Software Requirements Specification
Marito Multilingual Terminology PWA

Team Name: Velox

June 26, 2025

Contents

1. Introduction

1.1. Purpose

The purpose of this document is to define the software requirements of the Marito system, a multilingual terminology web application developed as part of the COS 301 Capstone project at the University of Pretoria. This document is intended to guide the development team, ensure alignment with client expectations, and serve as a reference for future maintenance, testing, and extension of the platform.

1.2. Scope

Marito is a multilingual, progressive web application (PWA) designed to provide users with access to curated statistical terminology in all 11 official South African languages. The system enables users to search and browse glossary terms, submit suggestions, and engage through a gamified contribution model. Marito supports offline functionality, glossary versioning, contributor uploads, and plugin-based feature extensions including AI-enhanced semantic search and analytics. The system is built using a microkernel-microservices architecture for flexibility, modularity, and future extensibility.

1.3. Intended Audience

This document is primarily intended for:

- **Developers and engineers** responsible for implementing and maintaining the system.
- **Project stakeholders and academic supervisors** overseeing the design, quality, and alignment of Mavito with its educational goals.
- **Testers and QA personnel** who require a clear specification of expected system behavior.
- **Future contributors** who may extend the application or onboard new glossaries and features.

1.4. Document Overview

The remainder of this document includes detailed user stories, functional and non-functional requirements, architectural and design patterns, service contracts, constraints, and system models. It is structured to follow industry best practices and academic guidelines for software specification documents, ensuring clarity and completeness for all stakeholders.

1.5. Definitions, Acronyms, and Abbreviations

- **PWA**: Progressive Web Application
- **CI/CD**: Continuous Integration and Continuous Deployment

2. User Stories

This section outlines the high-level user goals and core system interactions that informed the functional requirements. Detailed use case specifications and diagrams can be found in the official project repository.

2.1. User Story #1: Sync Updates

ID: US001 (Mavito Project)
Title: Automatic Synchronization of Downloaded Lexicon Data
As a: Mavito application user (e.g., language enthusiast, NLP researcher, student) who has downloaded lexicon data for offline use,
I want: the application to automatically check for and apply updates to my downloaded lexicon data whenever I am online,
So that: I can be confident that I am always working with the most current and accurate version of the linguistic information without needing to manually re-download or check for updates.

Acceptance Criteria:

1. **Automatic Update Check:**
   - **Given** I have previously downloaded lexicon data,
   - **And** I open the Mavito application with an active internet connection,
   - **Then** the application automatically initiates a check for updates to my downloaded data against the central repository in the background.
2. **Notification of Available Updates (Recommended):**
   - **Given** updates are available for my downloaded data,
   - **Then** the application clearly notifies me that updates are available (e.g., via a subtle in-app notification, a badge on a settings icon).
   - **And** the notification provides an option to view details about the updates (e.g., number of terms changed, lexicon version).
3. **Update Process:**
   - **Given** updates are available and I have an active internet connection,
   - **Then** the application allows me to initiate the download and application of these updates (or this happens automatically, depending on user settings or application design).
   - **And** the application provides clear feedback on the progress of the update (e.g., download progress bar, installation status).
   - **And** the update process is efficient and minimizes data usage (e.g., by only downloading changes/deltas if possible, rather than the entire dataset, for future enhancements).
4. **Successful Update:**

- **Given** the update process completes successfully,
- **Then** my locally stored lexicon data reflects the latest version from the central repository.
- **And** I receive a confirmation message that the data has been updated.

5. **Handling No Updates:**
   - **Given** I am online and no updates are available for my downloaded data,
   - **Then** the application does not interrupt my workflow with unnecessary notifications (or provides a subtle indication that data is "up-to-date").
6. **Offline State Post-Sync:**
   - **Given** my data has been successfully synced,
   - **Then** the newly updated data is fully accessible offline.
7. **Error Handling - Interrupted Connection:**
   - **Given** an update is in progress and my internet connection is lost,
   - **Then** the application gracefully pauses or stops the update process.
   - **And** I am notified of the interruption.
   - **And** the application attempts to resume the update when the connection is re-established, or allows me to manually retry.
   - **And** my previously downloaded (pre-update attempt) data remains intact and usable.
8. **Error Handling - Sync Conflict (Advanced Consideration for Future):**
   - **Given** a conflict occurs during synchronization (e.g., if local modifications were possible vs. server changes),
   - **Then** the system has a defined strategy for conflict resolution (e.g., prioritizes server version for this project's scope, notifies user if manual intervention were ever needed).
9. **User Control (Optional Enhancement):**
   - **Given** I am a user concerned about data usage or update timing,
   - **Then** I may have an option in settings to control sync behavior (e.g., sync only on Wi-Fi, schedule syncs, manual sync only).

Notes/Assumptions:

- This user story assumes that users primarily download data for offline reading and searching. Contributions and comments are made while online and sent to a central repository (as per Functional Requirement FR4.3).
- The complexity of only downloading changes/deltas (Acceptance Criterion 3.4) can be significant. For an initial implementation, syncing entire updated files might be a simpler starting point, with delta updates as a future enhancement.

- Conflict resolution (Acceptance Criterion 8) is likely simplified if the local data is treated as a read-only cache that gets overwritten by server updates, which aligns with the current understanding of the Mavito project.

2.2. User Story #2: Download Language Resources for Offline Access

ID: US002 (Mavito Project)
Title: Download Select Language Resources for Offline Use
As a: Mavito application user (e.g., language enthusiast, NLP researcher, student),
I want: to be able to select and download specific language resources (like individual lexicons, glossaries, or dictionaries) to my device,
So that: I can access and use this information even when I do not have an active internet connection.

Acceptance Criteria:

1. **Discover and Select Resources for Download:**
    - **Given** I am browsing the available language resources within the application (while online),
    - **Then** I can clearly identify which resources are available for download.
    - **And** I can select one or more language resources to download.
    - **And** the application shows the estimated size of the selected resource(s) before initiating the download.
2. **Initiate Download:**
    - **Given** I have selected one or more language resources for download,
    - **Then** I can initiate the download process with a clear action (e.g., a "Download" button).
3. **Download Process Feedback:**
    - **Given** a download is in progress,
    - **Then** the application provides clear visual feedback on the download status (e.g., progress bar, percentage complete, estimated time remaining).
    - **And** I can continue to use other online features of the application while a download is in progress (background download).
    - **And** I have the option to pause and resume a download if needed.
    - **And** I have the option to cancel an ongoing download.
4. **Successful Download and Storage:**
    - **Given** a language resource download completes successfully,
    - **Then** the application notifies me that the download is complete.
    - **And** the downloaded resource is stored locally on my device in a way that the Mavito application can access it offline.

- **And** the application clearly indicates which resources have been successfully downloaded and are available offline.

5. **Managing Downloaded Resources:**
   - **Given** I have downloaded language resources,
   - **Then** I can view a list of all my downloaded resources within the application.
   - **And** I can remove/delete downloaded resources from my device to free up storage space.
   - **And** the application shows the amount of local storage space currently used by downloaded Mavito resources.
6. **Error Handling - Insufficient Storage:**
   - **Given** I attempt to download a resource,
   - **And** my device has insufficient storage space,
   - **Then** the application informs me of the insufficient storage and the download does not proceed (or pauses until space is freed).
7. **Error Handling - Interrupted Connection During Download:**
   - **Given** a download is in progress and my internet connection is lost,
   - **Then** the application gracefully pauses the download process.
   - **And** I am notified of the interruption.
   - **And** the application attempts to resume the download automatically when the connection is re-established, or allows me to manually resume.
8. **Error Handling - Download Failure:**
   - **Given** a download fails for reasons other than connection loss or insufficient storage (e.g., server error, corrupted file),
   - **Then** the application notifies me of the failure and provides a reason if possible.
   - **And** I have the option to retry the download.
9. **Accessing Downloaded Resources Offline:**
   - **Given** I have successfully downloaded a language resource,
   - **And** I am offline,
   - **Then** I can access and use that downloaded resource within the Mavito application.

Notes/Assumptions:

- The user must be online to browse and initiate downloads.
- The application will need appropriate permissions to write to local device storage.
- The format of the downloaded data should be optimized for offline use and efficient storage.

- This user story focuses on the *download* functionality. A separate user story would cover the specifics of *accessing and using* the data offline (e.g., offline search within downloaded resources).

2.3. User Story #3: Search Feature

ID: US003 (Mavito Project)
Title: Multilingual Term Search
As a: Mavito application user (e.g., casual user, linguist, or contributor),
I want: to search across multiple multilingual glossaries and dictionaries using filters and smart suggestions,
So that: I can quickly find definitions, translations, and related entries in my preferred language.

