GitMaven Installer - Technical Documentation

1. Overview

GitMaven Installer is a Java-based cross-platform desktop application that streamlines the process of cloning, building, and installing Maven-based applications from GitHub or GitLab. It uses a graphical interface built with Java Swing and supports shortcut generation for each major OS.

2. Design Philosophy

The application emphasizes modularity and platform independence. The GUI acts as an orchestrator, delegating responsibilities to specialized classes. Extension points were intentionally placed to support future enhancements such as plugin-based builders, new repository providers (e.g., Bitbucket), and alternate build systems (e.g., Gradle).

3. Workflow Walkthrough

The application's core flow is as follows:

1. On launch, system requirements are validated (Java, Maven, Git).

2. The user provides a Git repository URL.

3. Branches are fetched (via Git CLI) for user selection.

4. User selects a target path and custom Maven repo (optional).

5. The repository is cloned. On success, README and owner info are extracted.

6. The application builds the project using Maven (with or without custom repo).

7. After a successful build, a JAR is located.

8. The user sets app name and icon, and a platform-specific shortcut is created.

4. Module & Class Responsibilities

- GUI: `GitMavenCloneUI` sets up and drives all UI actions.

- Git: `CloneRepository`, `FetchGitBranches`, `FetchGitOwnerInfo`, `FetchReadMeInfo`.

- Build: `RunMavenBuild`, `PomHelper`, `JarFinder`.

- Installer: `CreateInstaller`, `WindowsInstaller`, `MacInstaller`, `LinuxInstaller`.

- Helpers: `OperationSystemChecker`, `RepositoryHelper`, `CheckerHelper`.

5. Complex Functionalities

a) Git Repository Cloning:

The `CloneRepository` class handles both GitHub and GitLab.
It first checks whether the repository is GitHub or GitLab.
In the case of GitLab, the application tries to fetch the repository without authentication first. If that fails, it retries using a provided username and access token. This logic exists because GitLab returns an error for public repositories when a token is unnecessarily included in the headers.
For GitHub, the GitHub API automatically handles credentials and may prompt the user via an authentication window.
Upon successful cloning, the application proceeds to extract and display the README file and fetch contact information.

b) Platform Installer Creation:

Each OS has a dedicated class. On Windows, `.lnk` files are created using `mslinks`. On macOS, `.app` bundles are generated with AppleScript and Info.plist generation. Linux first attempts Flatpak packaging and falls back to `.desktop` file creation.

c) Requirement Checker:

`RequirementsChecker` checks versions and displays HTML content with status and download links, improving user experience while enforcing minimum tool versions.

6. Extension Points

- Support for additional version control platforms (e.g., Bitbucket).

- Support for automatically downloading and install missing SDKs.

- Support autoupdate the installer based on update on git repository.

- Support for more customizing the installer.

7. Error Handling Strategy

- UI-level validations and warnings for empty inputs and bad paths.

- SwingWorkers to handle async operations and keep UI responsive.

- Try-catch blocks around system calls, with user-friendly output in `outputConsole`.

8. Security Considerations

- GitLab tokens are never saved and only used in-memory for HTTPS clone.

- Network operations are not logged or persisted.

- All file system interactions are restricted to user-specified paths.

9. Extensibility Notes

- Future refactor: introduce interface-based plugin loader for build systems.

- Injecting dependency logic (e.g., through ServiceLoader) for fetch/build actions.

- Abstract out repository providers into interfaces (GitHub, GitLab, Bitbucket).

10. Appendix

Libraries used:

- mslinks (for Windows shortcut creation)

- org.json and jackson (for JSON parsing)

- Java Swing and AWT (for GUI components)

11. Libraries Used

- jackson-core, jackson-databind, jackson-dataformat (csv, xml, yaml, toml, properties) – For parsing different data formats

- json (org.json) – Lightweight JSON handling

- org.eclipse.jgit – Java Git implementation for SSH/HTTP operations

- javafx-controls, javafx-media, javafx-fxml – For GUI and media components (optional use)

- org.jetbrains.annotations – Static code analysis annotations
- mslinks – Creating Windows .lnk shortcuts
- commons-imaging – For icon/image manipulation
- gson – JSON parser used in parts of Git integration
- maven-model – For parsing Maven POM files