Intramurals Test Plan

Prepared by: Bach Tran

Date: 11/7/2022

# Objectives

**Primary Objective:** Create an automated test suite derived from analysis of the software requirements specification (SRS) document and the application (work in progress) that should be deployed by the team. The test suite will help to drive future development of the application by allowing for regression testing to be efficiently performed whenever features are added or modified. Requirements that have not yet been implemented/completed should be noted. The results of end-to-end test execution should be summarized in a **test closure report document.**

**Secondary Objective:** Develop unit and integration tests for the application’s backend. Unit tests should be focused on testing individual units of logic in the service layer of the application to achieve high test coverage (>80%) in the service layer. Integration tests should be focused on testing the API endpoints (integrated with the service layer, DAO layer + database) to achieve more comprehensive testing of the backend’s functionality. The number of integration tests should be sufficient to cover most API endpoints.

# Tasks

The team should analyze the software requirements and perform exploratory work on the application to fully understand the features of the application. Test cases/scenarios should be created in Gherkin. These test cases should include positive and negative scenarios. Test cases should then be automated through the aid of technologies such as Junit 4, Cucumber, and Selenium to the maximum extent possible. Any requirements that have not yet been fulfilled/completed should be noted. A test closure report document should be produced after the conclusion of end-to-end test execution.

The backend application’s code should be analyzed to determine unit tests that should be written to contribute to coverage of all important logic paths in the service layer (positive + negative). API endpoints should also be tested by writing a variety of positive + negative integration tests that will test the coupling of the controller layer, service layer, and DAO layer + database.

# Scope

## General

The test suite to be developed should consist of end-to-end test cases that are thoroughly documented as well as automated wherever feasible.

The backend application should consist of unit and integration tests that contribute to thorough testing of all backend logic for both positive + negative paths.

## Tactics

Every team member should be focused on both the creation of end-to-end test cases on a manual “by-hand” basis as well as in the automation of such test cases. This ensures that all team members are fully capable and involved in the entire testing process. Team members should then divide work into different features that can be tackled either independently or through the utilization of a “paired-programming” approach. Any team member that is finished with their tasks should assist other members to complete their tasks.

Unit testing + integration testing will not require manual “by-hand” creation, as the unit tests + integration tests should be clear enough as to be “self-documenting”. Each team member should be involved with both unit + integration testing. Delegation to each team member of which methods (unit) and API endpoints (integration) to be tackled should be determined by the team

**Team members should document what work they accomplished for the project AND what assistance they provided to other team members, if applicable, with as much specifics as possible.**

# Testing Strategy

The team will utilize Agile methodologies and emphasize daily communication and collaboration. There should be daily standups to communicate progress and identify blockers/problems that arise during the project. Progress should be tracked using a Kanban board on a platform such as Jira or Trello. Test cases should be created from the requirements and exploratory testing of the application.

## End-to-end Testing

### Definition

A form of blackbox testing that does not require knowledge of the internal workings of the application. It is a form of testing in which interactions with the application have the same perspective as that of a user/customer. End to end testing of this particular application involves interacting directly with a webpage using a web browser such as Google Chrome.

### Tools

Creation of test scenarios + test cases can be performed “by-hand” using tracking tools such as Jira/Trello. Automation of end-to-end tests will involve the utilization of JUnit 4, Cucumber, and Selenium.

### Project/Repository Setup

Automated end-to-end tests should be their own separate project in their own separate repository from the actual application being tested

## Unit Testing

### Definition

A form of whitebox testing that requires knowledge of the internal workings of the application. It is a form of testing that is 100% automated, where test cases directly invoke and interact with methods in the application code. Unit testing contributes to code coverage, whereby various tools can be utilized to provide code coverage reports. Methods should be tested in isolation, which will require the use of mocking.

### Tools

Either JUnit 4 or 5 can be utilized for unit testing alongside Mockito for mocking

### Project/Repository Setup

Unit tests should be part of the backend application project / repository since there is direct interaction with the backend application code

## Integration Testing

### Definition

A form of either black box OR white box testing depending on the context. In the case of API testing with Postman, integration testing (API testing) would be considered black box. The scope of this project is that integration tests should be written to interact directly with the application code, and therefore are WHITEBOX tests. Integration tests should be testing the API endpoints and ultimately test the integration between the controller, service, and DAO layers + database in order to validate that the components are working properly when integrated together. Hence, no mocking is utilized for integration testing unlike unit testing. Whitebox integration tests contribute to code coverage.