Acceptance Criteria:

1. **Query Input:**
   - **Given** I am on the main search page,
   - **When** I enter a query into the search bar,
   - **Then** the system searches across all selected data sources and returns matching terms.
2. **Filter Options:**
   - **Given** I want to refine my search,
   - **Then** I can apply filters like language, part of speech, or glossary type before or after submitting the query.
3. **Fuzzy Search (Optional):**
   - **Given** my query has a typo or partial match,
   - **Then** the system offers similar results using fuzzy matching.
4. **AI-Powered Suggestions (Optional):**
   - **Given** I start typing in the search bar,
   - **Then** I see AI-generated autocomplete suggestions based on common terms or semantic matches.
5. **Search History:**
   - **Given** I have searched for terms in the past,
   - **Then** the system shows my recent queries and allows re-searching them.
6. **Sorting and Result Presentation:**
   - **Given** the results are displayed,
   - **Then** I can sort them by relevance, alphabetical order, or popularity,
   - **And** each result shows the term, language, and a brief definition snippet.

Assumptions:

- Offline support will allow previously downloaded glossaries to be searched.
- Filters and sort options persist across sessions.
- AI and fuzzy search are optional enhancements.
- Glossary datasets are already available and preloaded or fetched via sync.

2.4. User Story #4: Feedback Submissions

ID: US004 (Mavito Project)
Title: User Contributions and Feedback
As a: logged-in user of the application,
I want: to be able to comment on terms and submit feedback or error reports,
So that: I can contribute to improving the accuracy and usefulness of the data content.

Acceptance Criteria:

1. **Commenting on Terms:**
   - **Given** I am logged into my account,
   - **And** I am viewing a term or entry,
   - **Then** I should see a comment section below the entry,
   - **And** I should be able to submit my own comment.
2. **Submitting Feedback:**
   - **Given** I am logged into my account,
   - **And** I am viewing a specific term or entry,
   - **And** I choose to provide feedback,
   - **Then** I should be presented with a feedback form,
   - **And** I should receive a confirmation message after successfully submitting the feedback.
3. **Voting on User Contributions:**
   - **Given** I am logged into my account,
   - **And** I am viewing a comment or suggestion submitted by another user,
   - **Then** I should see upvote and downvote buttons associated with it,
   - **And** I should be able to cast one vote per contribution,
   - **And** the vote count should update immediately after I vote.
4. **Approval Status of Feedback:**
   - **Given** I have submitted feedback for a term,
   - **And** the feedback has been approved by a moderator,

- **Then** the approved feedback should be integrated into the application content,
- **And** I should receive a notification confirming the integration.

5. **Marking Approved Submissions:**
   - **Given** user feedback or content has been approved and integrated,
   - **Then** the associated entry should be visibly marked as a "User Submission" to distinguish it from original content.
6. **Rewarding User Contributions (Optional Gamification):**
   - **Given** I am logged in and submit a comment, feedback, or report,
   - **When** my contribution meets a predefined threshold (e.g., approved, upvoted),
   - **Then** I should earn points or badges for that action,
   - **And** my contribution stats should be viewable in my profile.

Notes/Assumptions:

- Only logged-in users can comment, submit feedback, vote, or receive contribution rewards.
- Users must have verified accounts to interact with community features (e.g., comments, voting).
- Rate limiting will be implemented to prevent spam submissions.

2.5. User Story #5: Gamification Feature

ID: US005 (Mavito Project)
Title: Contribution-Based Rewards and Progress Tracking
As a: frequent Mavito contributor (e.g., user who submits suggestions or comments),
I want: to earn points, unlock badges, and track my contribution progress,
So that: I feel motivated to participate and can see recognition for my efforts.

Acceptance Criteria:

1. **Points for Contribution:**
   - **Given** I submit a suggestion, comment, or report an issue on a term,
   - **Then** I earn points based on the type and quality of the contribution.
2. **Achievement Unlocking:**
   - **Given** I reach a milestone,
   - **Then** I unlock a badge or achievement.
3. **Progress Dashboard:**
   - **Given** I navigate to my profile,
   - **Then** I can view my total points, badges earned, and contribution rank.

4. **Real-Time Feedback:**
    - **Given** I perform a gamified action,
    - **Then** I receive instant feedback such as a pop-up message.
5. **Leveling Up:**
    - **Given** I reach predefined thresholds,
    - **Then** my user rank or title updates accordingly.
6. **Offline Support:**
    - **Given** I contribute while offline,
    - **Then** my contributions and points are synced once I reconnect.

2.6. User Story #6: UpVote System

ID: US006 (Mavito Project)
Title: Crowdsourced Validation via UpVoting
As a: Mavito application user (e.g., casual user, linguist, or academic) who wants to contribute to the accuracy and quality of lexicon entries,
I want: to upvote (or downvote) suggested changes or comments on terms in the lexicon,
So that: the community can collectively validate contributions, and moderators can prioritize high-quality updates for integration into the central repository.

Acceptance Criteria:

1. **Voting Interface:**
    - **Given** I am viewing a term entry with user-submitted comments or suggested changes,
    - **Then** I see an option to upvote or downvote each contribution.
    - **And** the current vote count is displayed next to each contribution.
2. **Vote Submission:**
    - **Given** I am logged in and have not yet voted on a specific contribution,
    - **When** I click the upvote/downvote button,
    - **Then** my vote is recorded immediately (if online) or queued for sync (if offline).
    - **And** the UI reflects my vote and updates the vote count.
3. **Prevent Duplicate Voting:**
    - **Given** I have already voted on a contribution,
    - **Then** the UI prevents me from voting again (unless I undo my vote).
4. **Offline Handling:**
    - **Given** I vote while offline,

- **Then** the vote is stored locally and synced to the central repository when I reconnect.

5. **Moderation Visibility (Optional Enhancement):**
   - **Given** I am a moderator,
   - **Then** I can filter contributions by vote count to prioritize high-quality submissions.
6. **Feedback Transparency:**
   - **Given** a contribution receives significant downvotes,
   - **Then** the system may flag it for review (future enhancement).

Notes/Assumptions:

- Voting requires user authentication to prevent abuse.
- Vote counts are public to encourage transparency.
- Offline votes are treated as "pending" until synced.
- Future enhancements could include:
  - Weighted voting for trusted users (e.g., linguists).

2.7. User Story #7.1: Word Frequency Trends

ID: US007.1 (Mavito Project)
Title: Historical Word Usage Visualization
As a: Linguist or language researcher studying lexical evolution,
I want: To view historical trends of word usage frequency across different time periods,
So that: I can identify patterns in language adoption, obsolescence, or cultural influences.

Acceptance Criteria:

1. **Trend Visualization:**
   - **Given** I select a word or phrase,
   - **Then** I see a line chart showing its monthly/quarterly/yearly frequency.
   - **And** can toggle between absolute counts and percentage changes.
2. **Comparative Analysis:**
   - **Given** I select multiple words,
   - **Then** the system overlays their trends with distinct colors.
   - **And** provides a legend identifying each word.
3. **Contextual Data:**

- **Given** I hover over a data point,
- **Then** I see exact usage counts and sample sentences.
- **And** can click to view source documents.

4. **Export Functionality:**
   - **Given** I want to analyze data externally,
   - **Then** I can export charts as PNG or data as CSV/JSON.

Notes/Assumptions:

- Data aggregates nightly for performance.
- Supports all 12 official South African languages.
- Default view shows last 12 months.
- Future enhancements could include:
  - Regional usage heatmaps.
  - Sociolinguistic correlation analysis.

2.8. User Story #7.2: Contribution Analytics

ID: US007.2 (Mavito Project)
Title: User Contribution Visualization
As a: Regular contributor to the Mavito platform,
I want: To see a breakdown of my edits and comments across languages,
So that: I can track my impact and focus on underrepresented languages.

Acceptance Criteria:

1. **Personal Dashboard:**
   - **Given** I view my profile,
   - **Then** I see a pie chart of my contributions by language.
   - **And** a timeline of my activity.
2. **Progress Metrics:**
   - **Given** I'm an active user,
   - **Then** I see my percentile ranking in the community.
   - **And** suggested languages needing more contributions.

Notes/Assumptions:

- Updates within 5 minutes of contributions.
- Includes all contribution types.
- Future enhancements could include:
  - Team contribution tracking.
  - Contribution quality scoring.

2.9. User Story #7.3: Trending Terms

ID: US007.3 (Mavito Project)
Title: Real-Time Popularity Tracking
As a: Language learner or cultural researcher,
I want: To see which words are currently trending in popularity,
So that: I can stay current with evolving language usage.

Acceptance Criteria:

1. **Trending Display:**
    - **Given** I visit the homepage,
    - **Then** I see a "Trending Now" carousel with top terms.
    - **And** percentage change indicators.
2. **Contextual Information:**
    - **Given** I select a trending word,
    - **Then** I see related news/events driving popularity.
    - **And** its historical frequency graph.

Notes/Assumptions:

- Updates every 4 hours.
- Excludes spam/fake trends.
- Future enhancements could include:
    - User-submitted trend explanations.
    - Regional trend variations.

