Quality Assurance (QA) Plan

- Table of Contents

1. List of Contributors

2. Task Matrix

3. Quality Assurance (QA) Plan:

1. Quality Assurance Strategy

2. Quality Factors & Metrics

3. Test Plan

- List of Contributors

1. İbrahim Eren Yılmaz

2. Sencer Ali Şahin

3. Mete Oktar

4. Efe Arda Uzunova

- Task Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | İbrahim Eren Yılmaz | Sencer Ali Şahin | Mete Oktar | Efe Arda Uzunova |
| System Architecture & Technology Stack |  |  |  |  |
| Implementation (UI + Suggestor Classes) |  |  |  |  |
| Use Case Descriptions & Diagrams |  |  |  |  |
| Design Decisions & Comparisons |  |  |  |  |
| QA Strategy & Test Plan |  |  |  |  |
| Final Integration & Documentation |  |  |  |  |

**1. Quality Assurance Strategy**

**Overview**

We will use a **mix of unit tests, integration tests, and user acceptance tests**:

• **Unit Testing**: Test small pieces, like data loading or text “soup” creation.

• **Integration Testing**: Ensure the synergy between UI and Suggestor works (e.g., that the user input indeed flows to the recommendation).

• **User Acceptance Testing (UAT)**: Check if final recommendations match user expectations.

**Testing Methodologies**

1. **Unit Testing**

• Done using our own testing functions.

• Targets data cleaning functions, e.g., clean\_data(), create\_soup().

2. **Integration Testing**

• Manually run the full pipeline (UI → Generate Recommendations) to confirm no file path or indexing errors.

• Validate that the returned DataFrame from hybrid() is correct in shape and columns.

3. **Usability Testing**

• Observe a small group of users interacting with the UI.

• Collect feedback on clarity and ease of data entry.

**Manual Testing**

• The Tkinter UI usage.

• Exporting the Excel file and verifying correctness of the output file.

**2. Quality Factors & Metrics**

Below are the four quality factors, their description, and the measurement metrics.

|  |  |  |
| --- | --- | --- |
| Quality Factor | Description | **Measurement Metric** |
| Performance | Speed of generating recommendations | Average time (ms) to run hybrid() |
| Usability | Ease of use for end-users (UI clarity, steps to complete) | User satisfaction score from quick user surveys |
| Maintainability | Ease of modifying or extending the codebase | Code complexity (Cyclomatic Complexity) or lines of code in core modules |
| Security | Protection against potential attacks or unauthorized data exposure | Number of discovered security vulnerabilities (via scanning or penetration tests) |

1. **Performance**

• We measure how fast the system generates recommendations by logging the average time (in milliseconds) it takes for the hybrid() method to complete.

2. **Usability**

• Users will fill out a short survey regarding the UI’s intuitiveness. We track their satisfaction on a simple 1–5 scale and average the results.

3. **Maintainability**

• We check code complexity using a tool (e.g., coverage + cyclomatic complexity in pytest-cov). Lower complexity indicates better maintainability.

4. **Security**

• We will employ basic vulnerability scanning and code review to see how many potential security flaws (e.g., injection points, unsafe file operations, access control issues) surface. Minimizing vulnerabilities is our success metric.

**3. Test Plan**

**Test Cases**

Below is a sample subset of **5** test cases:

1. **TC1: Loading Existing Pickle Files**

**Precondition**: smd.pkl and ratings.pkl are present.

**Steps**:

1. Instantiate Suggestor with user data.

2. Observe if it bypasses CSV merging steps.

**Expected Result**: No errors; data quickly loads from pickles.

2. **TC2: No Existing Pickle Files**

**Precondition**: Remove or rename smd.pkl and ratings.pkl.

**Steps**:

1. Instantiate Suggestor.

2. Check if CSV merges are performed.

**Expected Result**: Data merges successfully; new .pkl files created.

3. **TC3: Hybrid Method with Valid Inputs**

**Precondition**: Provide valid movie titles and ratings.

**Steps**:

1. Fill out 4 movies + ratings in the UI, plus a 5th “target” movie.

2. Press Submit.

**Expected Result**: Output deneme\_gelismis.xlsx with ~20 recommended movies sorted by predicted rating.

4. **TC4: Invalid Movie Title**

**Precondition**: Enter a title that doesn’t match id\_map.

**Steps**:

1. Type a nonsense movie name, e.g. “ABCD1234.”

2. Press Submit.

**Expected Result**: The system warns the user (prints a message or logs an error) and does not break execution or produce an empty .xlsx file.

5. **TC5: Security Check**

**Precondition**: Attempt to provide malicious input (e.g. a string with special characters or partial code injection).

**Steps**:

1. Enter 'DROP TABLE Movies;-- or suspicious input in the movie title field.

2. Press Submit.

**Expected Result**: The system either sanitizes or rejects the invalid input and does not write it into CSV or cause an error.

**Bug Tracking**

• We will use a GitHub issue tracker.

• Each bug is labeled by severity (minor, major, or critical) and assigned to a team member.

• Once fixed, the commit message references the issue ID (e.g., “Fix #5”).