### Tools

JUnit 4 or 5 can be utilized for integration testing alongside an H2 in-memory database in order to avoid modifying the external Postgres database

### Project/Repository Setup

Integration tests should be part of the backend application project / repository since there is direct interaction with the backend application code

# Test Schedule

The testing of the application will occur over a duration of 2 - 3 weeks. It is up to the team to plan, delegate, and complete tasks in accordance with the deadline. Daily standups should be utilized in order to prioritize what needs to be completed daily and whether the project is proceeding ahead of schedule or behind schedule. Work should be kept track of using a Kanban board as previously mentioned to provide real-time feedback on progress. Any delays on project progress should be acknowledged by the entire team and focus/assistance directed toward that task.

# Project Repositories

## End-to-end Testing

End-to-end testing should be part of a separate Java project repository that the team should set up. Ensure that all team members have proper permissions to write and read from the repository.

## Unit and Integration Testing

Unit and integration testing should be written within the backend application’s project repository that the team should set up. Ensure that all team members have proper permissions to write and read from the repository. The baseline repository that can be forked can be found here (<https://github.com/adamranieri/IntraMuralsApp>).

# Software + Hardware Requirements

## Hardware

A computer is required with a 64-bit processor and ideally 8+ GB of RAM

## Software

* Oracle JDK 8 (<https://www.oracle.com/java/technologies/downloads/#java8>)
* IntelliJ Community Edition (<https://www.jetbrains.com/idea/download>)
* Maven (<https://maven.apache.org/download.cgi>)
* Google Chrome (<https://www.google.com/chrome/dr/download>)

# Features To Be Tested

Please refer to the software requirements specification document provided for the most up-to-date information regarding requirements

# 3.1.1. Unregistered player functionalities:

3.1.1.1. Register for an account

3.1.1.2. View the Captains of each team

3.1.1.3. View all the games

3.1.1.4. View all the seasons

3.1.1.5. View all the venues

3.1.1.6. View all the schedules

# 3.1.2. Player functionalities:

3.1.2.1. Login to the system

3.1.2.2. Update their own password

3.1.2.3. Update their own profile picture

3.1.2.4. Update their own height and weight

3.1.2.5. Choose if they want to display their biometric info or not

3.1.2.6. View the Captains of each team

3.1.2.3. View all the games

3.1.2.4. View all the series

3.1.2.5. View all the venues

3.1.2.6. View all the schedules

3.1.2.7. Apply to join a certain team

3.1.2.8. Apply to become an inactive player

3.1.2.9. Apply to become an active player

# 3.1.3. Referee functionalities:

3.1.3.1. Login to the system

3.1.3.2. Update their own password

3.1.3.3. Update their own profile picture

3.1.3.4. Update their own height and weight

3.1.3.5. Choose if they want to display their biometric info or not

3.1.3.6. View the Captains of each team

3.1.3.7. View all the games

3.1.3.8. View all the series

3.1.3.9. View all the venues

3.1.3.10. View all the schedules

3.1.3.11 Edit the scorecard for any game they officiate

# 3.1.4. Admin functionalities:

3.1.4.1. Login to the system

3.1.4.2. Update their own password

3.1.4.3. Update their own profile picture

3.1.4.4. Update their own height and weight

3.1.4.5. Choose if they want to display their biometric info or not

3.1.4.6. View the Captains of each team

3.1.4.7. View all the games

3.1.4.8. View all the seasons

3.1.4.9. View all the venues

3.1.4.10. View all the schedules

3.1.5.11. Create a season

3.1.5.12. Schedule all games in a season

3.1.5.13. Create new admins

3.1.5.14. Create new referees

3.1.5.15 Full CRUD abilities of games

# 3.1.5. Captain functionalities:

3.1.5.1. Login to the system

3.1.5.2. Update their own password

3.1.5.3. Update their own profile picture

3.1.5.4. Update their own height and weight

3.1.5.5. Choose if they want to display their biometric info or not

3.1.5.6. View the Captains of each team

3.1.5.7. View all the games

3.1.5.8. View all the series

3.1.5.9. View all the venues

3.1.5.10. View all the schedules

3.1.6.11. Approve or Deny requests from players to join their team

# 3.1.6. The RESTful API should have the following functionalities:

3.1.6.1. Register a team

3.1.6.2. Retrieve all teams

3.1.6.3. Retrieve all venues

3.1.6.4. Schedule a game

3.1.7.5. Retrieve all games

3.1.7.6. Retrieve all seasons

# 3.1.7 Additional Business Requirements

3.1.7.1 Games cannot be scheduled for past dates