Assumptions:

- Contributions are validated before points are awarded to prevent abuse.
- Gamification rewards are symbolic, not monetary.
- This feature is designed to encourage consistent, high-quality engagement.

2.10. User Story #8: Responsive Design

ID: US008 (Mavito Project)
Title: Responsive Layout for Mobile and Desktop Devices
As a: user who may need to access the Mavito application while on the go,
I want: the user interface to adapt and remain fully functional on mobile phones, tablets, and desktops,
So that: I can quickly and seamlessly use the application in any situation, regardless of the device I'm using.

Acceptance Criteria:

1. **Mobile Adaptability:**
    - **Given** I am using the Mavito application,
    - **And** I am on a mobile device,
    - **Then** the layout should automatically adjust to fit the screen without horizontal scrolling.
2. **Touch-Friendly UI:**
    - **Given** I am interacting with the app on a touchscreen device,
    - **And** I tap on any button or link,
    - **Then** the tapped element should be responsive to touch and have a minimum tap area of 48px $\times$ 48px.
3. **Touchscreen Swiping and Scrolling Support:**
    - **Given** I am using the Mavito app on a touchscreen device,
    - **And** I am viewing content that extends beyond the initial screen (e.g., a list or feed),
    - **Then** I should be able to scroll vertically or horizontally (depending on the content) using swipe gestures,
    - **And** the scrolling should be smooth and responsive.
4. **Adaptive Mobile Navigation Elements:**
    - **Given** I am using the Mavito app on a mobile device,
    - **Then** the navigation should adapt to a mobile-friendly layout, such as displaying a hamburger menu instead of a full-width navigation bar.
5. **Device Orientation Handling:**
    - **Given** I am using the app on a mobile device,
    - **And** I switch between portrait and landscape mode,
    - **Then** the UI layout should follow without breaking or cutting off content.
6. **Mobile Hardware and OS Compatibility:**
    - **Given** I am using the Mavito app on a mobile device running a supported OS (e.g., Android 10+, iOS 13+),
    - **And** the device has at least 2 GB of RAM and a modern mobile browser (e.g., Chrome),
    - **Then** all core features should work smoothly without crashes or performance lags.
7. **Error Handling - Network Errors (Mobile Use):**
    - **Given** I lose internet connectivity due to a poor signal while using the app,

- **Then** the application should seamlessly switch to its offline version without interruption,
- **And** provide an indication informing me that the app is currently in offline mode.

Notes/Assumptions:

- Devices with screen widths less than or equal to 768 pixels are considered mobile, and touch interaction is expected on these devices.

2.11. User Story #9: Dictionary Glossary

ID: US009 (Mavito Project)
Title: Glossaries User Stories
As a: terminology researcher
I want: to view detailed information about a specific term,
So that: I can understand its meaning and translations.

Acceptance Criteria:

1. **Definition Display:**
   - **Given** I have selected a term from search results,
   - **And** I should see its complete definition,
   - **Then** its assigned category with clickable link to the full glossary.
2. **Translation Panel**
   - **Given** I am viewing a term,
   - **And** all available translations should be displayed in a structured table,
   - **Then** clear language code labels (e.g., ”afr”, ”zul”).
3. **Missing Data Handling:**
   - **Given** a term lacks translation for certain languages,
   - **Then** those fields should display ”Translation not available”,
   - **And** be visually distinct from complete entries.

Notes/Assumptions:

- Integrates with US003 search results.
- Preserves all existing search filters when navigating from results to term view

2.12. User Story #10: Glossary Category Navigation

ID: US010 (Mavito Project)
Title: Glossary Category Navigation
As a: user interested in exploring specific domains (e.g., Agriculture, Legal),
I want: to browse all terms within a particular category,
So that: I can discover related terminology.

Acceptance Criteria:

1. **Category Selection:**
   - **Given** I access the glossary browser,
   - **Then** I should see all available categories.
2. **Term Listing:**
   - **Given** I select a category (e.g., "Agriculture"),
   - **Then** I should see paginated results of all terms,
   - **With** options to filter by language availability.
3. **Cross-Referencing:**
   - **Given** I view a term in a glossary,
   - **Then** I should see related terms from the same category,
   - **And** options to navigate to similar categories.

Notes/Assumptions:

- Categories are predefined based on dataset taxonomy
- Works both online and for downloaded glossaries

2.13. User Story #11: Term Bank Translations

ID: US011 (Mavito Project)
Title: Access Multilingual Translations
As a: multilingual user,
I want: to toggle between translations for a term,
So that: I can understand it in my preferred language.

Acceptance Criteria:

1. **Translation Toggle:**
   - **Given** I view a term,
   - **When** I click a language tab (e.g., "Zulu"),
   - **Then** I should see the translation ("Imbewu kawoyela").
2. **Missing Translations:**

- **Given** a term has no translation for a language,
- **Then** display "Translation not available.

Notes/Assumptions:

- Uses the translations field from the dataset.

3. Use Case Diagrams

This section presents the key use case diagrams developed for the Mavito application. Each diagram visualizes the interactions between user roles and the system, based on the user stories defined earlier in this document. These diagrams serve as a visual blueprint for understanding the system's expected behavior.

