**CS673 Software Engineering** 

**Team 6 - Blockbuster**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Alex Flinchum | QA Lead | *Alex Flinchum* | 09/22/2024 |
| Ricky Zheng | Configuration Leader | *Ricky Zheng* | 10/7/2024 |
| Rekik Mengstu | Requirement Leader | *Elizabeth Tyree* | 09/23/2024 |
| James Zheng | Security Leader | *James Zheng* | 9/23/2024 |
| Joshua Shilts | Leader | *Joshua Shilts* | 9/23/2023 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1.0.0** | **Alex** | **9/23/2024** | **Creation** |
| **2.0.0** | **Ricky Zheng** | **10/6/2024** | **Automated Testing Report (Pipeline changes)** |
| **2.0.1** | **Joshua Shilts** | **10/7/2024** | **Updates** |

[Testing Summary](#_heading=h.gjdgxs)

[Manuel Tests Reports](#_heading=h.30j0zll)

[Automated Testing Reports](#_heading=h.1fob9te)

[Testing Metrics](#_heading=h.3znysh7)

[References](#_heading=h.2et92p0)

[Glossary](#_heading=h.tyjcwt)

# Testing Summary

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

* + Unit Testing
    - Backend
      * Unit test for Flask Route calls
    - Frontend
      * Unit test for:
        + Clear button functionality
        + Submit Rest API call
    - Database
      * Unit test the creation of the database
  + Integration testing
  + System Testing
    - Test docker building docker image before commit to registry
  + Acceptance Testing
    - All Testing done this Iteration was Acceptance Testing, specifically manual testing.
    - The two test cases tested;
      * 1; Frontend | UI | Create Frontend MVP | Build the Application | Successful
      * 2; Backend | UI | Create Backend MVP | Run the Application | Successful
    - All tests had the expected outcomes.
  + Regression Testing

# 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.

* Test case ID, name:
  + 1; Frontend | UI | Create Frontend MVP | Build the Application | Successful
* New or old:
  + New
* Test items: (what do you test )
  + This test focuses on running the frontend from the developers machine
* Test priority (high/medium/low)
  + Medium
* Dependencies (to other test case/requirement if any):
  + Being able to run the project through Npm
* Preconditions: (if any)
  + N/A
* Input Data:
  + Npm start
* Test steps:
  + 1: pull down the latest dev branch
  + 2: Open a terminal (either in your ide or from your machine)
  + 3: cd into the frontend file
  + 4: execute npm start
* Postconditions:
  + None
* Expected output:
  + The fronted should be pulled up in your browser
* Actual output:
  + The input screen is pulled up in my browser
* Pass or Fail:
  + Pass
* Bug id/link: (this should link to your github issue id)
  + None
* Additional notes:
  + N/A
  + Test case ID, name
    - 2; Backend | UI | Create Backend MVP | Run the Application | Successful
  + New or old:
    - New
  + Test items: (what do you test )
    - This test focuses on running the backend through the frontend from the developers machine
  + Test priority (high/medium/low)
    - Medium
  + Dependencies (to other test case/requirement if any):
    - Being able to run the project through Npm
  + Preconditions: (if any)
    - N/A
  + Input Data:
    - Npm start
    - Test data for inputs
  + Test steps:
    - 1: pull down the latest dev branch
    - 2: Open a terminal (either in your ide or from your machine)
    - 3: cd into the frontend file
    - 4: execute npm start
    - 5: insert date into input section
    - 6: Click the Generate button
  + Postconditions:
    - None
  + Expected output:
    - The data should be displayed in a generated text on the web page (At this time an error message will display)
  + Actual output:
    - Error retrieving data
  + Pass or Fail:
    - Pass
  + Bug id/link: (this should link to your github issue id)
    - None
  + Additional notes:
    - N/A

# 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.

* + **GitHub Actions Pipeline Workflow**
    - A GitHub Actions pipeline workflow has been implemented where multiple jobs run after every commit or merge and occur on each project branch. Every job is dependent on a specific job that occurs before it. There are 3 flows: the backend flow, database flow, and frontend flow that contain similar jobs. These jobs are:
      * **Backend**
        + **Get-Branch-Name**

Sanitizes the current branch name to be used as variables in the following jobs below. This initial job needs to pass for both the backend and frontend to begin their job flow.

* + - * + **Docker-and-Python-Flask-Service**

Python Flask is installed.

Docker is used to build and run the Python backend service.

