AutoLoc App Architectural Design Overview

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

# Architectural Design

A screenshot of a computer

Description automatically generated

# Execution Flow

A screenshot of a diagram

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 using 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 organized into three main modules:

* UI Module (`app\_ui`): Manages the user interface and user interactions.
* Manager Module (`managers`): Manages core functionalities like app settings, database management, and localization logic.
* Localizer Module (`localizers`): Handles the localization process for different types of source code files.

# Future Enhancements

The current version of the architectural design does not include the full class list for the `android\_localizers` and `ios\_localizers` modules. These modules are intended to handle localization for Android and iOS apps, respectively. However, the specific classes and their functionalities within these modules have not yet been fully conceptualized and are planned for future versions. As the design evolves, subsequent versions will provide detailed definitions and responsibilities for these components to ensure comprehensive support for all target platforms.

This phased approach allows the design to remain flexible and adaptable as new requirements and use cases emerge. Future versions will also incorporate any additional components or modifications needed to enhance the app's localization capabilities across different platforms.

# Component-Level Design

Below is a concise explanation of each component within the AutoLoc App.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |
| **SettingsWidget** | app\_ui | Displays and manages general settings in the UI | SettingManager | set\_ui\_mode(), set\_key\_format() | Interacts with SettingsStackedWidget | UI and localization settings | Main thread |
| **LocalesWidget** | app\_ui | Manages the locales settings, allowing addition, deletion, and update of locales | SettingManager | add\_locale(), delete\_locale() | Interacts with SettingsStackedWidget and SettingManager | Locale settings data | Main thread |
| **LocaleWidget** | app\_ui | Handles single locale settings and displays them | SettingManager | update\_locale() | Part of the locales management in SettingsStackedWidget | Single locale data | Main thread |
| **AddLocaleWidget** | app\_ui | Provides UI for adding new locales | SettingManager | add\_locale() | Adds new locale records to the database via SettingManager | New locale data | Main thread |
| **UpdateLocaleWidget** | app\_ui | Provides UI for updating existing locales | SettingManager | update\_locale() | Updates locale records in the database via SettingManager | Updated locale data | Main thread |
| **ProjectsStackedWidget** | app\_ui | Manages the UI for project management | ProjectManager | render(), updateProjects() | Uses ProjectManager for CRUD operations | Project data | Main thread |
| **ProjectsWidget** | app\_ui | Displays and manages the list of projects in the UI | ProjectManager | add\_project(), delete\_project() | Interacts with ProjectsStackedWidget and ProjectManager | List of project records | Main thread |
| **ProjectWidget** | app\_ui | Displays detailed information for a single project | ProjectManager | update\_project() | Part of the project management in ProjectsStackedWidget | Single project data | Main thread |
| **AddProjectWidget** | app\_ui | Provides UI for adding new projects | ProjectManager | add\_project() | Adds new project records to the database via ProjectManager | New project data | Main thread |
| **UpdateProjectWidget** | app\_ui | Provides UI for updating existing projects | ProjectManager | update\_project() | Updates project records in the database via ProjectManager | Updated 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 |
| **SourceCodesWidget** | app\_ui | Displays and manages the list of source codes in the UI | SourceCodeManager | add\_source\_code(), delete\_source\_code() | Interacts with SourceCodesStackedWidget and SourceCodeManager | List of source code records | Main thread |
| **SourceCodeWidget** | app\_ui | Displays detailed information for a single source code | SourceCodeManager | update\_source\_code() | Part of the source code management in SourceCodesStackedWidget | Single source code data | Main thread |
| **AddSourceCodeWidget** | app\_ui | Provides UI for adding new source codes | SourceCodeManager | add\_source\_code() | Adds new source code records to the database via SourceCodeManager | New source code data | Main thread |
| **UpdateSourceCodeWidget** | app\_ui | Provides UI for updating existing source codes | SourceCodeManager | update\_source\_code() | Updates source code records in the database via SourceCodeManager | Updated source code data | Main thread |
| **MergeSourceCodeLocalesCommonsWidget** | app\_ui | Provides UI to merge common locale files for source codes | SourceCodeManager | merge\_locale\_commons() | Merges common locale files using SourceCodeManager | Merged locale data | Main thread |
| **SourceCodeTargetLocalesWidget** | app\_ui | Displays target locales associated with specific source codes | SourceCodeManager | get\_source\_code\_target\_locales() | Part of SourceCodesStackedWidget | Target locale data for source codes | Main thread |