3.1. Registration and Login Use Case Diagram

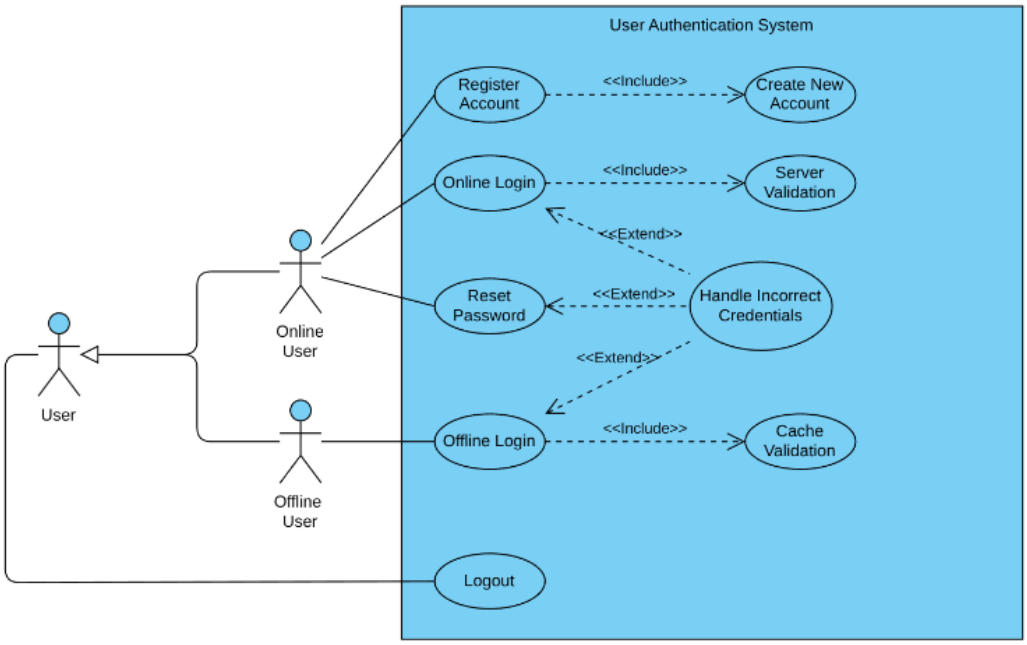

Figure 1: Registration and Login Use Case Diagram

3.2. Interface Language Switching Use Case Diagram

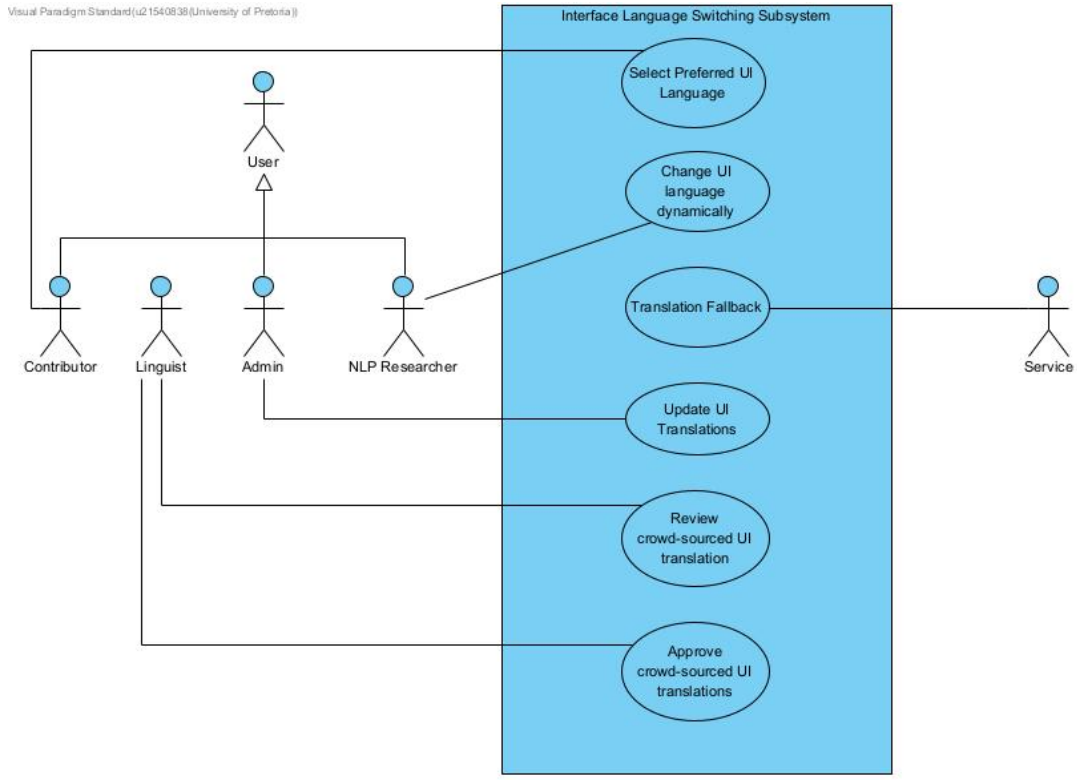

Figure 2: Interface Language Switching Use Case Diagram for Mavito

3.3. Search Term Use Case Diagram

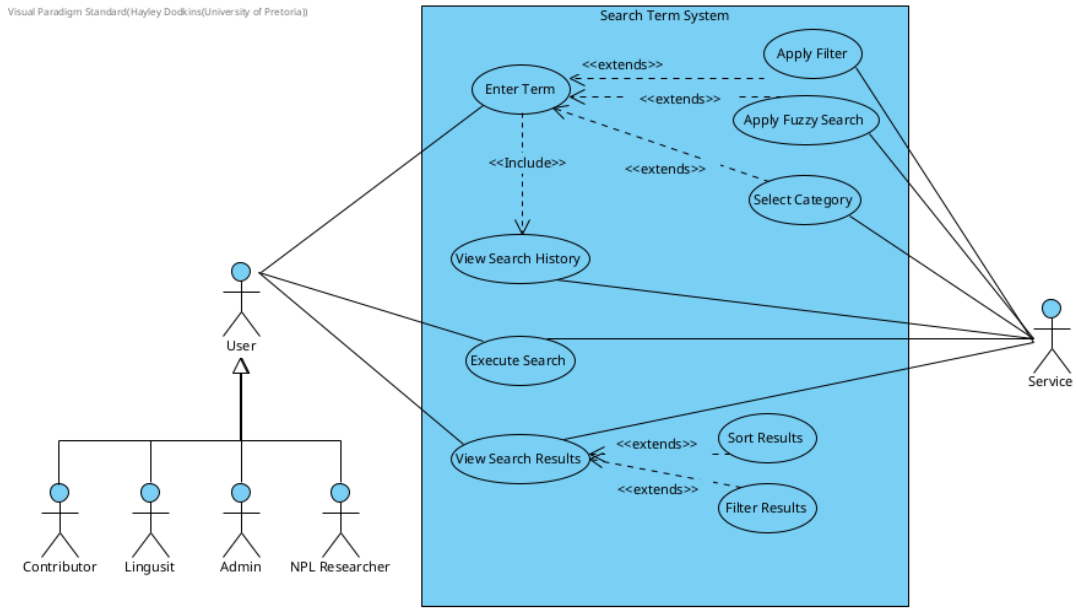

Figure 3: Search Term Use Case Diagram for Mavito

3.4. Gamification Use Case Diagram

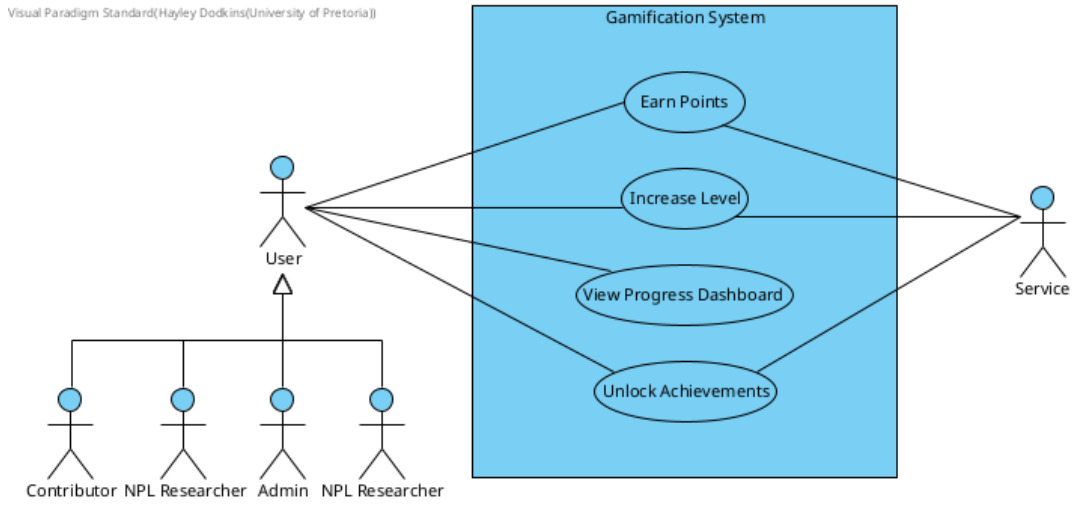

Figure 4: Gamification Use Case Diagram

3.5. Visualization Use Case Diagram

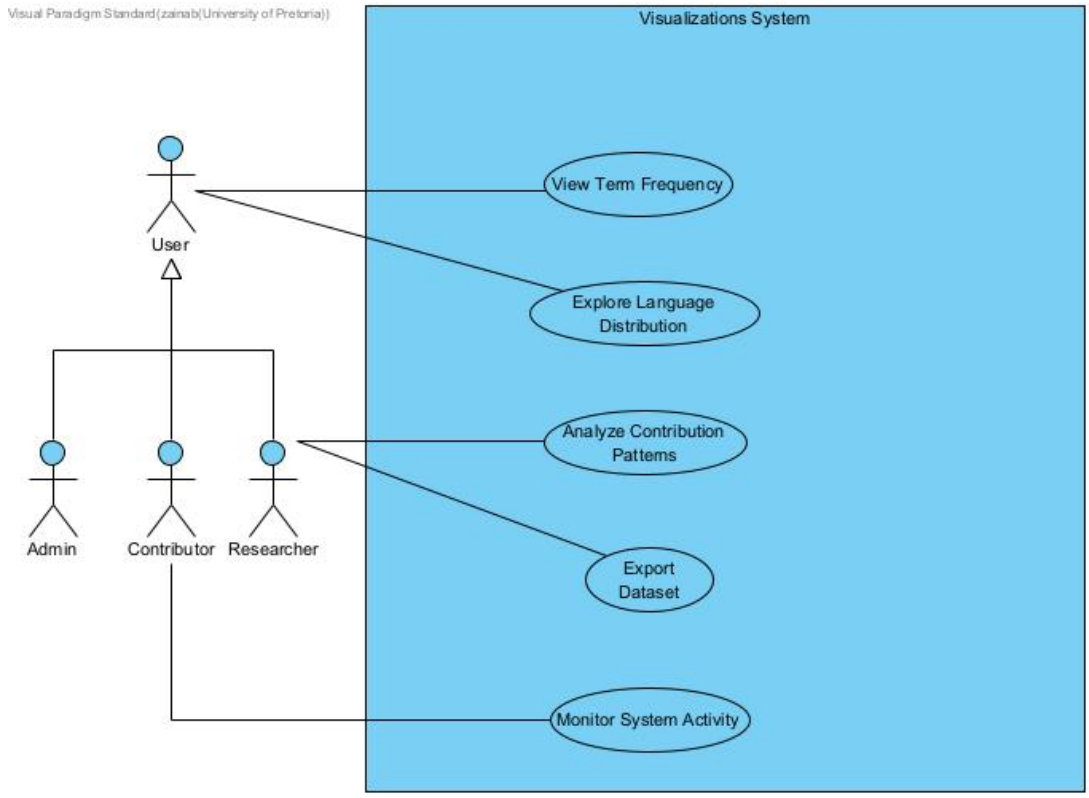

Figure 5: Visualization Use Case Diagram

3.6. Themes Use Case Diagram

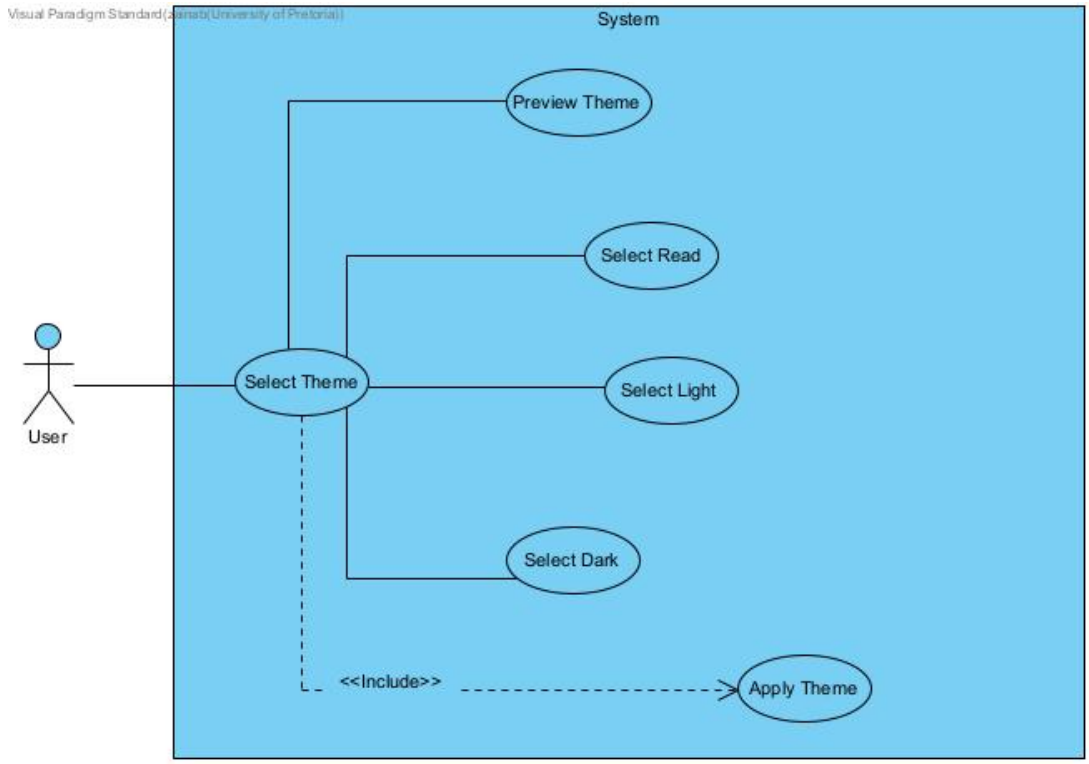

Figure 6: Themes Use Case Diagram

3.7. Contributions Use Case Diagram

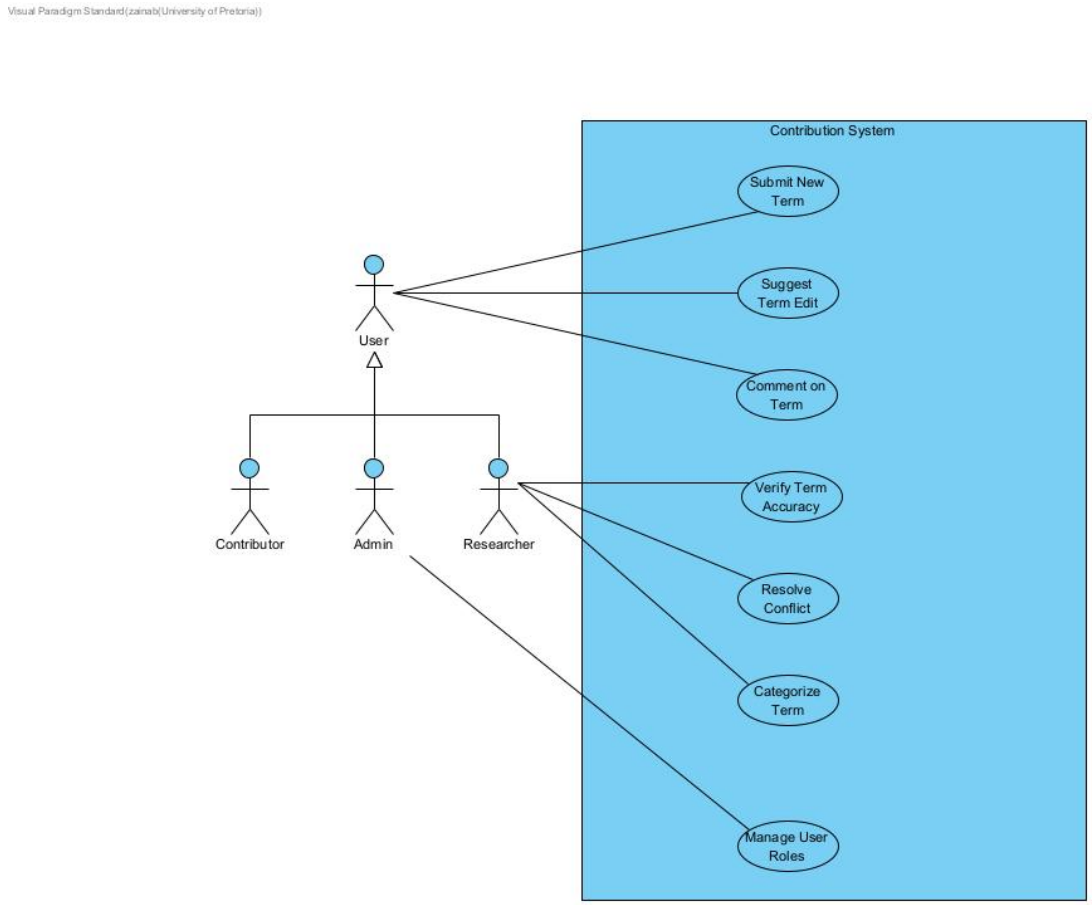

Figure 7: Contributions Use Case Diagram

3.8. Dictionary Use Case Diagram

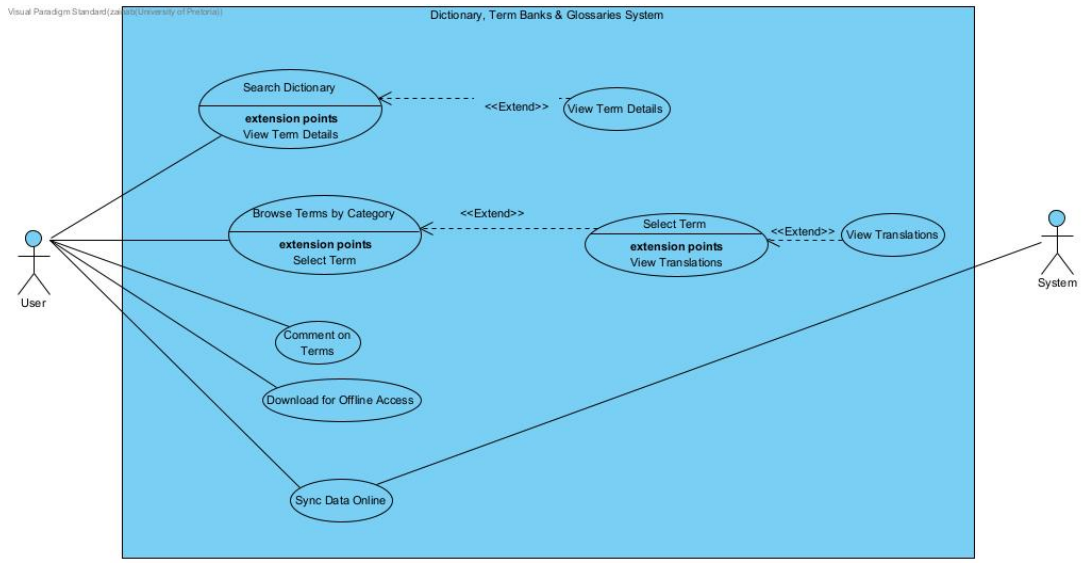

Figure 8: Contributions Use Case Diagram

3.9. Admin Use Case Diagram

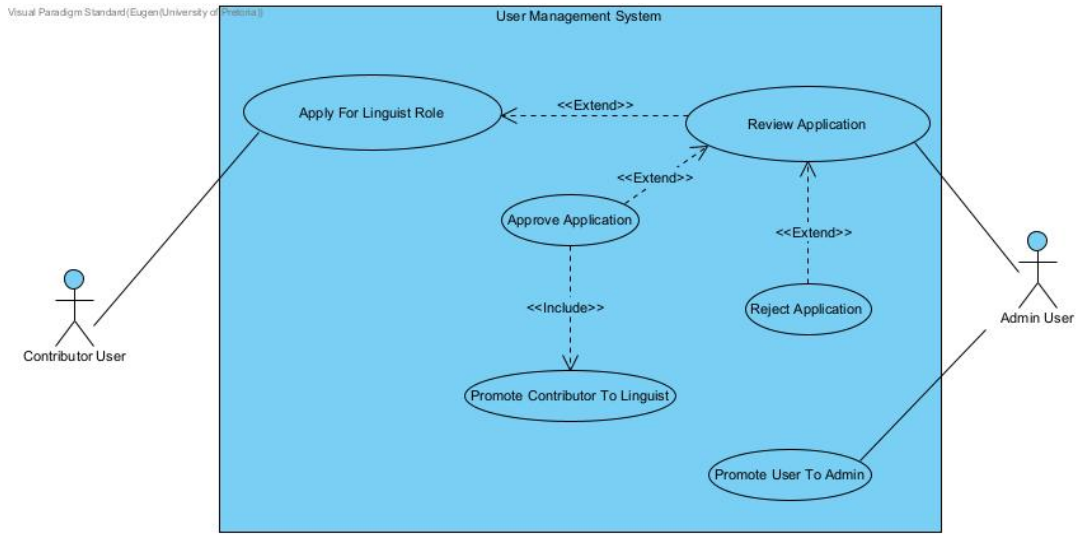

Figure 9: Contributions Use Case Diagram

3.10. Export Use Case Diagram

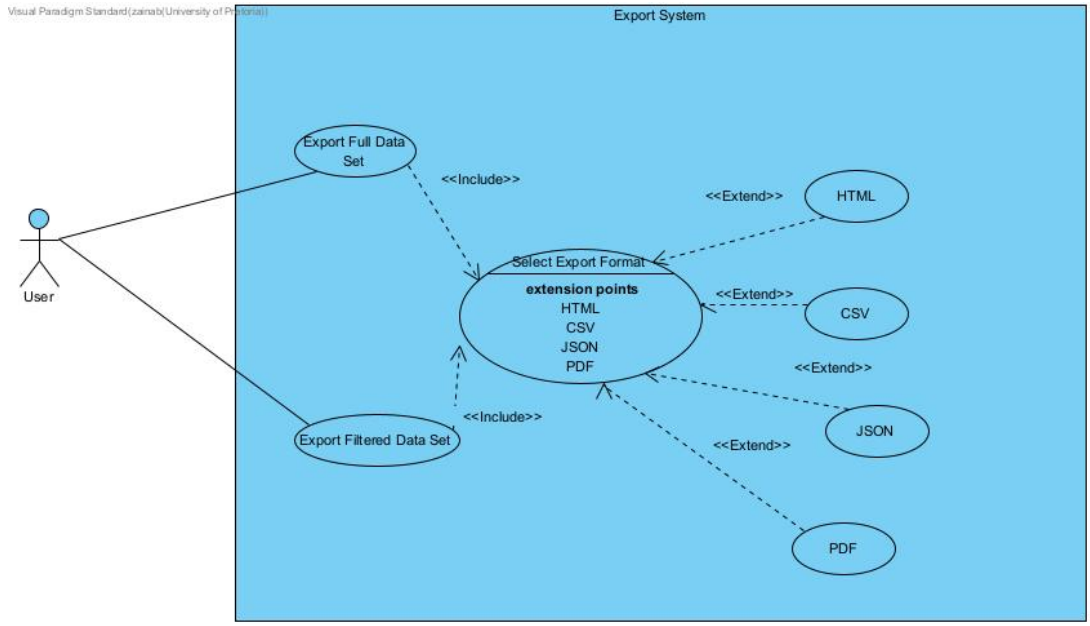

Figure 10: Contributions Use Case Diagram

3.11. Overview

The main use cases covered include:

- Browsing and searching multilingual terms
- Interface language switching
- Earning contribution points through gamified interactions

- Registration and login interactions
- User Contributions
- User themes
- Data visualizations
- Dictionary Use Case
- Admin Use case
- Export Use Case

4. Functional Requirements

FR1: **Glossary Browsing and Search**

- FR1.1: The system shall allow users to browse available glossaries and term banks.
- FR1.2: The system shall provide a unified search interface across all multilingual glossaries.
- FR1.3: The system shall allow users to search for terms using exact matches, partial strings, or semantic similarity.
- FR1.4: The search results shall be ranked by relevance and display key metadata (definition, part of speech, language).
- FR1.5: The system shall highlight related terms and translations in the term view.

FR2: **Multilingual Support**

- FR2.1: The system shall support interface localization in multiple South African languages.
- FR2.2: The system shall allow users to switch the UI language at any time.
- FR2.3: The system shall correctly render special characters where applicable.

FR3: **User Contributions and Feedback**

- FR3.1: The system shall allow users to submit comments.
