Testing Manual for Farmers Market

Introduction

This document provides detailed instructions for testing the Farmers Market Search Program. The goal is to ensure that all features work as expected and the application is free of defects.

Testing was used extensively throughout development to ensure the code was functioning as expected. JUnit 5 was used in the backend code to test functionality. The GUI was manually tested.

Test Strategy for `Farmers Market Search Program`

Objectives

The main objectives of the testing strategy are to:

- Ensure the program application fulfills all specified functional requirements.
- Check the application's behavior under a range of scenarios and edge cases.
- Assess the user interface for both usability and accessibility.
- Verify the application's performance efficiency across varying load conditions.

Test Environment

- **Operating System**: Windows 10, macOS, or Linux
- **Java Version**: JDK 11 or later
- **JavaFX Version**: 11 or later

Setup Instructions

1. **Install Java and JavaFX**:
    - Ensure you have a Java Development Kit (JDK) installed. You can download it from here.
    - Install JavaFX SDK from here.
2. **Compile the Application**:
    - Make sure you are in the correct directory:

      Something like so:

      src/edu/rpi/cs/csci4963/u24/kims35/hw05/farmersmarket

```
javac *.java
```

Run the Application:

```
java --module-path /path/to/javafx-sdk/lib --add-modules
javafx.controls,javafx.fxml
edu.rpi.cs.csci4963.u24.kims35.hw05.farmersmarket.FarkerApplication.java
```

Note: edit the `/path/to/javafx-sdk/lib` with your path to the JavaFx sdk

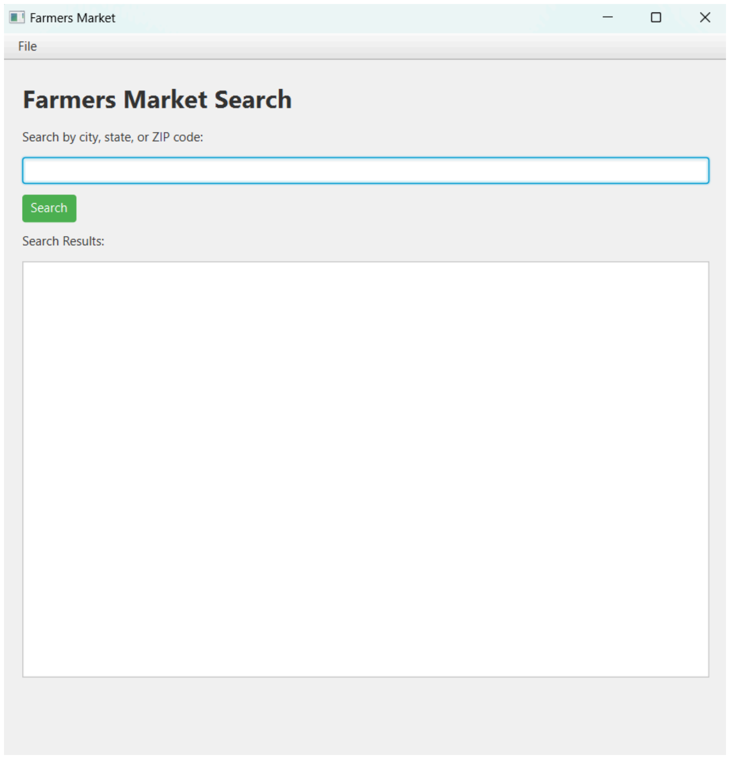
Test Cases:

Functional Tests:

Application Launch:

- **Test Case**: Launch the FarketApplication.
- **Steps**:
    1. Open a terminal or command prompt.
    2. Navigate to the directory containing the compiled classes.
    3. Run the application using the command provided in the setup instructions.
- **Expected Result**: The application window should open without errors.

  Main window should look like so:



Script Testing:

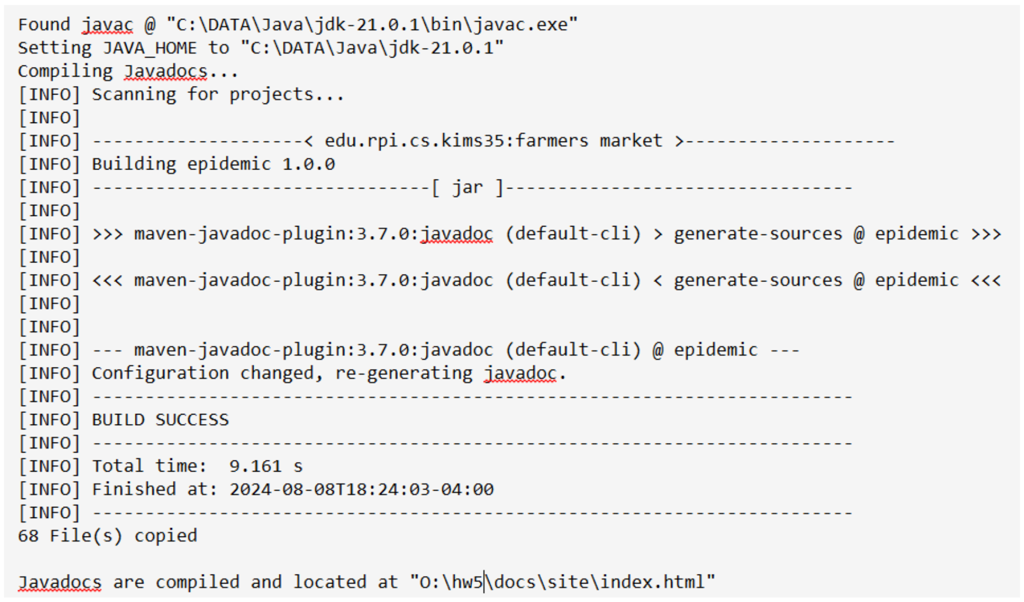
Javadoc Script:

Test Case: javadoc script correctness

Input:

scripts\run.bat javadoc

Expected Result:



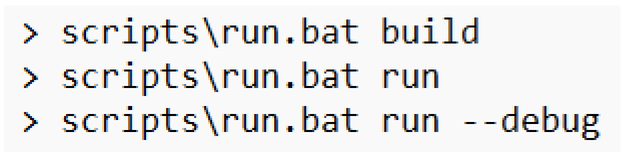
```
Found javac @ "C:\DATA\Java\jdk-21.0.1\bin\javac.exe"
Setting JAVA_HOME to "C:\DATA\Java\jdk-21.0.1"
Compiling Javadocs...
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< edu.rpi.cs.kims35:farmers market >--------------------
[INFO] Building epidemic 1.0.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> maven-javadoc-plugin:3.7.0:javadoc (default-cli) > generate-sources @ epidemic >>>
[INFO]
[INFO] <<< maven-javadoc-plugin:3.7.0:javadoc (default-cli) < generate-sources @ epidemic <<<
[INFO]
[INFO]
[INFO] --- maven-javadoc-plugin:3.7.0:javadoc (default-cli) @ epidemic ---
[INFO] Configuration changed, re-generating javadoc.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.161 s
[INFO] Finished at: 2024-08-08T18:24:03-04:00
[INFO] ------------------------------------------------------------------------
68 File(s) copied

Javadocs are compiled and located at "O:\hw5\docs\site\index.html"
```

Build and Run Script:

Test Case: script correctness

Input:



```
> scripts\run.bat build
> scripts\run.bat run
> scripts\run.bat run --debug
```

Expected Results: Program builds, compiles, and runs properly.

Error Cases:

Missing Config File:

Test Case: config.properites file does not exist

Expected Result: System.out.println("Sorry, unable to find config.properties");

Config Dialog File:

Test Case: ConfigDialog.fxml file missing

Expected Result:

```
catch (IOException e) {

   e.printStackTrace();

}
```

Query Part Parsing Error:

Test Case: Query to be parsed as doubles cannot be parsed

Expected Result:

```
catch (NumberFormatException e) {

   e.printStackTrace();

}
```

Database Connection Error:

Test Case: Driver Manager cannot establish a connection with the provided url, user, and/or password

Expected Result:

```
catch (IOException | SQLException e) {

   e.printStackTrace();

}
```

Review Submission Error:

Test Case: The name or rating is missing when submitting a review

Expected Result:

```
System.out.println("Name and rating are required.");
```

Improper Rating Input (number):

Test Case: A number outside of 1-5 (included) has been inputted

Expected Result:

```
throw new NumberFormatException();
```

Improper Rating Input (non-numerical):

Test Case: A non-numerical has been inputted

Expected Result:

```
catch (NumberFormatException e) {

   System.out.println("Rating must be a number between 1 and 5.");

   return;

}
```

CSV File Access Error:

Test Case: csv file cannot be found

Expected Result:

```
catch (IOException | CsvValidationException e) {

   e.printStackTrace();

}
```

Database Connection Error (market search table creation):

Test Case: Driver Manager cannot establish a connection with the provided url, user, and/or password

Expected Result:

```
catch (SQLException e) {
   e.printStackTrace();

}
```

Database Connection Error (review table creation):

Test Case: Driver Manager cannot establish a connection with the provided url, user, and/or password

Expected Result:

```
catch (SQLException e) {

   e.printStackTrace();

}
```

Database Connection Error (table search connection):

Test Case: Driver Manager cannot establish a connection with the provided url, user, and/or password

Expected Result:

```
catch (SQLException e) {

   e.printStackTrace();

}
```

Database Connection Error (hasUserReviewed review table check):

Test Case: Driver Manager cannot establish a connection with the provided url, user, and/or password

Expected Result:

```
catch (SQLException e) {

   e.printStackTrace();

}
```

Database Connection Error (addReview table connection):

Test Case: Driver Manager cannot establish a connection with the provided url, user, and/or password

Expected Result:

```
catch (SQLException e) {

   e.printStackTrace();

}
```

Testing Suite Description:

To test the Farmers Market Search Program, I covered key areas:

Boundary Value Tests: I checked how the program handles extreme cases, like empty inputs or max-sized data, ensuring it reacts correctly or shows helpful error messages.

Concurrency Tests: I tested the program's ability to handle multiple actions at once, focusing on database interactions to prevent data issues.

Usability Tests: I verified that the user interface is easy to navigate and that all buttons and menus work as expected.

Performance Tests: I assessed how well the program runs with large datasets and multiple users to ensure it stays fast and stable.

Accuracy Tests: I confirmed that the search and review features work correctly, accurately matching queries and recording ratings.

Error Handling Tests: I simulated common errors, like missing files or bad inputs, to make sure the program handles them gracefully.

Integration Tests: I made sure that all parts of the program work together smoothly without any glitches.