Verification and Validation Report: GenreGuru

Team 8 – Rhythm Rangers

Ansel Chen
Muhammad Jawad
Mohamad-Hassan Bahsoun
Matthew Baleanu
Ahmed Al-Hayali

March 11, 2025

1 Revision History

Date	**Version**	**Notes**
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

symbol	**description**
T	Test
UT	Unit Test
VnV	Verification and Validation

Contents

List of Tables

List of Figures

This document reports the results of executing the Verification and Validation Plan. VnV Plan

3 Functional Requirements Evaluation

This section details the results of the manual tests conducted for functional requirements, as outlined in the Verification and Validation (VnV) Plan. The results include expected vs. actual outcomes, and any issues encountered.

3.1 User Input and Interaction Tests

Test ID: UII-01
Description: Ensures that users can interact with the system by inputting data through wav file upload.
Test Steps Part 1:

1. Open the client application.
2. Upload a wav file.
3. Observe the system's response.

Expected Result: The system displays a confirmation that the file has been accepted, and the request process is initiated.
Actual Result: The system displays a confirmation that the file has been accepted, and the request process is initiated.

Test Steps Part 2:

1. Open the client application.
2. Search for song.
3. Select song for recommendations.
4. Observe the system's response.

Expected Result: The system displays search results according to the user search query. Upon song selection, the system displays song recommendations.

Actual Result: System accepts search query input and responds by displaying Spotify search results (normal behavior). Some UI elements have overflowing text (abnormal behaviour).

Test ID: UII-02
Description: Ensures that UI elements are interactive and respond to user actions.
Test Steps:

1. Open the client application.
2. Click buttons and interact with text fields.
3. Observe system responsiveness.

Expected Result: All UI elements function correctly and provide feedback.
Actual Result: Preview play button switches to stop button as expected. All UI elements function correctly and provide feedback.

3.2 Output Display Test

Test ID: OD-01
Description: Verifies that output is displayed correctly to users.
Test Steps:

1. Perform an analysis on an input song.
2. Check the output display in the client application.
3. Ensure recommended songs are correctly presented.

Expected Result: The analysis results should be clearly visible and formatted properly.
Actual Result: Recommendation results not displayed correctly. Blank page displayed (implementation error).

3.3 Data Processing and Feature Extraction Tests

Test ID: FE-01
Description: Confirms feature extraction occurs as intended (Requirement #3).

Input: A valid song reference link.
Output: Extracted song features (tempo, pitch, genre, etc.).
Test Steps:

1. Open the client application.
2. Input a valid reference song link.
3. Submit a request for feature extraction.
4. Observe and compare the extracted features with a known expected output.

Expected Result: The system successfully extracts correct song features (tempo, pitch, genre, etc.) that match the known expected output.
Actual Result: As this test requires for the full system integration to work, this test fails as the client application and the server are not integrated yet.

Test ID: AFD-01
Description: Ensures extracted features are presented clearly (Requirements #5 and #7).
Input: (None, occurs after feature extraction)
Output: Song features displayed in a readable format.
Test Steps:

1. Perform the feature extraction for a valid reference song.
2. Wait for the system to display extracted features.
3. Check that the displayed features are clear, correctly labeled, and readable.

Expected Result: The client application displays all extracted features (tempo, pitch, genre, etc.) in a clearly formatted and understandable layout.
Actual Result: We are no longer returning features to the users. We are now only focused on returning the recommended songs to the user. So this is no longer being tested.

3.4 Error Handling and Validation Tests

Test ID: IVED-01
Description: Checks input validation and error feedback (Requirements #6 and #8).
Input: Invalid or malformed song reference.
Output: Error message guiding the user.
Test Steps:

1. Open the client application.
2. Input an invalid or malformed song reference link.
3. Observe the system's response.

Expected Result: The system displays a clear error message indicating the input is invalid and provides guidance on how to correct it.
Actual Result: Not implemented yet.

