# Plans + Final

CS50 – edX – HarvardX
Python Translator App

Lee, Victoria (Solo Member)

VictoriaRaven GitHub Here
VictoriaRaven Translator App Here

# Table of Contents

# 1. Project Overview

**Project Purpose:** The purpose of this project is to design and develop a Python-based Translator & Dictionary Application that integrates both offline dictionary lookup and optional online translation, supported by a graphical user interface (Tkinter) and persistent storage (SQLite). The application functions as a hybrid local/online language tool, providing:

- Offline dictionary translation using text-file dictionaries
- Online translation fallback using the MyMemory API
- Robust search, filtering, and paginated dictionary browsing
- Export tools (PDF, JSON, SQL)
- Email sending system (ONLY OUTLOOK MAIL)
- A complete UI with translation history
    - User can access all information through README.md file!

In this Full Documentation & Code section, the **Translator Application** underwent extensive refactoring and restructuring to create a clean, modular, and maintainable architecture. Originally, many components were bundled tightly together.

Through refactoring, the application was redesigned into clear modules such as:

- **translation.py** — local + online translation logic
- **algorithms.py** — search, pagination, helper algorithms
- **spelling.py** — spell-checking and auto-correction
- **validation.py** — input validation and sanitization
- **database.py** — SQLite CRUD operations
- **exports.py** — PDF/JSON export logic
- **emailer.py** — email helper
- **admin_panel.py** — dictionary management
- **gui.py** — Tkinter interface
- **main.py** — single entry point

Each module now has a **well-defined responsibility**, reducing coupling and dramatically improving testability.

All UI logic (Tkinter) is **strictly separated** from translation logic, database operations, and file processing. This allows your GitHub Actions tests to run in **test mode without launching a GUI**, and ensures better performance, debugging, and modular growth.
Inline comments and optional docstrings were added to all primary functions, explaining parameters, behavior, and edge cases. Deprecated code was removed, duplicate logic was consolidated, and naming conventions were standardized.

The deployment process was also formalized:

- A polished **README.md**

- Requirements files for both runtime & dev tools
- Automated **Windows GitHub Actions workflow** to test translations, spelling checks, exports, and PDF generation
- A single entry point (python main.py)
- Complete test suite

The repository also includes a full /docs package with UML diagrams, the CS50 Design Document, the Presentation Slides, and PDF submissions.
This satisfies CS50's documentation requirements.

**Project Deliverables:** The final deliverables will include:

1. Software Deliverables:
   - Fully functional Translator & Dictionary Application (Python + Tkinter)
   - Dictionary module supporting multiple languages (EN ⇄ RU, ES, FR, PO, POL, UKR, IT, DE, etc.)
   - SQLite database storing translation history
   - PDF and JSON export utilities
   - Search, filter, and pagination systems
   - Optional online translation integration
   - Exception handling, threading, and UI responsiveness
2. Documentation Deliverables:
   - UML class diagrams for all modules
   - High-level architecture diagram
   - CS50-style Final Project README
   - User Guide
   - Testing Plan
   - Final Report

Overall, this application serves as an advanced, full-stack Python software project that incorporates data structures, algorithms, GUI design, SQL, API usage, and modular architecture, aligning with CS50 and CMSC 495 software engineering principles.

**NOTE: The Final Demo and Submission Video (per CS50 requirements with version control platform for version tracking, collaboration, and future enhancements) will be shown on the GitHub README.md file. Please refer to there to see the video link!!!**

# 2. Group Organizational Structure

**Overview**

Since this is a solo project, roles are consolidated but still defined according to industry and CMSC 495 expectations. While only one developer is responsible for all tasks, the project maintains a

structured approach by assigning “roles” to conceptual responsibility areas. This mirrors real-world SDLC practices and ensures that all phases of development are completed thoroughly.

**Meetings & Workflow**

Because this is a solo project:

- Work sessions occur daily or every other day, self-scheduled
- Task planning is tracked via GitHub Projects
- Documentation is updated weekly
- Milestones correspond to deadlines
- Tasks and completions all on GitHub Tabs: Milestones/Issues/Pull requests/Projects/Actions/Testing, etc..

| Communication Type | Application | Frequency | Purpose | Owner |
|---|---|---|---|---|
| Development Sessions | IDE (VS Code/PyCharm) | Daily | Implement features, debug, improve modules | Developer |
| Project Review | GitHub / Local Docs | Weekly | Review progress against milestones | Project Manager |
| Issue Resolution | Personal Notes / GitHub Issues | As needed | Log errors, refactor, resolve blockers | Developer |
| Deliverable Submission | UMGC Submission Folder | Weekly | Submit project components | Project Manager |
| Final Demo | UMGC Final Submission | Final Week | Demonstrate all functions | Project Manager / Developer |

**Roles/Responsibilities:**

**Project Lead / Project Manager — Victoria Lee**

- Oversees development progress
- Manages documentation and submission timeline
- Ensures alignment with CMSC 495 requirements
- Maintains communication records and change logs

**Developer — Victoria Lee**

- Implements all modules (dictionary system, GUI, SQLite, API integration)
- Writes algorithms for search, lookup, pagination, and export
- Builds Tkinter UI components
- Integrates database CRUD operations
- Ensures performance, modularity, and maintainability