| **SourceCodeTargetLocaleWidget** | app\_ui | Handles single target locale settings and displays them | SourceCodeManager | update\_source\_code\_target\_locale() | Manages individual target locales for source codes | Single target locale data | Main thread |
| **AddSourceCodeTargetLocaleWidget** | app\_ui | Provides UI for adding new target locales to source codes | SourceCodeManager | add\_source\_code\_target\_locale() | Adds new target locale records via SourceCodeManager | New target locale data | Main thread |
| **UpdateSourceCodeTargetLocaleWidget** | app\_ui | Provides UI for updating existing target locales | SourceCodeManager | update\_source\_code\_target\_locale() | Updates target locale records in the database via SourceCodeManager | Updated target locale data | Main thread |
| **L10nWidget** | app\_ui | Manages localization tasks in the UI and displays localization progress | L10nManager | start\_localization(), show\_progress() | Interacts with L10nManager to start and display localization progress | Localization progress data | Main thread |
| **Styles** | app\_ui | Defines and manages UI styles and themes for consistent appearance | None | apply\_style(), get\_current\_style() | Applies styles to various UI components | UI style data | 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() | 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 |
| **WebAppHTMLFileHandler** | web\_localizers | Handles HTML file localization for web apps | WebAppFileHandler | process\_html\_files() | Specific to handling HTML files for web localization | HTML source files | Separate thread |
| **WebAppJSFileHandler** | web\_localizers | Handles JavaScript file localization for web apps | WebAppFileHandler | process\_js\_files() | Specific to handling JavaScript files for web localization | JavaScript source files | Separate thread |
| **WebAppEJSFileHandler** | web\_localizers | Handles EJS file localization for web apps | WebAppFileHandler | process\_ejs\_files() | Specific to handling EJS files for web localization | EJS source files | Separate thread |
| **WebAppJSXFileHandler** | web\_localizers | Handles JSX file localization for web apps | WebAppFileHandler | process\_jsx\_files() | Specific to handling JSX files for web localization | JSX source files | Separate thread |
| **WebAppTSFileHandler** | web\_localizers | Handles TypeScript file localization for web apps | WebAppFileHandler | process\_ts\_files() | Specific to handling TypeScript files for web localization | TypeScript source files | Separate thread |
| **WebAppTSXFileHandler** | web\_localizers | Handles TSX file localization for web apps | WebAppFileHandler | process\_tsx\_files() | Specific to handling TSX files for web localization | TSX source files | Separate thread |
| **WebAppVueFileHandler** | web\_localizers | Handles Vue file localization for web apps | WebAppFileHandler | process\_vue\_files() | Specific to handling Vue files for web localization | Vue source files | Separate thread |
| **WebAppJSONFileHandler** | web\_localizers | Handles JSON file localization for web apps | WebAppFileHandler | process\_json\_files() | Specific to handling JSON files for web localization | JSON source files | Separate thread |
| **WebAppCSVFileHandler** | web\_localizers | Handles CSV file localization for web apps | WebAppFileHandler | process\_csv\_files() | Specific to handling CSV files for web localization | CSV source files | Separate thread |
| **AndroidAppBasedLocalizer** | android\_localizers | Localizes Android app source code files | AndroidAppFileHandler | process\_l10n() | Manages Android-specific files for localization | Android source code files | Separate thread |
| **AndroidAppFileHandler** | android\_localizers | Handles file operations specific to Android app localization | SettingManager, SourceCodeManager | get\_android\_files(), generate\_key(), save\_android\_resource\_file() | Provides Android file handling and localization logic | Android resource files | Separate thread |
| **IOSAppBasedLocalizer** | ios\_localizers | Localizes iOS app source code files | IOSAppFileHandler | process\_l10n() | Manages iOS-specific files for localization | iOS source code files | Separate thread |
| **IOSAppFileHandler** | ios\_localizers | Handles file operations specific to iOS app localization | SettingManager, SourceCodeManager | get\_ios\_files(), generate\_key(), save\_ios\_resource\_file() | Provides iOS file handling and localization logic | iOS resource files | Separate thread |
| **JavaAppBasedLocalizer** | java\_localizers | Localizes Java server-side app source code files | JavaAppFileHandler | process\_l10n() | Manages Java-specific files for localization | Java server-side source code files | Separate thread |
| **JavaAppFileHandler** | java\_localizers | Handles file operations specific to Java server-side app localization | SettingManager, SourceCodeManager | get\_java\_files(), generate\_key(), save\_java\_resource\_file() | Provides Java file handling and localization logic | Java resource 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`, which initializes the core components (`AppManager` and `DBManager`) and sets 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.