Test ID: EF-01
Description: Confirms each component provides feedback when errors occur (Requirement #6).
Input: Deliberate errors introduced in any component.
Output: Specific error messages generated by each component.
Test Steps:

1. Introduce a deliberate error in one of the system components (e.g., corrupted data or forced exception).
2. Attempt to use the functionality related to that component.
3. Observe the displayed error messages.

Expected Result: The system displays a clear, component-specific error message indicating which part failed and why.
Actual Result: This is no longer a test that is required as the only errors that could be introduced are from the user end which is covered in test EF-01.

3.5 Connection and Data Flow Tests

Test ID: CDH-01
Control: Manual
Initial State: Client application is connected to the server and external APIs.
Input: User submits a request using a valid song link.
Output: Successful data retrieval and confirmation message.
Test Case Derivation: Verifies communication between client, server, and APIs (Requirement #2).
How test will be performed:

- The tester will submit a reference as input and observe the system processing the request.
- After processing, the tester will check logs confirming the request has been received.

Expected Result: The client receives a confirmation of successful retrieval, and logs show the incoming request from the client to the server.
Actual Result: Not implemented yet.

Test ID: EADR-01
Control: Manual
Initial State: Server and client are operational.
Input: A song reference link.
Output: Song data is retrieved from external API.
Test Case Derivation: Ensures valid links trigger API data retrieval (Requirement #4).
How test will be performed:

- The tester will input a valid song link and check the system for API data retrieval confirmation.

Expected Result: The client application displays a success message, and external API data appears in the system logs or interface.
Actual Result: Not implemented yet.

4 Nonfunctional Requirements Evaluation

This section details the results of the manual tests conducted for nonfunctional requirements, as outlined in the Verification and Validation (VnV) Plan. The results include expected vs. actual outcomes, and any issues encountered.

4.1 Minimalist Layout

Test ID: TAPR1
Description: Ensures that the interface follows a minimalist layout design.
Test Steps:

1. Open the application.
2. Visually inspect the layout for clutter and distractions.

Expected Result: The layout is clean and minimalistic.
Actual Result: The layout is clean and minimalistic and there is no clutter.

4.2 High Contrast

Test ID: TAPR2
Description: Ensures that UI contrast meets readability standards.
Test Steps:

1. Open the application.
2. Verify that all text and UI elements have sufficient contrast using WCAG guidelines.

Expected Result: Text and elements should be readable against the background.
Actual Result: Text and UI elements have sufficient contrast using WCAG guidelines.

4.3 Intuitive Navigation

Test ID: TAPR3
Description: Verifies that users can easily navigate the interface.
Test Steps:

1. Open the application.
2. Attempt to access all major features within three clicks.

Expected Result: Users can reach important functions quickly.
Actual Result: Users can access all major functions within three clicks (search, upload, request).

4.4 Consistent Button Styles

Test ID: TSTR1
Description: Ensures that button styles are consistent.
Test Steps:

1. Open the application.
2. Visually inspect all buttons for uniform design.

Expected Result: All buttons have the same size, shape, and color scheme.
Actual Result: All buttons have the same size, shape, and color scheme.

4.5 Tooltip Visibility

Test ID: TEUR1
Description: Ensures that tooltips appear when hovering over interactive elements.
Test Steps:

1. Hover over all interactive elements.
2. Verify that tooltips appear with relevant descriptions.

Expected Result: Tooltips should be visible and contain helpful information.
Actual Result: Not implemented.

4.6 Customizable Color Themes

Test ID: TPIR1
Description: Ensures that users can change color themes without issues.
Test Steps:

1. Open the application settings.
2. Change to different color themes.
3. Verify that all UI elements update accordingly.

Expected Result: The color theme updates correctly without graphical glitches.
Actual Result: Not implemented.

4.7 Initial Tutorial

Test ID: TLR1
Description: Verifies that a tutorial is displayed for first-time users.
Test Steps:

1. Open the application for the first time.
2. Observe if a tutorial appears.

Expected Result: The tutorial should guide the user through the main features.
Actual Result: Not implemented yet.