- FR3.2: The system shall allow users to submit corrections or suggestions for terms.
- FR3.3: The system shall send submitted feedback to a backend repository for moderation.
- FR3.4: The system shall support version control for submitted glossary data and prevent overwriting of validated entries.
- FR3.5: The system should be able to allow users to upvote/downvote term changes.

FR4: **AI-Enhanced Functionality**

- FR4.1: The system shall support AI-powered semantic search to retrieve conceptually related terms.
- FR4.2: The system may provide suggested definitions or translations using AI models.
- FR4.3: The system may automatically cluster terms based on meaning or domain.
- FR4.4: The system may auto-tag glossary entries with linguistic metadata (e.g., part-of-speech).
- FR4.5: The system may integrate with external NLP APIs for research purposes.

FR5: **Gamification**

- FR5.1: The system shall assign points to users when they submit valid suggestions, comments, or issue reports on glossary terms.
- FR5.2: The system shall track milestones based on user activity and unlock achievements or badges when predefined thresholds are reached.
- FR5.3: The system shall display a progress dashboard on the user's profile page, including total points, badges earned, and contribution rank.
- FR5.4: The system shall provide real-time feedback when users perform actions that affect their gamification status.
- FR5.5: The system shall support user rank or title progression as users accumulate points and reach specific contribution levels.
- FR5.6: The system shall queue gamified contributions made offline and synchronize them with the server once the user is reconnected.
- FR5.7: The system shall validate contributions before assigning gamification points to prevent abuse.

FR6: **Progressive Web Application Functionality**

- FR6.1: The system shall function as a PWA and be installable on mobile and desktop devices.
- FR6.2: The frontend shall use service workers to cache glossary data, enabling offline access to core features such as term lookup and browsing.
- FR6.3: The system shall store user feedback or contributions made offline and queue them for submission once connectivity is restored.
- FR6.4: The application shall automatically synchronize cached content with the server when an internet connection is detected.
- FR6.5: The system shall provide visual feedback or indicators when the app is operating in offline mode.
- FR6.6: The system shall support background updates of cached glossary data to maintain consistency with the server.