**Testing Lead — Victoria Lee**

- Creates test scenarios and unit tests
- Conducts manual GUI and functional testing
- Ensures reliability, error handling, and edge-case coverage

**Documentation Lead — Victoria Lee**

- Writes all project documentation (README, SOW, User Guide, etc.)
- Produces diagrams, architecture documents, and design specs
- Prepares final consolidated report

# 3. Statement of Work & Project Management

**Project Statement of Work (SOW)**

This project will produce a fully functional Translator & Dictionary Application that integrates offline dictionaries, online translation API usage, GUI interaction, database storage, and export utilities. The application is intended to demonstrate advanced Python skills, modular software engineering, data management, GUI design, and applied CS50 principles. The final product will run locally on any desktop system that supports Python 3.x, Tkinter, and SQLite.
All work will be completed individually but with adherence to professional SDLC structure, version control through GitHub, and academic documentation standards.

**Work Location & Timeline**

- Entirely virtual (Remote at my area where I live)
- Local development environment using Python, Tkinter, SQLite, and third-party libraries
- Project duration aligns with CMSC 495:
    - **Start Date:** March 16, 2025
    - **Last Final Submission:** December 17, 2025 (Early can be done so date is not included if I decide to turn in earlier.)
    - **Course End Date:** December 30, 2025

**Assumptions, Constraints, Quality, and Costs**

**Assumptions**

- User has basic knowledge of operating a desktop application
- Python and required libraries will install without conflict
- Text-file dictionaries are sufficiently structured for parsing
- Internet access may or may not be available (offline-first approach)

**Constraints**

- Solo project — time management essential
- Software limited to desktop environment
- API limitations: MyMemory free tier may rate-limit requests
- Tkinter UI constraints vs. modern frameworks

**Quality Standards**

- PEP 8 compliant Python code
- Modular architecture (separation of concerns)
- Proper database normalization (where applicable)
- Exception handling and robust defensive programming
- Extensive testing: unit, integration, and GUI-level

**Costs**

- No financial cost — all tools are open-source
- Largest "cost" is time investment

**Project Risks**

**Technical Risks**

- Complex GUI interactions may cause race conditions or freezing
- Database corruption risk if not handled carefully
- API availability not guaranteed
- Multithreading (optional) may introduce sync issues

**Development Risks**

- Single developer = bottleneck for tasks
- Refactoring necessary as project scales
- Potential feature creep

**User Experience Risks**

- UI complexity
- Inconsistent dictionary formats
- Handling unsupported languages

**Mitigation Strategies**

- Thorough weekly testing
- Incremental commits to GitHub
- Strict scope control
- Clear backup/export utilities

**SDLC Process Model**

I will be following the SDLC process model to help develop and deploy the Python Translator App. This allows me to plan in the early stages to prevent major design flaws from developing and below is the SDLC (GeeksforGeeks, n.d.)

This project follows the Systems Development Life Cycle (SDLC) using a modified Waterfall + Iterative Enhancement hybrid model and the Agile Model:

1. Requirements Analysis
2. Design
3. Implementation
4. Testing

5. Deployment
6. Maintenance (ongoing improvements before final submission)

**Application Requirements:** These requirements define the core functionalities of the Python Translator application.

**Functional Requirements (Repeat and expanded)**

**Main UI**

- Translate text between supported languages
- Select source and target languages
- Access translation history
- View dictionary entries with pagination
- Export data (PDF, JSON, SQL)
- Send results via email

**Offline Dictionary**

- Load text-file dictionaries
- Perform exact and partial matching
- Filter/search entries
- Display results with pagination
- Handle missing words gracefully

**Online Translation**

- Use MyMemory API
- Provide fallback for missing local entries
- Identify whether result is “local” or “online”

**Database**

- Store translation history
- Store timestamps, languages, method (local/online)
- Support export/import functionality

**Exporting**

- PDF export using ReportLab
- JSON export
- SQL export / import

**Email Sending**

- Send translation via SMTP
- Configurable sender information
- ONLY WITH OUTLOOK MAIL!!!!

**Technical Requirements**

- Python 3.x
- Tkinter
- SQLite

- Requests (for API calls)
- smtplib (email)
- openpyxl, reportlab, json libraries
- IDE: Visual Studio Code / PyCharm
- Version Control: Git + GitHub
- Documentation: Word + Markdown

The development team will use a Python IDE (e.g., PyCharm or Visual Studio Code) to write and test the code. These IDEs provide excellent debugging tools and support for Python, making them ideal for development. Git will be used for version control, allowing the team to manage changes, track progress, and collaborate effectively. The code will be shared via a Git repository (such as GitHub or GitLab), ensuring that all members have access to the latest updates and can work on different parts of the app concurrently. All project documentation will be maintained in Microsoft Word. This ensures the documentation is easily accessible and compatible for team collaboration and future reference.

# 4. Phases 3-6 Sections

### 6.1. Refactor the Software

Refactoring focused on making the Translator Application modular, maintainable, and scalable.