4.8 Query Request Time Precision

Test ID: TPAR1
Description: Verifies that the system accurately displays the query request time for any valid query.
Input/Condition: Issue any valid query.
Output/Result: Server returns response, and the client application displays the precise query request time.
Test Steps:

1. Use a web driver (e.g., Selenium) or open the client application directly.
2. Input a valid query and submit it.
3. Record the displayed query request time for verification.

Expected Result: The displayed query request time is accurate and precise (no significant deviation from actual request time).
Actual Result: We cannot test this yet as the system integration is not yet complete, so this test Fails.

4.9 Rounding Accuracy

Test ID: TPAR2
Description: Ensures values with decimals are rounded accurately according to the specification.
Input/Condition: Input values with decimals.
Output/Result: Values are rounded accurately.
Test Steps:

1. Provide pre-defined decimal values (via the relevant input fields or tests).
2. Trigger the system's rounding function.
3. Observe the resulting rounded values.

Expected Result: The values are rounded precisely according to the specified rounding rules (e.g., 2 decimal places).
Actual Result: We are no longer testing rounding accuracy as our feature extraction needs to be as accurate as possible. Thus the system will use as many digits as it can.

4.10 Fault Tolerance

Test ID: TRAR1
Description: Verifies the server's ability to remain operational under any load state (idle, little load, intermediate, or strenuous), including transitions between these states.
Test Steps:

1. Ensure the server is operational in any initial load state (idle, little, intermediate, or strenuous).
2. Transition the server through various load states over a 3-day monitoring period (e.g., from idle to little load, then from little to intermediate, and so on).
3. Use the `uptime` command on Ubuntu Server periodically to track server availability.
4. (Optional) Extend monitoring to 30 days or more if feasible.

Expected Result:

- The server remains fully operational during and after state transitions (idle, little, intermediate, or strenuous).
- Observed `uptime` indicates stable performance with minimal or no downtime.

Actual Result: Fail. Server has not been completed yet, thus cannot be tested.

4.11 Minimum Concurrent Users Load

Test ID: TCR1
Description: Ensures the system can handle the expected number of concurrent users without performance degradation.
Test Steps:

1. Start the system in an idle state.
2. Use a load testing tool (e.g., JMeter) to simulate multiple users logging in and performing actions simultaneously.
3. Gradually increase the number of concurrent users to the expected limit (and potentially beyond it to test the upper threshold).
4. Observe system response times, error rates, and resource utilization.

Expected Result: The system successfully handles the predefined concurrent user limit without critical errors or significant performance degradation (e.g., no timeouts or crashes).
Actual Result: We are no longer supporting concurrent users. thus, we are no longer testing this.

4.12 Server Device

Test ID: TEPER1
Description: Confirms that the physical server hardware matches the specified make and model (Dell OptiPlex 3050).
Test Steps:

1. Ensure the server is available for inspection.
2. Visually inspect the server hardware (using “los ojos”).
3. Verify that the make and model are Dell OptiPlex 3050.

Expected Result: Server is confirmed to be a Dell OptiPlex 3050 based on visual inspection and any accompanying documentation.
Actual Result: Pass. The server is confirmed to be a Dell OptiPlex 3050 based on visual inspection.

4.13 Tutorial Completion Time

Test ID: TLR2
Description: Ensures that users can complete the tutorial within five minutes.
Type: Dynamic, Manual
Initial State: Tutorial in progress.
Input/Condition: Measure time for tutorial completion.
Output/Result: Users complete the tutorial within 5 minutes.
Test Steps:

1. Begin the tutorial as a new user.
2. Track the time taken to finish all tutorial steps.

Expected Result: Completion of the tutorial in 5 minutes or less.
Actual Result: Not implemented yet.

4.14 Friendly Feedback

Test ID: TUPR1
Description: Ensures that the system provides friendly, clear feedback in error states.
Type: Static, Manual
Initial State: Error states are accessible.
Input/Condition: Trigger common errors.
Output/Result: System provides friendly, clear feedback.
Test Steps:

1. Manually trigger typical error conditions in the application.

2. Observe the content and tone of the error messages.

Expected Result: Polite and helpful error messages that guide the user to resolve the issue.
Actual Result: ?

