensure that the order object is created properly.
2. Ensure that the order service can get the available orders in the database.
3. Ensure that the order service can create new orders in the database.
4. Ensure that the order service can update order info in the database.
5. Ensure that the order service can find an order when given an id.
6. Ensure that the order service can delete an order from the database.
7. Ensure that the order repository can do a GET request for getting all the orders.
8. Ensure that the order repository can do a POST request for adding a new order.
9. Ensure that the order repository can do a POST request for updating an existing order.

10. Ensure that the order repository can do a GET request for getting a specific order by specifying an ID.

Core feature: Real time statistics

1. Ensure that the restaurant stats object is created properly.
2. Ensure that the restaurant stats service can get the available restaurant stats in the database.
3. Ensure that the restaurant stats service can create new restaurant stats in the database.
4. Ensure that the restaurant stats service can update restaurant stats info in the database.
5. Ensure that the restaurant stats service can find restaurant stats when given an id.
6. Ensure that the restaurant stats service can delete any restaurant stats from the database.
7. Ensure that the restaurant stats repository can do a GET request for getting all the restaurant stats.
8. Ensure that the restaurant stats repository can do a POST request for adding a new restaurant stats.
9. Ensure that the restaurant stats repository can do a POST request for updating an existing restaurant stats.
10. Ensure that the restaurant stats repository can do a GET request for getting a specific restaurant stats by specifying an ID.
11. Ensure that users past orders are analyzed correctly for recommendation

Integration Tests:

Core Feature: Account

1. Registration
2. Login

Core Feature: Dashboard

1. View restaurants
2. View restaurant menu
3. View dish
4. View recommended combos

Core Feature: Shopping Cart

1. Add to cart

Core Feature: Real-time Analytics

1. Markers for nearby restaurants on the embedded Google Map
2. Display of the number of orders placed along with basic details of the restaurant moused over on the embedded Google Map
3. Radar chart on user dietary preferences

2.2 Test Completeness

Testing will be complete when:

- 100% test coverage
- All automated test cases are executed successfully in both frontend and backend
- Bug issues are fixed or will be fixed for the next release

3 Resource & Environment Needs

3.1 Testing Tools

General Tools:

- Github Actions (CI pipeline)
- Github Issues for tracking dev-tasks and bugs
- Intellij IDE for backend development
- VS code for frontend development
- Testing Library (jest and junit)

Frontend:

- JavaScript
- React 18
- React Bootstrap
- Jest

Backend:

- Java 21
- Spring boot 3
- PostgresSQL
- JUnit 5

3.2 Test Environment

The minimum **hardware** requirements that will be used to test the Application are:

- A machine with Windows 10-11, and Mac operating systems

The following **softwares** are required in addition to client-specific software:

- Node JS v18.x.x
- Java 17, 21
- Maven 3.x.x

4 Terms/Acronyms

Make a mention of any terms or acronyms used in the project

TERM/ACRONYM	DEFINITION
API	Application Program Interface
AUT	Application Under Test
UI	User Interface
UX	User Experience