**Major Refactoring Improvements**

| Module | Improvements Made |
|---|---|
| **gui.py** | Extracted business logic; added test_mode; organized event callbacks; improved layout separation. |
| **translation.py** | Split local dictionary lookup vs. online translator; added fallback logic; added error handling and caching. |
| **spelling.py** | Added language detection, caching, more reliable spell correction, and decoupled it from GUI. |
| **algorithms.py** | Centralized search, pagination, frequency scoring, and utility algorithms used across the app. |
| **database.py** | Full CRUD; added pagination queries; ensured safe parameterized SQL statements. |
| **exports.py** | Refactored PDF generator, registered fonts properly, supported multilingual rendering. |
| **emailer.py** | Improved SMTP handling; better error messages; isolated email logic from UI. |

| Module | Improvements Made |
|---|---|
| **validation.py** | Unified input validation, whitespace cleansing, and language-specific safety checks. |
| **admin_panel.py** | Added dictionary management logic separate from the GUI. |
| **main.py** | Clean single entry point linking all modules; handles initialization safety. |

**Key Refactoring Outcomes**

- Each component is now **testable** independently
- The GUI no longer blocks translations
- Spell checking uses **caching + language detection**
- Translators (local/online) are modular and replaceable
- PDF exports now support **Unicode** (Russian, Chinese, Korean, etc.)
- Database queries are optimized and safe
- Common logic no longer duplicated across modules
- GitHub Actions successfully runs without opening a GUI

**Translator Application's Critical Chosen Algorithms:**

The Translator Application incorporates foundational concepts from Software Engineering, Data Structures, and Algorithms, which align with well-established LeetCode problem patterns. These algorithmic techniques appear throughout core modules, including algorithms.py, spelling.py, translation.py, database.py, and validation.py, each implementing logic that emphasizes performance, scalability, and clarity.

A major example is the dictionary search pipeline implemented across algorithms.py and database.py, where the system first sorts dictionary entries using a custom merge-sort implementation (similar to LeetCode #912: Sort an Array). Once sorted, the search functionality performs Binary Search (mirroring LeetCode #704: Binary Search) to locate exact matches, nearest neighbors, and filtered subsets in logarithmic time. This two-stage strategy of Sort → Binary Search, enables highly efficient lookup, pagination, and suggestion generation even when handling large datasets.

The spell-correction system in spelling.py applies edit-distance and token-similarity scoring reminiscent of LeetCode #72 (Edit Distance), augmented with caching for improved speed. Its fuzzy-matching and autosuggestion logic parallel techniques found in Trie-based LeetCode problems such as #208 (Implement Trie) and #648 (Replace Words).

In translation.py, the application uses a Greedy Failover Strategy to seamlessly switch between local dictionary lookups and online API requests, a decision pattern often discussed in algorithmic resiliency and dynamic error-handling problems. Input cleansing routines in

validation.py use string-normalization patterns similar to LeetCode #125 (Valid Palindrome) and #242 (Valid Anagram) to ensure consistency across translation scenarios.

Database functions in database.py implement stable ordering, paging, and filtered selection using well-known concepts from sorting networks and efficient indexing techniques aligned with the principles detailed by Kleinberg & Tardos (2006) and Mount (n.d.). Export generation in exports.py applies state-machine and text-rendering logic typical of parsing algorithms used in advanced formatting and PDF-generation tasks.

Together, these algorithmic choices with the combined sorting + binary search architecture deliver an efficient, scalable, and academically grounded implementation. Refer to (Leetcode, n.d.); (Tsui, Karam, & Bernal, 2014); (Williams & Zhang, 2020); (Kleinberg & Tardos, 2006); (Erickson, n.d.); (Mount, n.d.); (Nievergelt, n.d.); (Liang, 2023) for foundational background on these algorithmic concepts.

| **Module / Function** | **Purpose / Behavior** | **Matching LeetCode Pattern / Algorithmic Concept** | **LeetCode #** |
|---|---|---|---|
| algorithms.py → *merge_sort()* | Sorts dictionary entries for faster searching | Merge Sort (Divide + Conquer) | #912: Sort an Array |
| algorithms.py → *binary_search()* | Locates word entries in sorted list | Binary Search | #704: Binary Search |
| algorithms.py → *paginate()* | Splits list of results into pages | Sliding Window / Two-Pointer | #209, #3, #643 |
| algorithms.py → *filter_results()* | Filters dictionary rows by substring match | String Matching / Subsequence Search | #392: Is Subsequence |
| spelling.py → *correct_word()* | Suggests nearest correct word using edit distance | Edit Distance (Dynamic Programming) | #72: Edit Distance |
| spelling.py → *generate_candidates()* | Generates possible spelling variants | BFS neighbor generation | #127: Word Ladder |
| spelling.py → *detect_language()* | Estimates the input language | Frequency Analysis / HashMap Patterns | #242: Valid Anagram |
| translation.py → *translate_word()* | Local → online fallback translation | Greedy Failover Strategy (multi-step decision path) | Conceptual |
| translation.py → *pick_best_translation()* | Chooses best translation from | Greedy Choice / Priority Selection | #253: Meeting Rooms II |