4.15 Standardized Iconography

Test ID: TUPR2
Description: Verifies that interface iconography is standardized and inoffensive.
Type: Static, Manual
Initial State: A subset of complete interface iconography is available.
Input/Condition: Developer reviews subset of complete interface iconography.
Output/Result: Subset of complete interface iconography is deemed standard and inoffensive.
Test Steps:

1. Manually review each icon in the subset.

2. Confirm consistency and clarity of the iconography.

Expected Result: All icons follow a consistent style and contain no offensive or confusing imagery.
Actual Result: ?

4.16 Accessible Fonts

Test ID: TACR1
Description: Ensures font choices meet WCAG 2.2 accessibility guidelines.
Type: Static, Manual
Initial State: Font-defining code is available.
Input/Condition: Developer reviews font-defining code.
Output/Result: Font-defining code is deemed to only contain accessible fonts in accordance with WCAG 2.2.
Test Steps:

1. Manually inspect the code for font definitions.

2. Compare chosen fonts against WCAG 2.2 recommendations.

Expected Result: All fonts meet or exceed minimum accessibility standards, ensuring readability.
Actual Result: ?

4.17 Color Blind Mode

Test ID: TACR2
Description: Confirms that the system supports a color blind visibility mode.
Type: Dynamic, Manual
Initial State: Interface set to default visibility mode.
Input/Condition: Enable color blind mode.
Output/Result: Interface set to color blind visibility mode.
Test Steps:

1. Use the application's settings to enable color blind mode.
2. Observe all interface elements to confirm consistent color adjustments.

Expected Result: The color scheme updates accurately for color blind accessibility.
Actual Result: Not implemented yet.

4.18 Data Storage

Test ID: TCR2
Description: Verifies the database stores the necessary data related to songs and queries.
Type: Dynamic, Automated
Initial State: Mock database active.
Input/Condition: Issue any valid query.
Output/Result: Database stores appropriate data related to songs and query information.
Test Steps:

1. Use frameworks like `factoryboy` and Pytest fixtures to mock a database instance.

2. Submit a valid query, capturing how the system processes and stores the data.

3. Verify that the stored records match expected fields and content.

Expected Result: The system reliably populates the database with accurate song and query data.
Actual Result: Not implemented yet

5 Productization Requirements Evaluation

This section details the results of the manual tests conducted for productization requirements, as outlined in the Verification and Validation (VnV) Plan. The results include expected vs. actual outcomes, and any issues encountered.

5.1 Production Readiness Verification

Test ID: PRR1
Description: Verifies production readiness by running the system end to end.
Input/Condition: Perform a complete end-to-end run of all critical functionalities.
Output/Result: Confirms system stability and readiness for production.
Test Steps:

1. Open the client application.

2. Perform major functionalities (e.g., referencing a song, extracting features, handling error scenarios).

3. Observe whether the system runs without critical errors from start to finish.

Expected Result: The system runs all processes end to end smoothly, indicating it is production-ready.
Actual Result: We cannot test this yet as the system integration is not yet complete, so this test Fails.

5.2 Ease of Code Updates

Test ID: TMR1
Description: Reviews the codebase structure for easy updates (Requirements #X – fill as needed).
Input/Condition: Review of the code structure and modular design.
Output/Result: Codebase is structured and documented for maintainability.
Test Steps:

1. Examine the project's folder structure and modular design.
2. Evaluate documentation and comments for clarity and ease of updates.
3. Note any obstacles that might complicate future changes.

