**CS673 Software Engineering** 

**Team 2 - CareerForge**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Stacey Burns | QA Leader | *SB* | 09/22/2025 |
| Qi Chen | Requirement Leader | *QC* | 09/22/2025 |
| Gopi Rayini | Configuration Leader | *GR* | 09/22/25 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1.0.0** | **Team** | **09/22/2025** | **Initial Draft** |
|  |  |  |  |

[Testing Summary](#_sm5odwyvuk3j)

[Manuel Tests Reports](#_pqso2mbjyzx4)

[Automated Testing Reports](#_mtfbusfb0eq3)

[Testing Metrics](#_rijyjeu2ojqa)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Testing Summary

In this section, you will summarize what was tested, who is involved in testing, when to test, testing techniques used, and testing result. You may have the following tests

* + Unit Testing - Our unit tests are automatically executed as part of every Maven build and deployment, ensuring continuous validation of the system. The tests cover all core entities and security components, including:
    - Entity Layer: Validation of User, Job, ApplicationTracking, Benefits, Values, and LocationCoordinate entities, focusing on field integrity, default/null handling, relationships, lifecycle methods (timestamps, updates), and equality/hashCode correctness.
    - Security Layer: Validation of JwtUtil for token generation, validation, and expiration handling, as well as integration tests for secured endpoints (/secure, /admin/dashboard) to confirm proper authentication and role-based authorization.
    - Roles Involved: Developers are responsible for writing, reviewing, and executing the unit tests as part of the CI/CD pipeline.

Our unit tests are executed automatically during each Maven build and before every deployment. Developers also run these tests locally before pushing changes.

The techniques used are JUnit5 for test execution and assertions, SpringBootTest with MockMvc for security and endpoint validation, Dynamic Property Injection to generate secrets and configuration values at runtime, ensuring isolation and preventing flaky tests, and test Doubles/Mocks where applicable to isolate functionality.

* + Specific unit tests:
    - Unit tests for ApplicationTracking - Verifies that the entity behaves correctly in terms of construction, field values, and business logic between the user, job, and application tracking.
    - Unit tests for Benefits - Verifies that the entity correctly handles creation, field updates, nulls/empties, equality, hashing, and string representation.
    - Unit tests for Jobs- Verifies Job entity. It checks that job postings are created, validated, and updated correctly, while also verifying related behaviors like deadlines, salaries, applications, and relationships.
    - Unit tests for LocationCoordinates - Verifies that latitude and longitude values are handled correctly, along with equality, hashing, and string output.
    - Unit tests for Users - Validates that a user can be created, updated, and used correctly both as an employee and as an employer, while also testing related business logic, timestamps, embedded objects, and Spring Security integration.
    - Unit tests for Values - Verifies that the Values entity can be created, updated, compared, and represented reliably.
    - Unit tests for JWT - Unit test for security utility. It verifies that JSON Web Tokens (JWTs) are generated, validated, and expired correctly within the application.
    - Unit tests for UserService - Unit test for UserServiceImpl which manages users in the application. It uses JUnit5 and Mockito to mock dependencies and validate business logic.
  + Integration testing

Our integration testing validates how different components of the system interact, with a particular focus on authentication, authorization, and API workflows. Integration tests are created and maintained by developers. These tests have been run manually during the development of these functionalities via Postman. The collection can be found [here](https://metcs673-team-2.postman.co/workspace/METCS673-Team-2-Workspace~2e290ce4-6167-408d-8f8b-60e4fc88d7dd/collection/15824451-823da234-e01b-481d-8e5d-5c2efd03ebce?action=share&creator=15824451) and can be run manually at any time. These tests provide confidence that authentication, registration, and secure endpoint access behave as expected across the system, preventing regressions and ensuring reliability before deployment.

* + - Security Integration Tests (JUnit and MockMvc):

We maintain an automated test suite for our Spring Security and JWT setup. These tests verify that user authentication and role-based authorization function correctly when interacting with real endpoints, ensuring secure access control between users and admins.

* + - Postman API Test Collection:

We supplement automated integration tests with a Postman collection that validates key API endpoints:

* + - * Login Tests: Confirm that user credentials return a 200 OK response on successful login.
      * Registration Tests: Verify that new users can be created with all required fields (username, email, password, firstName, lastName, userType, and active status).
      * JWT Tests: Confirm that the authentication API returns a valid JWT token upon login.
      * Secure Endpoint Tests: Validate that authorized endpoints accept valid tokens and reject invalid or missing tokens.
      * Job Tests: To be completed
  + System Testing - System testing is not yet fully implemented, as we are still in the process of preparing the application for end-to-end validation of functionality. While the foundational components are in place, additional integration and acceptance tests need to be built before we can comprehensively verify complete business workflows across the system.
  + Acceptance Testing - Our acceptance testing framework is in its early stages of development. At present, we have implemented a basic login test that is undergoing peer review. The primary purpose of this test was to establish the foundation for using Selenium to drive our acceptance, system, and regression testing efforts moving forward. These tests will be built by our engineering team and will be reviewed collaboratively by peers to ensure coding standards and test coverage align. We will be using Selenium WebDriver with Java for UI automation, focused on simulating real user actions and verifying UI-level outcomes. Future iterations will expand coverage to more comprehensive user workflows.
  + Regression Testing - Our unit tests also function as a form of regression testing, since they are executed automatically with every code push to ensure that existing functionality continues to work as expected. This provides an early safeguard against unintended side effects from new changes. At present, we have a small subset of tests that require maintenance, and these are being addressed to restore full regression coverage.

# Manual Testing Report

In this section, you will give a detailed description of each manual test case performed and the result. If this is a previous You shall list what are existing tests developed in the previous semester and what are new tests developed currently.

Here is a sample template that can be used for each test case. For system tests or acceptance tests, you may also include some screenshots.

* Test case ID, name
* New or old:
* Test items: (what do you test )
* Test priority (high/medium/low)
* Dependencies (to other test case/requirement if any):
* Preconditions: (if any)
* input data:
* Test steps:
* Postconditions:
* Expected output:
* Actual output:
* Pass or Fail:
* Bug id/link: (this should link to your github issue id)
* Additional notes:

(You can use an additional spreadsheet for this section as well)

# Automated Testing Report

Describe briefly the automated testing you have done, including where the test code resides in your code repository, what test frameworks are used, and the screen shots or generated testing report.

All of our automated unit tests can be found under the code/backend/src/test directory of our repository, organized by package to mirror the structure of the main application. Our Selenium tests can be found in src/test, with our feature files to be built in the resources folder and our pages/steps to exist in the java folder. The frameworks we are using are JUnit5, Mockito, Spring Boot Test, Spring Test, and Selenium WebDriver.

* + JUnit5 - Core framework used in our unit tests
  + Mockito - Used for mocking dependencies that allows us to stub method calls and verify interactions
  + Spring Boot Test Framework - Used to load the full Spring context and specific beans for integration testing
  + Spring Test - Used MockMvc to simulate HTTP requests to controllers without needing to start a real server
  + Selenium WebDriver - Framework to drive browser testing and simulate end to end user behavior

Our unit test results can be found in the code/backend/target/surefire-reports folder.

# Testing Metrics

In this section, you shall report any metrics used for the evaluation, e.g. # of test cases, test coverage, defects rate, etc.

* **Test coverage:** At the moment we have a total of 118 unit tests
* **Test pass rate:** We have 16 failures at the moment that need to be addressed, leaving us with about an 86% passing rate on unit tests solely.

# References

# Glossary