| **Module / Function** | **Purpose / Behavior** | **Matching LeetCode Pattern / Algorithmic Concept** | **LeetCode #** |
|---|---|---|---|
| | multiple candidates | | |
| database.py → *query_dictionary()* | SQL filtering + sorted ORDER BY results | Sort + Search Combo | #56/#57 Interval Sorting |
| database.py → *search_and_paginate()* | Keyword search + pagination | Binary Search + Sliding Window | #704 + #209 |
| validation.py → *clean_text()* | Normalizes input (lowercase, strip punctuation) | String Sanitization | #125: Valid Palindrome |
| validation.py → *validate_word()* | Ensures word is valid for processing | Character Checking | #20: Valid Parentheses |
| validation.py → *is_language_supported()* | Verifies supported language codes | HashMap Lookup | #1: Two Sum |
| exports.py → *export_json()* | Serializes structured translation data | Tree/Graph → JSON Mapping | #102/#429 (Level Order) |
| exports.py → *export_pdf()* | Stateful text rendering into PDF | Finite-State Machine / Parser Pattern | Conceptual |
| emailer.py → *send_email()* | Formats and sends structured email | String Construction + Queue-like dispatch | #394: Decode Strings |
| gui.py → *update_results_list()* | Efficiently updates large GUI result lists | Lazy Loading / Incremental Rendering | #173: BST Iterator |
| gui.py → *search_event_handler()* | Handles input, debouncing, and triggers search pipeline | Event-Driven State Machine | Conceptual |
| algorithms.py + spelling.py synergy | Fast sorted lookup → binary search → correction fallback | Combine Sort + Search Architecture | #912 + #704 |

**6.2. Deploy the Software**

**Documentation & Readability Enhancements**

- **Inline Comments** across all modules
- **Optional docstrings** describing behavior, parameters, and returns
- A complete **README.md** including:
    - Installation
    - Usage

    - Exporting
    - Online translator details
    - Troubleshooting
    - CS50 explanations
    - Screenshots and diagrams
- **Requirements files** for both dev and runtime
- **/docs folder** containing UML diagrams and design document
- **/tests** containing unit tests
- **.github/workflows/** containing
    - Windows GitHub Actions
    - Auto-test configurations

**Packaging the Application for Users**
Install requirements:
pip install -r requirements.txt
Run app:
python main.py
The README also explains optional online translation setup.

**Cross-Platform Deployment**
**Option A — GitHub Source Code (Recommended for CS50)**
Users clone/download ZIP and run locally.
**Option B — Packaged Executable (Optional)**
PyInstaller build command documented:
pyinstaller --onefile main.py
Assets added via --add-data as documented in README.

**Hosting Options**
- GitHub Releases (optional)
- Local distribution via ZIP or folder sharing
- Cloud drive backup

**Post-Deployment Processes**
- Bug reporting via GitHub Issues
- Automated CI testing
- Feature roadmap (more languages, improved admin panel, speech-to-text)

**6.3. Maintain the Software**
Maintenance includes:
- Fixing GUI responsiveness

- Improving dictionary file parsing
- Updating online translation APIs
- Optimizing pagination and search
- Expanding unit tests
- Supporting more Unicode fonts and languages

**Refactoring Summary**
During the final development phase:

- Old legacy code from the early monolithic version was removed
- All modules were cleaned and standardized
- Repeated code was pushed into helper modules
- Database schemas were polished
- GUI was made more robust for bad inputs
- Online translator fallback now handles API failures gracefully
- Inline comments explain nearly every block of logic

**Feature Improvements: Phase 1 → Phase 2**

| Module | Phase 1 | Phase 2 (Final) |
|---|---|---|
| gui.py | tightly coupled logic | fully modular GUI w/ test mode |
| translation.py | basic dictionary lookup | multi-source translator w/ fallback, error handling |
| spelling.py | basic English-only correction | multilingual detection + caching |
| exports.py | simple PDFs | Unicode PDFs w/ custom fonts |
| database.py | basic history table | full CRUD, pagination, search |
| admin_panel.py | N/A | now includes dictionary editing |
| algorithms.py | scattered utilities | unified algorithms library |
| main.py | scattered init code | clean entry point |

**Future Maintenance (If I choose to do this after December 2025)**

- Add voice/speech translation
- Enhance dictionary management
- Add live-suggestion autocomplete
- More languages & Unicode support
- Cloud sync (optional future feature)

# 5. Python Design Document (UML/Architecture,

# etc.): Translator Application

This document contains UML class diagrams for each module in the Translator Application, presented as Mermaid code blocks and ASCII-style diagrams for inclusion in the CS50 final project submission.
Note: Auto-generated by a third-party application while analyzing code….

## main.py

Program entry point. Initializes DB and launches TranslatorApp.

### *Mermaid UML Code:*

```
classDiagram
    class Main {
        +main()
    }
    Main : +init_db()
    Main : +run TranslatorApp()
```

### *ASCII-style Diagram:*

```
+--------------------+
|       main.py      |
+--------------------+
| - none             |
+--------------------+
| + main()           |
+--------------------+
```

## gui.py (TranslatorApp)

Tkinter GUI class that manages windows, input, translation flow, and callbacks.