Expected Result: The codebase is modular, well-documented, and easily maintainable.
Actual Result:

- Overall Fail
- Codebase is Modular (**pass**)
  There is file separation in folders for each major section (recommend, server, featurizer). Testing documentation and files are in their own dedicated folders as well. There are no hanging plots.
- Evaluate documentation (**FAIL**) due to being incomplete and is currently outdated. Will test this again once all documentation has been evaluated/updated.
- Easily Maintainable (**pass**) because our codebase is modular it is easily maintainable. Also, with the use of GitHub, branches, issues etc. we have been able to keep things good? (update this last sentence)

5.3 Access Logs for User Sessions

Test ID: TAUR1
Description: Ensures all user session activities are accurately captured in the access logs.
Test Steps:

1. Confirm the system is active with at least one or more user sessions running.
2. Access the user session logs (e.g., via server logs or application log interface).
3. Compare the log entries against actual user actions (logins, file uploads, queries, etc.).

Expected Result: All user activities (start session, actions taken, logout) appear correctly in the logs, with accurate timestamps and details.
Actual Result: We are no longer supporting concurrent users and as such have no need to capture user session activities. thus, we are no longer testing this.

5.4 Multilingual Support

Test ID: TCUR1
Description: Verifies that the system can adapt to different languages without errors.
Test Steps:

1. Open the system interface.
2. Switch the display language to various available language options (e.g., English, Spanish, French).
3. Observe interface text, menus, and labels for correct translations and layout.

Expected Result: The user interface correctly updates all relevant text to the selected language, with no formatting issues or missing translations.
Actual Result: No longer being tested as the system will only be available in english.

5.5 Compliance with Copyright Laws

Test ID: TLGR1
Description: Ensures all music content adheres to copyright requirements.
Test Steps:

1. Open the content library (music files, metadata, etc.).
2. Inspect each file and confirm that it has the proper licensing and copyright attributions.
3. Verify that any usage terms (e.g., Creative Commons, royalty-free licensing) are properly documented.

Expected Result: All music content includes valid copyright attributions, matching the requirements of relevant laws and licensing agreements.
Actual Result: Pass. we have verified that we comply with the developer regulations for Deezer, Spotify and Apple Music for how we are allowed to use their music.

5.6 Adherence to Data Protection Regulations

Test ID: TLGR2
Description: Verifies that the system's user data management complies with applicable data protection regulations (e.g., GDPR, CCPA).
Test Steps:

1. Review the system's data management policies and documentation.
2. Inspect any data storage and access control mechanisms (e.g., encryption, restricted access).
3. Confirm that the collection and handling of user data follow legal regulations (e.g., consent, right to be forgotten, data retention limits).

Expected Result: User data is handled in accordance with all relevant data protection laws (proper consent obtained, secure storage, compliance with user rights, etc.).
Actual Result: Pass. We are not collecting user data as the song name is part of Spotify, Deezer,, and Apple Music and we follow their developer regulations. As for audio files uploaded by the user, we do not store that audio file or the features associated with it.

5.7 Database Backup

Test ID: TIR1
Type: Static & Manual, Dynamic & Automated
Initial State: Database layout configured & backup code is complete.
Input/Condition: Conduct code review to ensure backup functionality is correct, and simulate it running (on-command rather than weekly) to confirm backup success.
Output/Result: Database backup functionality is found to be correct, with ad-hoc generated backup artifacts to confirm it live.
How test will be performed:

- Conduct a code review/walkthrough.
- Use both unit and integration testing through Pytest with fixtures (alongside `factoryboy`) to ensure correct backup artifacts are generated.

Expected Result: Backup artifacts are produced successfully, matching the current database state with no data loss.
Actual Result: Not implemented yet.

5.8 Database Deduplication

Test ID: TIR2
Type: Static & Manual, Dynamic & Automated
Initial State: Database layout configured & deduplication code is complete.
Input/Condition: Conduct code review to ensure deduplication functionality is correct, and insert duplicate records to ensure the mechanism prevents insertion or "fails loudly."
Output/Result: Database deduplication functionality is found to be correct, with induced duplicate insertions prevented.
How test will be performed:

- Conduct a code review/walkthrough.
- Use both unit and integration testing through Pytest with fixtures (alongside `factoryboy`) to ensure correct deduplication behavior.

