AutoLoc App Architectural Design Overview

The AutoLoc App is designed to automate the localization process of software source code for various types of applications, including websites, mobile apps, and server-side applications. It is built using the PySide6 Python framework and organized into several modules that manage different aspects of localization, database management, user interface, and application settings.

A screenshot of a computer

Description automatically generated

# Objectives

The primary objectives of the AutoLoc App are:

- To provide an automated solution for extracting, managing, and localizing text from source code files.

- To ensure compatibility across different platforms (Web, Android, iOS, Java) by leveraging specific localization handlers for each file type.

- To maintain a modular and scalable architecture that can be easily extended to support new file types and localization requirements.

# Key Components and Modules

The architecture of the AutoLoc App is divided into three main modules:

1. UI Module (`app\_ui`): Manages the user interface and user interactions.

2. Manager Module (`managers`): Manages core functionalities like app settings, database management, and localization logic.

3. Localizer Module (`localizers`): Handles the localization process for different types of source code files.

# Component-Level Design

Below is a detailed explanation of each component within the AutoLoc App in a concise tabular format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Component Name** | **Module** | **Responsibilities** | **Dependencies** | **Key Methods/Functions** | **Interactions** | **Data Managed** | **Execution Context** |
| MainAppWindow | app\_ui | Main entry point for the app; initializes and manages UI components and overall app flow | AppManager, DBManager | initialize(), setupUI() | Interacts with UI widgets, AppManager, and DBManager to control app behavior | Application state data | Main thread |
| HeaderWidget | app\_ui | Displays the app header and manages header interactions | None | N/A | Part of the UI; integrated into MainAppWindow | UI elements | Main thread |
| SideBarWidget | app\_ui | Provides navigation sidebar with buttons for different app functionalities | None | N/A | Integrated with MainAppWindow; interacts with other widgets | UI elements | Main thread |
| FooterWidget | app\_ui | Displays the app footer information | None | N/A | Part of the UI; integrated into MainAppWindow | UI elements | Main thread |
| HomeWidget | app\_ui | Renders the home screen with a banner and welcome information | None | N/A | Interacts with MainAppWindow for display | UI elements | Main thread |
| SettingsStackedWidget | app\_ui | Manages app settings UI, allows changing localization settings and themes | SettingManager | render(), updateSettings() | Uses SettingManager for CRUD operations | Application settings | Main thread |
| ProjectsStackedWidget | app\_ui | Manages UI for project management | ProjectManager | render(), updateProjects() | Uses ProjectManager for CRUD operations | Project data | Main thread |
| SourceCodesStackedWidget | app\_ui | Manages UI for handling source code management | SourceCodeManager, L10nManager | render(), updateSourceCodes() | Uses SourceCodeManager and L10nManager for CRUD and localization tasks | Source code data, localization configurations | Main thread |
| AppManager | managers | Manages app startup, initializes directories and configurations | None | initialize\_app(), get\_app\_data\_path() | Provides paths and configurations to other managers | Configuration data | Main thread |
| DBManager | managers | Handles database creation, table management, and CRUD operations | AppManager | create\_db(), create\_tables(), insert\_records() | Manages SQLite database for storing application data | Database records (projects, locales, etc.) | Separate thread |
| SettingManager | managers | Manages all application settings, such as UI mode and localization preferences | DBManager | set\_ui\_mode(), set\_key\_format(), html\_is\_duplicated() | Updates and retrieves app settings data from the database | Localization and UI settings | Main thread |
| ProjectManager | managers | Handles project management, including CRUD operations for project data | DBManager | get\_projects(), add\_project(), delete\_project() | Manages project data and synchronizes with the UI components | Project records in the database | Main thread |
| SourceCodeManager | managers | Manages source code files, handles localization preparation tasks | DBManager, AppManager | add\_source\_code(), save\_source\_code\_files(), export\_files() | Manages and prepares source code for localization | Source code data, localization configurations | Separate thread |
| L10nManager | managers | Coordinates the localization processes for various source code types | SourceCodeManager | reset\_l10n(), localize\_source\_code() | Invokes the appropriate localizer class based on file type | Localization data | Separate thread |
| WebAppBasedLocalizer | localizers | Localizes web-based source code files, managing HTML, JS, and related formats | WebAppFileHandler | process\_l10n(), get\_file\_type\_counts() | Uses specific file handlers to manage localization of different file types | Web source code files | Separate thread |
| WebAppFileHandler | localizers | Base class for web app file handlers; provides shared functionality for handling various file types | SettingManager, SourceCodeManager | get\_files\_by\_extension(), generate\_key(), save\_resource\_file() | Handles file operations for localization, delegates tasks to specific file handlers | Localization files for various locales | Separate thread |
| AndroidAppBasedLocalizer | localizers | Localizes Android app source code files | AndroidAppFileHandler | process\_l10n() | Similar to WebAppBasedLocalizer, but for Android-specific files | Android source code files | Separate thread |
| IOSAppBasedLocalizer | localizers | Localizes iOS app source code files | IOSAppFileHandler | process\_l10n() | Similar to WebAppBasedLocalizer, but for iOS-specific files | iOS source code files | Separate thread |
| JavaAppBasedLocalizer | localizers | Localizes Java server-side app source code files | JavaAppFileHandler | process\_l10n() | Similar to WebAppBasedLocalizer, but for Java-specific files | Java server-side source code files | Separate thread |

# Additional Information

## Execution Context

- Main Thread: Used for UI-related components and operations to ensure a responsive and interactive interface.

- Separate Threads: Used for long-running tasks such as localization and database operations to prevent blocking the main thread and improve application performance.

## Data Management

- The `DBManager` is responsible for managing all database operations, including creating necessary tables (`ui\_modes`, `l10n\_settings`, `locales`, `projects`, `source\_codes`, and `source\_code\_target\_locales`), and performing CRUD operations.

- The `SourceCodeManager` and `L10nManager` work closely together to prepare source codes for localization, handle file operations, and manage localization data.

## Interactions and Workflow

- The app starts with the `MainAppWindow`, initializing the core components (`AppManager` and `DBManager`) and setting up the user interface.

- The `AppManager` manages startup configurations and paths, while the `DBManager` handles database-related operations.

- User interactions with the UI components (like adding a project or uploading source code) trigger actions in the `SettingManager`, `ProjectManager`, or `SourceCodeManager`, which in turn interact with the database.

- When a localization task is initiated, the `L10nManager` determines the type of source code and delegates the localization to the appropriate localizer class (`WebAppBasedLocalizer`, `AndroidAppBasedLocalizer`, etc.), which handles specific file types.

## Conclusion

The AutoLoc App’s architecture is modular, scalable, and designed for extensibility. It separates concerns across distinct modules, enabling easy maintenance and support for future enhancements. The app leverages PySide6 for a rich user interface and uses a combination of threading to ensure smooth and efficient operations without blocking the main application thread. The design supports the primary goals of the AutoLoc App: to provide an automated, efficient, and user-friendly solution for managing the localization of software source codes across various platforms.