### *Mermaid UML Code:*

```
classDiagram
    class TranslatorApp {
        - word_input: Text
        - lang_var: StringVar
        - result_label: Label
        + __init__(test_mode=False)
        + translate()
        + _online_translate_thread(word, lang_code, lang_name)
        + clear()
        + open_admin()
        + resize_to_fit_text()
    }
```

### *ASCII-style Diagram:*

```
+----------------------------------------+
|           TranslatorApp (gui.py)       |
+----------------------------------------+
| - word_input: Text                      |
| - lang_var: StringVar                   |
| - result_label: Label                   |
+----------------------------------------+
| + __init__(test_mode=False)             |
```

```
| + translate()                              |
| + _online_translate_thread(...)            |
| + clear()                                  |
| + open_admin()                             |
| + resize_to_fit_text()                     |
+--------------------------------------------+
```

## admin_panel.py (AdminPanel, EditDialog)

Admin UI with tabs for Dictionary, History, and Settings. Includes import/export and edit dialogs.

### *Mermaid UML Code:*

```
classDiagram
    class AdminPanel {
        - dict_tree
        - hist_tree
        - dict_page
        - hist_page
        + __init__(master)
        + load_dictionary_page()
        + load_history_page()
        + import_dictionary_csv()
        + export_dictionary_csv()
        + export_history_csv()
        + export_history_pdf()
        + email_history_dialog()
    }

    class EditDialog {
        - row_id
        - table
        + __init__(master,row_id,field1,field2,field3,table,refresh_callback)
        + save()
        + delete_row()
        + delete_history_row()
    }

    AdminPanel --> EditDialog : uses
```

### *ASCII-style Diagram:*

```
+-------------------------------------------------------+
|                AdminPanel (admin_panel.py)            |
+-------------------------------------------------------+
| - dict_tree                                           |
| - hist_tree                                           |
| - dict_page, hist_page                                |
+-------------------------------------------------------+
| + __init__(master)                                    |
| + load_dictionary_page()                              |
| + load_history_page()                                 |
| + import_dictionary_csv()                             |
| + export_dictionary_csv()                             |
| + export_history_csv()                                |
| + export_history_pdf()                                |
```

```
| + email_history_dialog()                                |
+---------------------------------------------------------+

+-----------------------------+
|        EditDialog           |
+-----------------------------+
| - row_id                    |
| - table                     |
+-----------------------------+
| + save()                    |
| + delete_row()              |
| + delete_history_row()      |
+-----------------------------+
```

## database.py

SQLite helpers: init_db, CRUD operations for dictionary and history, pagination helpers.

### *Mermaid UML Code:*

```
classDiagram
    class Database {
        + init_db()
        + insert_dictionary_row(word, translation, language)
        + update_dictionary_row(row_id, word, translation, language)
        + delete_dictionary_row(row_id)
        + query_dictionary(filter_text=None, order_by='id DESC', limit=None,
offset=None)
        + query_history(filter_text=None, order_by='id DESC', limit=None,
offset=None)
        + save_history(input_word, output_word, language, used_online=False)
    }
```

### *ASCII-style Diagram:*

```
+----------------------------------------+
|              database.py               |
+----------------------------------------+
| - DB_NAME (from settings)              |
+----------------------------------------+
| + init_db()                            |
| + insert_dictionary_row(...)           |
| + update_dictionary_row(...)           |
| + delete_dictionary_row(row_id)        |
| + query_dictionary(...)                |
| + query_history(...)                   |
| + save_history(...)                    |
+----------------------------------------+
```

## translation.py (sql_translate, online_translate)

Translation logic: local SQL lookup and online translation wrapper.

### *Mermaid UML Code:*

```
classDiagram
    class Translator {
        + sql_translate(word, lang_code)
```

```
        + online_translate(word, lang_code)
    }
```

***ASCII-style Diagram:***

```
+-------------------------------------+
|         translation.py              |
+-------------------------------------+
| - (uses external API libs optionally) |
+-------------------------------------+
| + sql_translate(word, lang_code)    |
| + online_translate(word, lang_code) |
+-------------------------------------+
```

## exports.py (export_history_json, export_history_pdf)

Export utilities: JSON and PDF (ReportLab) with UTF-8 font fallback to text.

***Mermaid UML Code:***

```
classDiagram
    class ExportUtils {
        + export_history_json(path, rows)
        + export_history_pdf(path, rows, title='Translation History')
    }
```

***ASCII-style Diagram:***

```
+---------------------------------------------+
|                exports.py                   |
+---------------------------------------------+
| + export_history_json(path, rows)           |
| + export_history_pdf(path, rows, title)     |
+---------------------------------------------+
```

## emailer.py (send_history_email)

Email helper: Outlook COM draft or mailto fallback.

***Mermaid UML Code:***

```
classDiagram
    class EmailUtils {
        + send_history_email(to_addr, subject, body, attachment_path=None)
    }
```

***ASCII-style Diagram:***

```
+-----------------------------------------+
|              emailer.py                 |
+-----------------------------------------+
| + send_history_email(to_addr,subject,body, |
|                      attachment_path=None) |
+-----------------------------------------+
```

## algorithms.py

Educational algorithm implementations: selection_sort and binary_search used in admin search.