- FR6.8: The system shall allow users to download selected glossaries for offline access.

FR7: **Data Visualization and Analytics**

- FR7.1: The system should be able to display stats on word frequency and usage trends.
- FR7.2: The system should be able to visualize a user's contribution to different languages.
- FR7.3: The system should highlight trending words and new entries to the database.
- FR7.4: The system should be able to provide interactive charts or graphs for linguistic data.

FR8: **Data Import and Export**

- FR8.1: The system shall allow users to export glossary data in JSON format.
- FR8.2: The system shall allow users to export glossary data in CSV format.
- FR8.3: The system may allow authorized users to import glossaries in JSON or CSV format.

5. Non-Functional Requirements

NFR1: **Offline Accessibility**

- NFR1.1 The system shall support offline access to previously downloaded language resources.
- NFR1.2 The frontend shall use service workers and caching to allow uninterrupted use of core features without an active internet connection.
- NFR1.3 Synchronization of updated resources shall automatically occur once the device is back online.

NFR2: **Performance and Responsiveness**

- NFR2.1 The system shall deliver fast search responses, with results displayed within 2 seconds for standard queries.
- NFR2.2 The frontend application shall load and become interactive within 3 seconds on devices with moderate hardware and average bandwidth.
- NFR2.3 UI interactions shall be smooth and not exceed 100ms latency where possible.

NFR3: **Scalability**

- NFR3.1 The backend shall be able to handle simultaneous requests from a growing user base, including researchers and contributors.
- NFR3.2 The data architecture must accommodate the addition of new glossaries, languages, and APIs without the need for major refactoring.

NFR4: **Security and Privacy**

- NFR4.1 All data transmissions between client and server shall be encrypted using HTTPS.
- NFR4.2 User-submitted feedback shall be sanitized and validated to prevent injection attacks.
- NFR4.3 The backend shall implement basic access control to restrict sensitive actions to authorized roles.

NFR5: **Usability and Accessibility**

- NFR5.1 The user interface shall support all 11 official South African languages through dynamic localization.
- NFR5.2 The application shall conform to WCAG 2.1 Level AA accessibility guidelines to accommodate visually impaired users.
- NFR5.3 The system shall provide a clean and intuitive interface that requires no more than three clicks to reach key features.