Expected Result: Duplicate records are either blocked or promptly flagged, with no persistent duplicates.
Actual Result: Not implemented yet.

5.9 Data Encryption Verification

Test ID: TPR1
Type: Static, Manual
Initial State: Database system in use.
Input/Condition: Inspect database for encryption protocols.
Output/Result: All sensitive data is encrypted in storage.
How test will be performed:

- Review encryption settings in the database configuration.

Expected Result: Sensitive records (e.g., user data) are encrypted at rest, and no plain-text sensitive data is stored.
Actual Result: Not implemented yet.

5.10 Licensed Song Access

Test ID: TMR1
Type: Dynamic, Automated
Initial State: Mock database active with multiple queries from different (at least 2 unique) users already loaded.
Input/Condition: Issue queries to access songs requested by other users, not the current user.
Output/Result: Database returns an empty response because the requesting user does not have access (or a license) to the songs uploaded/licensed by the other user(s).
How test will be performed:

- Use frameworks like `factoryboy` and Pytest's fixtures to mock a database and entries.
- Issue queries, capture the response, and verify that it is empty.

Expected Result: The system should return an empty response when a user attempts to access songs licensed to another user.
Actual Result: Not implemented yet.

6 Comparison to Existing Implementation

(This section does not apply to our project, so this section will be left empty).

7 Unit Testing

Most files—excluding the third-party library, top-level, and GUI modules—had dedicated unit test files. We use tox to run all these tests as a single suite. The coverage report confirms that all tests have passed successfully. See Code Coverage Metrics Section 12.

8 Changes Due to Testing

There were a few bugs that were caught as a result of testing, most of the issues came from module imports or pytest configuration issues. For feedback from users and supervisor, we will make changes after the system has been fully integrated

9 Automated Testing

Automated testing has not yet been set up, but we hope to have that set up before Rev 1.

10 Trace to Requirements

See Table 1 for trace to functional requirements and Table 2 and 3 for trace to nonfunctional requirements

10.1 Traceability Between Test Cases and Requirements

11 Trace to Modules

11.1 Traceability Between Test Cases and Modules

Table 4 maps each test case to its corresponding module(s) from the system architecture.

	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9
UII-01	X								
CDH-01		X							
FE-01			X						
EADR-01				X					
AFD-01					X		X		
IVED-01						X		X	
EF-01						X			
OD-01							X		
UII-02									X

Table 1: Traceability Matrix Showing the Connections Between the Tests and Functional Requirements

Test Case	**Module(s) Tested**
UII-01	GUI Module, Client Communication Module
UII-02	GUI Module, Search Query Module
CDH-01	Driver Module
FE-01	Feature Extraction Modules (All)
EADR-01	Server Communication Module, Client Communication Module
AFD-01	Audio File Input Module
IVED-01	Instrument Type Feature Extraction Module, Vocal Gender Feature Extraction Module
EF-01	Dynamic Range Feature Extraction Module
OD-01	Program Results Interface, GUI Module
APR1	GUI Module
APR2	GUI Module
APR3	GUI Module, Recommendation Module
STR1	GUI Module
EUR1	GUI Module
PIR1	GUI Module, Settings Module
LR1	GUI Module, Program Results Interface
LR2	GUI Module, Program Results Interface
UPR1	GUI Module, Program Results Interface
UPR2	GUI Module
ACR1	GUI Module, Settings Module
ACR2	GUI Module, Settings Module
PAR1	Database, Spotify API, Deezer API

	APR1	APR2	APR3	STR1	EUR1	PIR1	LR1	LR2	UPR1	UPR2	ACR1	ACR2	PAR1	PAR2
APR1	X													
APR2		X												
APR3			X											
STR1				X										
EUR1					X									
PIR1						X								
LR1							X							
LR2								X						
UPR1									X					
UPR2										X				
ACR1											X			
ACR2												X		
PAR1													X	
PAR2														X

Table 2: Traceability Matrix Showing the Connections Between the Tests and Non-Functional Requirements 1-14