#### *Mermaid UML Code:*

```
classDiagram
    class Algorithms {
        + selection_sort(arr)
        + binary_search(sorted_list, target)
    }
```

#### *ASCII-style Diagram:*

```
+----------------------------------+
|           algorithms.py          |
+----------------------------------+
| + selection_sort(arr)            |
| + binary_search(sorted_list,target)|
+----------------------------------+
```

### settings.py

Configuration constants: DB_NAME, LANG_OPTIONS, PAGE_SIZE, TXT_FILES, DEEP_TRANSLATOR_AVAILABLE, font paths.

#### *Mermaid UML Code:*

```
classDiagram
    class Settings {
        - DB_NAME
        - LANG_OPTIONS
        - PAGE_SIZE
        - TXT_FILES
        - DEEP_TRANSLATOR_AVAILABLE
        - FONT_PATHS
    }
```

#### *ASCII-style Diagram:*

```
+----------------------------+
|         settings.py        |
+----------------------------+
| - DB_NAME                  |
| - LANG_OPTIONS             |
| - PAGE_SIZE                |
| - TXT_FILES                |
| - DEEP_TRANSLATOR_AVAILABLE |
+----------------------------+
```

### System Architecture (Mermaid)

```
flowchart TD
    User((User)) -->|Input| GUI[TranslatorApp (Tkinter)]
    GUI --> Translator[translation.py]
    Translator -->|Local lookup| DB[(SQLite Database)]
    Translator -->|Online| OnlineAPI((MyMemory API / Deep Translator))
    GUI --> Admin[AdminPanel]
    Admin --> Database
    GUI --> Exporter[exports.py]
```

```
    Exporter --> Files[(JSON / PDF)]
    GUI --> Emailer[emailer.py]
    Emailer --> EmailClient((Outlook / mailto))
```

---

## 1) Full UML (Mermaid) + ASCII diagrams for each module

### admin_panel.py

```
classDiagram
    class AdminPanel {
        - dict_tree
        - hist_tree
        - dict_page
        - hist_page
        + __init__(master)
        + load_dictionary_page()
        + load_history_page()
        + import_dictionary_csv()
        + export_dictionary_csv()
        + export_history_csv()
        + export_history_pdf()
        + email_history_dialog()
    }
    class EditDialog {
        - row_id
        - table
        + __init__(master,row_id,field1,field2,field3,table,refresh_callback)
        + save()
        + delete_row()
        + delete_history_row()
    }
    AdminPanel --> EditDialog : uses
```

ASCII:

```
+-------------------------------------------------------+
|                AdminPanel (admin_panel.py)            |
+-------------------------------------------------------+
| - dict_tree                                           |
| - hist_tree                                           |
| - dict_page, hist_page                                |
+-------------------------------------------------------+
| + __init__(master)                                    |
| + load_dictionary_page()                              |
| + load_history_page()                                 |
| + import_dictionary_csv()                             |
| + export_dictionary_csv()                             |
| + export_history_csv()                                |
| + export_history_pdf()                                |
| + email_history_dialog()                              |
+-------------------------------------------------------+
```

---

### algorithms.py

```
classDiagram
    class Algorithms {
        + selection_sort(arr)
        + binary_search(sorted_list, target)
```

```
    }
ASCII:
+----------------------------------+
|           algorithms.py          |
+----------------------------------+
| + selection_sort(arr)            |
| + binary_search(sorted_list,target)|
+----------------------------------+
```

---

## database.py

```
classDiagram
    class Database {
        + init_db()
        + insert_dictionary_row(word, translation, language)
        + update_dictionary_row(row_id, word, translation, language)
        + delete_dictionary_row(row_id)
        + query_dictionary(filter_text=None, order_by='id DESC', limit=None,
offset=None)
        + query_history(filter_text=None, order_by='id DESC', limit=None,
offset=None)
        + save_history(input_word, output_word, language, used_online=False)
    }
```

## ASCII:

```
+----------------------------------------+
|              database.py               |
+----------------------------------------+
| - DB_NAME (from settings)              |
+----------------------------------------+
| + init_db()                            |
| + insert_dictionary_row(...)           |
| + update_dictionary_row(...)           |
| + delete_dictionary_row(row_id)        |
| + query_dictionary(...)                |
| + query_history(...)                   |
| + save_history(...)                    |
+----------------------------------------+
```

---

## emailer.py

```
classDiagram
    class Emailer {
        + send_history_email(to_addr, subject, body, attachment_path=None)
    }
ASCII:
+-----------------------------------------+
|               emailer.py                |
+-----------------------------------------+
| + send_history_email(to_addr,subject,body, |
|                      attachment_path=None) |
+-----------------------------------------+
```

---

## exports.py

```
classDiagram
    class ExportUtils {
```

```
        + export_history_json(path, rows)
        + export_history_pdf(path, rows, title='Translation History')
    }
ASCII:
+-----------------------------------------------+
|                 exports.py                    |
+-----------------------------------------------+
| + export_history_json(path, rows)             |
| + export_history_pdf(path, rows, title)       |
+-----------------------------------------------+
```

---

**gui.py (TranslatorApp)**

```
classDiagram
    class TranslatorApp {
        - word_input: Text
        - lang_var: StringVar
        - result_label: Label
        + __init__(test_mode=False)
        + translate()
        + _online_translate_thread(word, lang_code, lang_name)
        + clear()
        + open_admin()
        + resize_to_fit_text()
    }
```