Test checks if Docker image and container are created and can be manipulated and shutdown.

* + - * + **Push Docker Python Flask to GitHub Package registry**

Upon successful completion of Docker-and-Python-Flask-Service job, the backend container is pushed to GitHub Package registry where each version of the package is designated by the branch that ran the pipeline. This is done before unit testing so that GitHub Actions has access to the package in the registry to use for unit testing jobs.

* + - * + **Run backend unit tests**

Upon successful completion of the Push Docker Python Flask to GitHub Package registry job, run a process\_query\_request\_test.py script to test the backend.

* + - * + **Run Pylint on Backend code**

Upon successful completion of Run Backend Unit Tests, the code is further vetted by running Pylint on the source files in the repo to report any vulnerabilities or coding standard issue with the code.

* + - * + **Publish Docker Python Flask to GitHub Package registry**

Upon successful completion of the Run PyLint on Backend code job, the backend container is pushed to GitHub Package registry again to update to the new version.

* + - * **Database**
        + **Get-Branch-Name**

Sanitizes the current branch name to be used as variables in the following jobs below. This initial job needs to pass for both the backend and frontend to begin their job flow.

* + - * + **Docker-and-Database-Service**

Docker is used to build and run the Python database service.

Test checks if Docker database image and container are created and can be manipulated and shutdown.

* + - * + **Push Docker Sqlite Database Image to GitHub Package registry**

Upon successful completion of Docker-and-Database-Service job, the database container is pushed to GitHub Package registry where each version of the package is designated by the branch that ran the pipeline. This is done before unit testing so that GitHub Actions has access to the package in the registry to use for unit testing jobs.

* + - * + **Run database unit tests**

Upon successful completion of the Push Docker Sqlite Database Image to GitHub Package registry to GitHub Package registry job, run a movies\_unit\_test.py to test the database container.

* + - * + **Publish Docker Sqlite Database Image to GitHub Package registry**

Upon successful completion of the Run database unit tests job, the database container is pushed to GitHub Package registry again to update to the new version.

* + - * **User Interface**
        + **Get-Branch-Name**

Sanitizes the current branch name to be used as variables in the following jobs below. This initial job needs to pass for both the backend and frontend to begin their job flow.

* + - * + **Docker-and-Node Service**

Node.js is installed along with npm.

Docker is used to build and run frontend web service.

Test checks if Docker image and container are created and can be manipulated and shutdown.

* + - * + **Push Docker Node Image to GitHub Package registry**

Upon successful completion of Docker-and-Node Service job, the package is pushed to GitHub Package registry where each version of the package is designated by the branch that ran the pipeline. This is done before unit testing so that GitHub Actions has access to the package in the registry to use for unit testing jobs.

* + - * + **Run frontend unit tests**

Upon successful completion of the Push Docker Node Image to GitHub Package registry job, run a submitform.test.js to test the frontend.

* + - * + **Publish Docker Node Frontend to GitHub Package registry**

Upon successful completion of the Run frontend unit tests job, the frontend container is pushed to GitHub Package registry again to update to the new version.

* + - The pipeline structure is a single workflow that can be divided into 3 flows run in parallel: a backend, database, and frontend flow with similar steps involved. As of 10/7/2024, a full pipeline workflow run takes approximately 12 minutes to complete..
    - The workflow file, “current-workflow.yml”, is located in the “.github/workflows/” folder at the top level of the dev branch.

# Testing Metrics

Every test case should have its desired outcome; if the desired outcome is met then the test will pass.

* **Successful Testing**
  + All stories that have implemented some functionality for the project, whether that is frontend or backend, should have one or two successful test cases.
  + The test cases will check that our functionality has the desired outcome.
* **Edge/Failed Testing**
  + Many stories that have implemented functionality in the project will have outcomes in which they could fail.
  + The test cases can range from checking that the correct data type is being used to an action that could break the functionality of the project.

# 

# References

**GitHub Packages :** <https://docs.github.com/en/packages/learn-github-packages/introduction-to-github-packages>

**GitHub Package: Container Registry:** <https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry>

**GitHub Packages: Publishing Docker Images:**

<https://docs.github.com/en/actions/use-cases-and-examples/publishing-packages/publishing-docker-images>

**GitHub Actions Job Dependency:**

<https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/using-jobs-in-a-workflow>

# Glossary

**API:** Application Programming Interface

**ID:** Identification

**MVP:** Minimum Viable Product

**npm:** Node Package Manager

**UI:** User Interface