NFR6: **Maintainability**

- NFR6.1 The codebase shall follow modular and well-documented design practices to enable ease of maintenance.
- NFR6.2 The frontend and backend shall use consistent code formatting enforced via linting tools.
- NFR6.3 New contributors shall be able to understand and modify the codebase with minimal onboarding effort.

NFR7: **Extensibility**

- NFR7.1 The system shall support the addition of new term banks and glossaries via a modular import system.
- NFR7.2 New frontend features shall be integrable without affecting the core application features.
- NFR7.3 The backend shall expose extensible REST API routes following OpenAPI standards.

NFR8: **Reliability and Fault Tolerance**

- NFR8.1 The system shall gracefully handle failed API requests and notify the user when a feature is temporarily unavailable.
- NFR8.2 The frontend shall include fallback mechanisms for critical resources, ensuring minimal disruption in case of partial data loss.
- NFR8.3 The backend shall log all failed operations and expose logs for future debugging or auditing.

NFR9: **Portability and Cross-Platform Compatibility**

- NFR9.1 The PWA shall work on major browsers (Chrome, Firefox, Edge, Safari) and platforms (Windows, Android, iOS, Linux).

- NFR9.2 The UI shall be fully responsive and usable on screen sizes ranging from smartphones to desktop monitors.
- NFR9.3 No feature shall be dependent on platform-specific behavior.

NFR10: **Deployment and DevOps Readiness**

- NFR10.1 The backend shall be containerized using Docker to support consistent deployment across environments.
- NFR10.2 CI/CD pipelines shall be configured to run tests, build artifacts, and deploy the application to the cloud or GitHub Pages.

6. Constraints

6.1. Overview

This section outlines the constraints that govern the design, development, deployment, and maintenance of the **Mavito** application. These constraints stem from ethical considerations, client requirements, project context, budgetary limitations, and architectural guidelines. Adherence to these constraints is critical to the project's success and sustainability.

6.2. Constraints

2.1 **Privacy and Data Minimization**

- The application must adopt a privacy-first approach.
- No personal user information may be collected unless voluntarily submitted.
- No personal data should be stored without explicit user consent.
- Any user data collected must comply with ethical research standards and university data policies.
- All stored data must follow ethical and data protection principles.

2.2 **Maintainability and Sustainability**

- The project must use a maintainable and reliable technology stack that future students or stakeholders can easily understand and extend.
- All source code must be well-documented, modular, and follow clean architecture principles.
- Technologies selected must have active community support and comprehensive documentation to minimize onboarding effort.

2.3 **Use of Open Source Technologies**

- Wherever possible, the project must use open-source frameworks and libraries.
- External packages or modules must have permissive licenses (e.g., MIT, Apache 2.0) and be cited appropriately.

2.4 **Budget Constraints**

- The project must be developed with minimal to zero cost.
- The team is not permitted to incur costs unless explicitly approved by the client.
- Hosting should utilize free tiers.

2.5 Offline Functionality

- The application must support offline access as a core feature.
- Core features must function without an active internet connection, using Progressive Web App (PWA) technologies such as service workers and local caching.
- Synchronization with the central repository must occur seamlessly once connectivity is restored.

2.6 Version Control of Glossary Data

- Glossary data and language resources must be version-controlled to prevent loss or corruption of validated content.
- User feedback must be stored separately and reviewed before merging into the main dataset.
- No glossary data may be overwritten without proper validation and versioning mechanisms.

2.7 Ethical and Legal Considerations

- No proprietary or restricted datasets may be used without explicit authorization.
- Contributions and authorship must be tracked where possible to preserve academic and community recognition.

2.8 Performance and Efficiency

- The application must be responsive and performant on low-resource or mobile devices.
- Heavy assets (e.g., large datasets or images) must be lazy-loaded or paginated.
- Search and filtering mechanisms must be optimized for performance on slower devices and large datasets.

2.9 Accessibility Requirements

- The application must adhere to WCAG 2.1 AA accessibility standards.
- Features such as text-to-speech, keyboard navigation, color contrast, and screen reader support should be considered where feasible.
- Dark mode and adjustable text size are encouraged to support visual diversity in users.

2.10 Deployment and Portability

- The system must be deployable using containerization to support ease of testing and reproducibility.

- CI/CD pipelines must be used for automated builds and testing.
- Both frontend and backend should be deployable without manual intervention.

2.11 Multi-Language Support

- The interface must support multiple South African languages.
- The system must be internationalization-ready using industry-standard practices (e.g., translation JSON files, locale selectors).
- UI elements, glossaries, and feedback must render correctly regardless of language.

6.3. Summary

These constraints serve as the foundation for all architectural, design, and implementation decisions made during the development of **Mavito**. They ensure the solution is ethical, sustainable, inclusive, affordable, and aligned with the client's goals of making multilingual resources accessible to all.

7. Domain Model

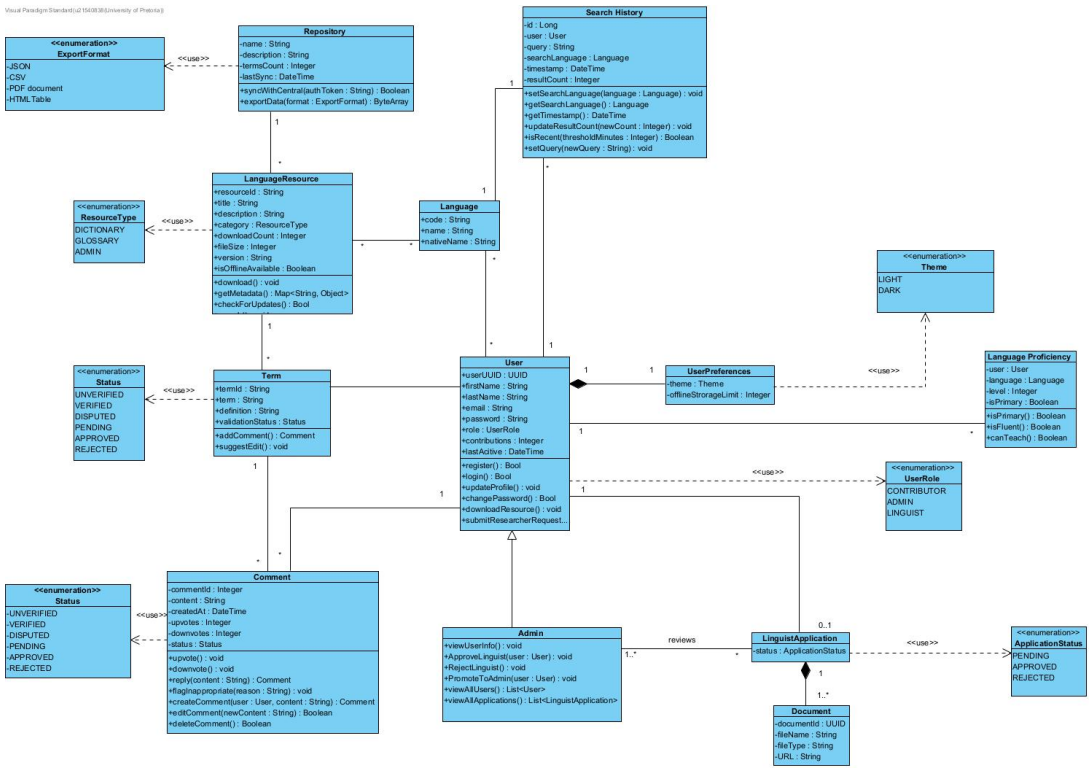

Figure 11: Domain Model

8. Architecture Patterns

The architectural foundation of Mavito combines the Microkernel and Microservices patterns to address both extensibility and service independence. This hybrid approach is

essential for building a maintainable, scalable, and community-driven multilingual terminology platform. Each pattern plays a distinct role in supporting the system's goals, and their selection is justified below.

8.1. Microkernel Pattern

The Microkernel pattern is employed in the frontend and core backend logic to support feature modularity. The Mavito PWA core includes only essential functionality: multilingual glossary viewing, search, and feedback submission. Additional services such as AI-enhanced semantic search, gamification metrics, or user-uploaded terms implemented as optional plugins that extend the core without modifying it.
Motivation and Usage:

- Allows teams to add or disable features without affecting the base system.
- Enables experimentation with novel features like gamified contribution scoring or data validation without bloating the core.
- No personal data should be stored without explicit user consent.
- Ideal for open-source collaboration where contributors might develop new plugins independently.

8.2. Microservices Pattern

On the backend, the system uses a Microservices architecture. Each core function is encapsulated within an independently deployable service. These services communicate via well-defined APIs, often exposed through REST endpoints.
Motivation and Usage:

- Services like term search, user contribution tracking, and feedback moderation are implemented and deployed independently.
- Facilitates horizontal scaling where search and analytics services can scale independently of the glossary browser.
- Increases fault isolation as a failure in one service does not compromise the rest of the application.
- Supports technology diversity, enabling future services to be written in different languages.