ASCII:

```
+----------------------------------------+
|           TranslatorApp (gui.py)       |
+----------------------------------------+
| - word_input: Text                     |
| - lang_var: StringVar                  |
| - result_label: Label                  |
+----------------------------------------+
| + __init__(test_mode=False)            |
| + translate()                          |
| + _online_translate_thread(...)        |
| + clear()                              |
| + open_admin()                         |
| + resize_to_fit_text()                 |
+----------------------------------------+
```

---

main.py

```
classDiagram
    class Main {
        + main()
    }
    Main : +init_db()
    Main : +run TranslatorApp()
```

ASCII:

```
+--------------------+
|       main.py      |
+--------------------+
| - none             |
+--------------------+
| + main()           |
```

```
+--------------------+
```

---

## requirements.txt / requirements-dev.txt

```
(plain text; list of packages)
requirements.txt – contains runtime dependencies (e.g., requests, reportlab, deep-
translator (optional), python-docx, python-pptx)
requirements-dev.txt – contains development deps (pytest, flake8, etc.)
```

---

## settings.py

```
classDiagram
    class Settings {
        - DB_NAME
        - LANG_OPTIONS
        - PAGE_SIZE
        - TXT_FILES
        - DEEP_TRANSLATOR_AVAILABLE
        - FONT_PATHS
    }
```

### ASCII:

```
+-----------------------------+
|         settings.py         |
+-----------------------------+
| - DB_NAME                   |
| - LANG_OPTIONS              |
| - PAGE_SIZE                 |
| - TXT_FILES                 |
| - DEEP_TRANSLATOR_AVAILABLE |
+-----------------------------+
```

---

## spelling.py

```
classDiagram
    class Spelling {
        + auto_correct(word)
        + load_cache()
        + save_cache()
    }
```

### ASCII:

```
+-----------------------------------+
|           spelling.py             |
+-----------------------------------+
| + auto_correct(word)              |
| + load_cache()                    |
| + save_cache()                    |
+-----------------------------------+
```

---

## translation.py

```
classDiagram
    class Translator {
        + sql_translate(word, lang_code)
        + online_translate(word, lang_code)
    }
```

ASCII:

```
+-------------------------------------+
|         translation.py              |
+-------------------------------------+
| - (uses external API libs optionally) |
+-------------------------------------+
| + sql_translate(word, lang_code)    |
| + online_translate(word, lang_code) |
+-------------------------------------+
```

---

validation.py

```
classDiagram
    class Validation {
        + validate_input_word(word)
    }
```

ASCII:

```
+------------------------------+
|         validation.py        |
+------------------------------+
| + validate_input_word(word)  |
+------------------------------+
```

# 6. Schedule

Below is our team's sample of the Gantt Chart of our plans, and we will be using this schedule to make sure we stay focused; however, plans are not set completely and therefore might be changed (UMGC, n.d.).

| **Lead** | **Topic** | **Description** | **Deliverables** |
|---|---|---|---|
| Developer | Project Selection | Finalize idea, gather requirements | N/A |
| PM | Project Plan | Milestones, structure, deliverables | Project Plan |
| Documentation | Requirements | Draft SOW, finalize architecture | Project Plan |
| Developer | Design | UI structure, UML diagrams, module layout | Design + UML |
| Developer | Phase 1 Source | Core modules, dictionary loader, DB | Phase 1 Source |
| Tester | Testing | Manual tests + refinements | Test Plan |
| Developer | Phase 2 Source | GUI finalization, API, email, export | Phase 2 + User Guide |
| Developer | Finalization | Debugging + improvements | Final Phase 2 |
| PM | Final Report | Integrate all final documentation | Final Report + |

Project Plan - Last Update: 27 November 2025

| Lead | Topic | Description | Deliverables |
|---|---|---|---|
| | | | Demo |

**Note:** The Last day for CS50 Submissions for my project is December 17, 2025, at 11:59 pm PST. I did not put dates since I may turn it in earlier than the expected date.
The expected date to finish is 12/17/2025.

# 7. References

Amazon Web Services. (n.d.). *What is unit testing?* Amazon Web Services, Inc. https://aws.amazon.com/what-is/unit-testing/

Deitel, P. J., & Deitel, H. (2015). *Java: How to program (Early objects)* (10th ed.). Pearson. Online textbook available. (n.d.)

Devaraj, K. (2024, December 19). 6 key phases of testing in software testing. *Testsigma*. https://testsigma.com/blog/phases-of-testing/

Doe, J., & Lee, A. (2020). Creating effective contribution reports in software projects: Guidelines and strategies. *IEEE Software Engineering Journal*, 45(6), 102–115. https://doi.org/10.1109/SEJ.2020.3082548

Erickson, J. (n.d.). *Algorithms*. University of Illinois at Urbana-Champaign. Online textbook available.