	RAR1	CR1	CR2	EPER1	PRR1	MR1	ACCR1	IR1	IR2	PR1	AUR1	CUR1	LGR1	LGR2
RAR1	X													
CR1		X												
CR2			X											
EPER1				X										
PRR1					X									
MR1						X								
ACCR1							X							
IR1								X						
IR2									X					
PR1										X				
AUR1											X			
CUR1												X		
LGR1													X	
LGR2														X

Table 3: Traceability Matrix Showing the Connections Between the Tests and Non-Functional Requirements 15-28

12 Code Coverage Metrics

The code coverage was given by pytest-cov:

```
---------- coverage: platform win32, python 3.10.11-final-0 ----------
Name                                               Stmts   Miss  Cover   Missing
--------------------------------------------------------------------------------
src\featurizer\Audio_Splitter.py                      20      0   100%
src\featurizer\Beats_Per_Minute_Featurizer.py         29      0   100%
src\featurizer\Dynamic_Range_Featurizer.py            12      0   100%
src\featurizer\Instrumentalness_Featurizer.py         11      0   100%
src\featurizer\Key_and_Scale_Featurizer.py            37      0   100%
src\featurizer\RMS_Featurizer.py                      12      0   100%
src\featurizer\Spectral_Bandwidth_Featurizer.py       24      0   100%
src\featurizer\Spectral_Centroid_Featurizer.py        28      0   100%
src\featurizer\Spectral_Contrast_Featurizer.py        35      0   100%
src\featurizer\Spectral_Flux_Featurizer.py            22      0   100%
src\featurizer\Spectral_Rolloff_Featurizer.py         37      0   100%
src\featurizer\__init__.py                             0      0   100%
src\featurizer\main_featurizer.py                     90      0   100%
src\recommendation\__init__.py                         0      0   100%
src\recommendation\recommend.py                       24      0   100%
--------------------------------------------------------------------------------
TOTAL                                                381      0   100%
```

References

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

1. **What went well while writing this deliverable?**

   The structure of the VnV Report closely followed the VnV Plan, which made it easier to organize and document the tests systematically. Since many of the tests were manually conducted, this reduced the complexity of setting up automated test scripts. Additionally, having well-defined requirements and a structured approach helped in systematically evaluating functional and nonfunctional aspects. Collaboration within the team also went smoothly, as members were clear on their assigned sections.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**

   One major challenge was ensuring that the tests were properly mapped to the correct functional and nonfunctional requirements, as the original VnV Plan did not anticipate all the nuances of implementation. Some tests had to be modified or removed due to changes in the project scope. Another challenge was handling LaTeX formatting issues, particularly in structuring tables and ensuring they fit within the page. These were resolved by adjusting column widths, reducing text size, and restructuring tables for better readability.

3. **Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g., your peers)? Which ones were not, and why?**

   The test cases and evaluation methods were largely derived from discussions with peers and feedback from the client (or proxy). Specifically, the evaluation criteria for usability-related tests (e.g., UI consistency, accessibility features, and user interaction feedback) were refined based on input from potential users. However, the documentation of actual test results and the refinement of the VnV Report structure were mostly internally decided by the team. This was because execution of the tests was an internal process, and modifications had to be made based on practical constraints rather than external input.

4. **In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality?**

   The VnV Plan initially assumed a higher level of automated testing and unit test coverage. However, during implementation, it became clear that many of the usability and UI-based tests were better suited for manual evaluation. This led to modifications in the plan, where some unit tests were omitted in favor of structured manual testing. Additionally, some tests became irrelevant due to changes in project scope, requiring updates to the test suite.

   These changes occurred because certain features evolved during development, making some planned tests obsolete or requiring new test criteria. Anticipating such changes in future projects would involve incorporating a degree of flexibility in the VnV Plan, ensuring that it accommodates evolving requirements and implementation constraints. A more iterative approach, where validation strategies are reassessed at each milestone, would help mitigate these deviations in future projects.