8.3. Benefits of the Combined Approach

- **Extensibility:** New features can be added as plugins without disrupting the rest of the system.
- **Separation of Concerns:** Responsibilities are clearly delineated between base functionality and modular enhancements.
- **Deployment Flexibility:** Dockerized services and a plugin-based UI allow different deployments to include only the relevant modules.

This combination ensures that Mavito remains agile, extensible, and robust, with clearly isolated responsibilities across its components. These are key traits for a system designed to support a growing multilingual terminology community.

9. Design Patterns

The Mavito system employs several proven design patterns to promote modularity, clarity, and maintainability. These patterns are used across both frontend and backend components to ensure clean separation of concerns, extensibility, and testability.

9.1. Backend Design Patterns

- **Repository Pattern**: This pattern abstracts the data access layer, decoupling the application's business logic from its persistence mechanisms. In Mavito, this is used to isolate FastAPI routes from direct interaction with JSON files or potential databases. This makes it easier to substitute or refactor storage formats later.
- **Service Layer Pattern**: This pattern encapsulates business logic in service classes rather than embedding it in route handlers. For example, the glossary service processes user input, validates term entries, and formats responses. This keeps the FastAPI route functions minimal and focused on HTTP handling.
- **Proxy Pattern**: Used to control access to sensitive glossary data, this pattern allows features like lazy-loading or permission-based access control. For instance, the proxy layer can prevent unreviewed user-uploaded data from being merged directly into the main glossary.
- **Adapter Pattern**: This is used to integrate external APIs, particularly AI/NLP tools for semantic search or text analytics into the system without modifying the core glossary services. Adapters provide a uniform interface so different NLP services can be swapped in or out easily.
- **Builder Pattern**: Glossary term responses often contain complex nested structures (e.g., term, translations, part of speech, examples, contributor metadata). The builder pattern helps assemble these objects step-by-step, ensuring clarity and consistency in API responses.

9.2. Frontend Design Patterns

- **Observer Pattern**: React's reactive rendering model naturally supports this pattern. In Mavito, UI components subscribe to state changes such as term updates, language switching, or sync status. This enables real-time UI updates without manual DOM manipulation.
- **Model–View–Controller (MVC)**: Mavito uses React components as Views, application state as the Model, and event handlers or hooks as Controllers. This separation ensures testable, reusable components and a clear flow of data and control.

- **Model–View–ViewModel (MVVM)**: For more complex UI flows, especially those with async behavior or derived state (e.g., loading indicators, sync status), Mavito uses custom React hooks as ViewModels to isolate logic from presentation. This allows clean and declarative UIs while keeping logic reusable across components.

Together, these patterns ensure that both backend services and frontend components remain modular, understandable, and adaptable to future requirements. They also enable team members to work in parallel with minimal friction.

10. Technology Requirements

The Mavito system is built using modern, open-source technologies selected for their performance, maintainability, and compatibility with offline-first web applications and modular architectures. These technologies are categorized below according to their role in the stack.

10.1. Frontend

- **Framework:** React (via Vite) for fast, modular UI development.
- **Language:** TypeScript for type safety and developer productivity.
- **Styling:** SCSS modules and Tailwind CSS for flexible and responsive design.
- **Build Tool:** Vite for fast local development and optimized production builds.
- **PWA Features:** Service workers and localStorage/IndexedDB for offline capabilities.
- **Testing:** Vitest for unit testing; Cypress for end-to-end testing.
- **Linting and Formatting:** ESLint and Prettier for code consistency.
- **Pre-commit Hooks:** Husky for running ESLint and Prettier on staged files, and for enforcing commit message conventions.

10.2. Backend

- **Framework:** FastAPI for high-performance, Pythonic web APIs.
- **Language:** Python 3.11 with Pydantic for validation and typing.
- **Service Model:** Microservices organized by function.
- **Database:** PostgreSQL.
- **API Documentation:** FastAPI automatically generates OpenAPI-compliant service contracts, which are accessible via a ReDoc interface. This provides structured, human-readable API documentation derived directly from the backend codebase and Pydantic models.
- **Linting:** Ruff for fast, extensible linting.

- **Static Type Checking:** Mypy to ensure type correctness.
- **Formatting:** Black for consistent code formatting.
- **Testing:** Pytest for integration and unit tests.
- **Cloud Platform:** Google cloud for hosting.

10.3. DevOps and Deployment

- **Containerization:** Docker for consistent development and deployment environments.
- **Orchestration:** Docker Compose for local service management.
- **CI/CD:** GitHub Actions for automated linting, testing, and deployment.
- **Hosting:** GitHub Pages.

11. Technology Choices

Our system is built using a modular microservices architecture, designed for scalability, isolation, and maintainability. We evaluated at least three technology options for each core component before selecting the final stack.

Frontend Framework

Option	Overview	Pros	Cons
React (Chosen)	Declarative JavaScript library for UIs	Large ecosystem, reusable components, PWA support	Requires setup, JSX learning curve
Angular	Full-featured frontend framework	Strong typing, built-in tools, opinionated structure	Heavy, complex, steep learning curve
Vue.js	Progressive framework	Lightweight, easy to learn, good docs	Smaller ecosystem for enterprise-scale systems

Justification: React provides modularity and performance while supporting Progressive Web App (PWA) functionality which is ideal for our offline-first goal. Its community and integration with modern tooling enhance maintainability.

Backend Framework (API Layer)

Option	Overview	Pros	Cons
FastAPI (Chosen)	Async Python framework for APIs	Very fast, OpenAPI docs, type hinting, async support	Newer, less mature than Django
Flask	Minimal WSGI Python web framework	Simple, flexible, lightweight	No async support, more boilerplate
Django	Full-stack Python web framework	ORM, admin panel, built-in features	Monolithic, less suitable for microservices

Justification: FastAPI is a performant async framework with native support for OpenAPI docs and type hinting. It aligns well with our modular design and Python-based data processing.

Backend Language

Option	Overview	Pros	Cons
Python (Chosen)	General-purpose scripting language	Rich ML/NLP libraries, fast prototyping	Slower runtime compared to compiled languages
Go	Compiled systems language	High performance, excellent concurrency support	Verbose, smaller ecosystem for data science
Node.js	JS runtime for backend	Unified stack, async by default	Callback complexity, fragmented tooling

Justification: Python enables rapid development and integrates well with NLP tools like spaCy. Since our AI search requires natural language features, Python is the most suitable choice.

Containerization

Option	Overview	Pros	Cons
Docker (Chosen)	OS-level virtualization platform	Portable, consistent dev/production environments	Requires setup and Docker knowledge
Vagrant	VM-based development tool	OS-level control	Heavyweight, slower than containers
Bare Metal	Manual installs	Full flexibility	Not reproducible, error-prone

Justification: Docker enables reproducible builds, isolates dependencies, and simplifies onboarding. It is essential for deploying and maintaining microservices efficiently.

Summary

Our technology stack: React, FastAPI with Python, and Docker was chosen for its strong modularity, ease of maintenance, and developer productivity. React enables the creation of responsive, user-friendly interfaces that function seamlessly across all device types. FastAPI offers fast development with built-in validation and documentation, making it

ideal for building clean and maintainable APIs. Docker ensures consistent environments across development and production, supporting a scalable microservices architecture. These choices align directly with our goals of building an offline-capable, user-centered platform that is simple to deploy, easy to extend, and reliable across a wide range of usage contexts.

12. Service Contracts

12.1. API Information

- **OpenAPI Version:** 3.1.0
- **Title:** Marito API Default
- **Version:** 0.1.0

12.2. Endpoints Overview

Endpoint	Description
`/api/v1/auth/register` (POST)	Register new user. Expects first_name, last_name, email, password, and optionally role. Returns user object on success.
`/api/v1/auth/login` (POST)	OAuth2-compatible login. Frontend sends email (username) and password. Returns access token for future requests.
`/api/v1/auth/me` (GET)	Get details of the current logged-in user. Requires OAuth2 bearer token.
`/api/v1/analytics/descriptive` (GET)	Retrieve descriptive analytics summary.
`/api/v1/search/` (GET)	Search endpoint for multilingual terms with optional filters: language, domain, part of speech, sorting, pagination. Returns list of terms and total count.
`/api/v1/suggest/` (GET)	Autocomplete suggestions based on partial search term. Returns up to 10 suggestion objects.
`/` (GET)	Root endpoint. Returns a welcome message and API documentation link.

12.3. Schemas Summary

- **User** — Email, first name, last name, role, profile picture, status, timestamps.
- **UserCreate** — Email, first name, last name, password, optional role.
- **Token** — Access token and token type.
- **Suggestion** — ID and label for autocomplete.
- **ValidationError** — Location, message, and error type.
- **UserRole Enum** — `linguist`, `researcher`, `contributor`.

12.4. Security Schemes

- **OAuth2PasswordBearer** — Password flow using `/api/v1/auth/login`.