GeeksforGeeks. (n.d.). *Software Development Life Cycle (SDLC)*. https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/

GeeksforGeeks. (2024). *Control Flow Software Testing*. https://www.geeksforgeeks.org/control-flow-software-testing/

GeeksforGeeks. (2024). *Software engineering: Differences between coupling and cohesion*. https://www.geeksforgeeks.org/software-engineering-differences-between-coupling-and-cohesion/

GeeksforGeeks. (2024). *Difference between Function-Oriented Design and Object-Oriented Design*. https://www.geeksforgeeks.org/difference-between-function-oriented-design-and-object-oriented-design/

GeeksforGeeks. (2024). *Object-Oriented Design (OOP)*. https://www.geeksforgeeks.org/oops-object-oriented-design/

GeeksforGeeks. (2024). *Functions decomposition in software engineering*. https://www.geeksforgeeks.org/functions-decomposition-in-software-engineering/

GeeksforGeeks. (2024). *What is a Project Plan*. https://www.geeksforgeeks.org/project-plan/

Gehry, F. (Producer). (2019, April 10). *How to make your design stand out* [Video]. YouTube. https://www.youtube.com/watch?v=Va64slVDpI4

Git. (n.d.). *Git user manual*. https://git-scm.com/docs/user-manual

Jones, M., & Davis, L. (2019). Effective project design planning in software engineering: Best practices and strategies. *IEEE Software Engineering Conference Proceedings*, 31(2), 95–110. https://doi.org/10.1109/SEConf.2019.2787093

Khan, S., & Patel, A. (2018). Designing and implementing a centralized game hub for multiplayer platforms. *IEEE Transactions on Game Development*, 39(1), 77–89. https://doi.org/10.1109/TGD.2018.2876062

Kleinberg, J., & Tardos, É. (2006). *Algorithm design*. Pearson Education.

LeetCode. (n.d.). *LeetCode - The world's leading online programming learning platform*. LeetCode. Retrieved April 12, 2025, from https://leetcode.com/

Lehman, E., Leighton, F. T., & Meyer, A. R. (2021). *Mathematics for computer science*. LibreTexts, MIT, & UC Davis.

Liang, Y. D. (2023, May 23). *Introduction to Java programming and data structures* (13th ed.). Pearson. Online textbook available. (n.d.)

Mount, D. (n.d.). *Algorithms [Lectures CMSC 251]*. University of Maryland Global Campus.

Nievergelt, J. (n.d.). *Algorithms and data structures*. University of Maryland Global Campus.

Sierra, K., & Bates, B. (2005). *Head First Java* (1st ed.). O'Reilly Media.

Smith, J., & Taylor, R. (2021). Using Git for version control in Python-based game development. *IEEE Transactions on Software Engineering*, 47(3), 215–230. https://doi.org/10.1109/TSE.2021.3056789

Solomon, J. (2015). *Numerical algorithms*. MIT.

Stellman, A., & Greene, J. (2005). Software project planning. In *Applied software project management* (Part I, Chapters 5, 6, 8, 9, 10, 11). O'Reilly Media. https://go.oreilly.com/umgc/https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html

Sweigart, A. (n.d.). *Making games with Python & Pygame*. https://inventwithpython.com/makinggames.pdf

Tsui, F., Karam, O., & Bernal, B. (2014). *Essentials of software engineering* (3rd ed.). Jones and Bartlett Learning. https://library-books24x7-com.ezproxy.umgc.edu/toc.aspx?site=VGX8U&bookID=51648

University of Maryland Global Campus. (n.d.). *Sample Project Plan*. https://leocontent.umgc.edu/content/dam/course-content/tus/cmsc/cmsc-495/document/2020%20Sample%20Project%20Design%2004-2022.pdf?ou=1248245

University of Maryland Global Campus. (n.d.). *Sample Project Plan 1*. UMGC. https://leocontent.umgc.edu/content/dam/course-content/tus/cmsc/cmsc-495/document/Sample%20project%20plan%20UMGC%20CMSC%20495.pdf?ou=1248245

University of Maryland Global Campus. (n.d.). *Sample Project Plan 2*. UMGC. https://leocontent.umgc.edu/content/dam/course-content/tus/cmsc/cmsc-495/document/Sample%20Project%20Plan%202.pdf?ou=1248245

Unity Technologies. (2020, September 3). *Testing and quality assurance tips for Unity projects*. https://unity.com/how-to/testing-and-quality-assurance-tips-unity-projects

Winters, T., Manshreck, T., & Wright, H. (2020). Software engineering at Google (Chs. 18–19). O'Reilly Media. https://learning.oreilly.com/library/view/software-engineering-at/9781492082781/ch18.html

## 8. Table: Project Plan – GitHub Projects Tab

**Note:** The Last day for CS50 Submissions for my project is December 17, 2025, at 11:59 pm PST. These dates are just predictions if I plan on extending it; otherwise, I may turn it in earlier than the expected date. The expected completion date is 12/17/2025.

**GitHub Projects Link: https://github.com/VictoriaRaven/Translator-Language-Dictionary-App/projects**

**GitHub Actions Link: https://github.com/VictoriaRaven/Translator-Language-Dictionary-App/actions**

**GitHub Issues Link: https://github.com/VictoriaRaven/Translator-Language-Dictionary-App/issues**

**GitHub Pull Request and Push Links: https://github.com/VictoriaRaven/Translator-Language-Dictionary-App/pulls**

**REFER TO THESE LINKS TO SEE THE ACTUAL DUE DATES AND OTHER UPDATES!**