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

Interface Modes

- **Simple UI**
  A minimal interface with a single **Install** button that automates the full pipeline:
  - Validates system requirements
  - Clones the selected Git repository
  - Builds the Maven project
  - Creates an OS-specific launcher shortcut
    This is ideal for users who want a one-click setup experience.
- **Advanced UI**
  Offers fine-grained control for developers who prefer step-by-step actions:
  - Customize Maven repository path
  - Choose specific branches
  - Manually trigger cloning, building, and installer creation
  - View README.md and project owner information
  - Export logs and import config files for repeatable builds

Users can switch between these UIs at any time using the **"Switch to Advanced UI"** or **"Switch to Basic UI"** button.

Full Process Flow

1. **System Requirements Check**
   When the app launches, it validates the presence and version of Java, Git, and

Maven. If a newer version is available, a download link is shown. The user must accept terms to proceed.

2. **Enter Repository URL**
   The user pastes the HTTPS or SSH link of a GitHub/GitLab repository. GitLab credentials can be entered if needed.
3. **Fetch Branches**
   Upon clicking "Fetch Branches", the available branches are retrieved using git ls-remote, and the dropdown menu is populated.
4. **Select Target Path & Optional Maven Repo**
   Users specify the install path where the repository will be cloned. Optionally, a custom Maven local repository path can be provided.
5. **Clone Repository**
   The repository is cloned. For GitLab, the app intelligently handles authentication. Once cloned:
   - The application name is extracted from the pom.xml
   - The README.md and project owner info are fetched and displayed
6. **Build Project**
   Using mvn install, the project is compiled. The application validates that the output .jar is created and stored in the target/ folder.
7. **Create Installer**
   The user sets an application name and optional icon file. The installer then generates platform-specific launch shortcuts:
   - **Windows**: .lnk file on Desktop and optionally in Start Menu (via mslinks)
   - **macOS**: .app bundle with optional Dock pinning (via AppleScript)
   - **Linux**: .desktop file or Flatpak manifest for app launcher
8. **Export & Import**
   - **Export Log**: The full console output can be saved for debugging or support.
   - **Import Config File**: Saved setup info can be restored to pre-fill fields in a new session.

4. Module & Class Responsibilities

The GitMaven Installer codebase follows a modular structure that separates concerns by functionality. Below is an overview of the main components and their responsibilities:

GUI

- **GitMavenCloneUI**
  Main advanced UI layout. Handles user interactions, layout rendering, and action routing for clone/build/install operations.
- **SimpleUI**
  Minimal interface for one-click install flow. Connects core logic with simplified

layout.

- **InitializeDefaults**
  Utility to initialize or reset UI fields and defaults.
- **UpdateBranchSelector**
  Dynamically updates the branch dropdown after fetching remote branches.

Requirements Check

- **RequirementsChecker**
  Entry point to system checks. Displays HTML-based results and manages license agreement flow.
- **CheckerHelper**
  Executes version checks and CLI detection for Java, Git, and Maven. Compares against latest known versions and formats output.

Git & Project Metadata

- **CloneRepository**
  Handles repository cloning (GitHub and GitLab). Manages GitLab token logic, triggers metadata fetching, and populates UI.
- **FetchGitBranches**
  Chooses the correct branch fetcher based on Git provider (GitHub/GitLab).
- **FetchGitHubBranches, FetchGitLabBranches**
  Uses git ls-remote or GitLab API to retrieve branches.
- **FetchGitOwnerInfo**
  Routes to provider-specific metadata classes.
- **FetchGitHubOwnerInfo, FetchGitLabOwnerInfo**
  Fetch and parse JSON metadata (owner login, email, API links).
- **FetchReadMeInfo**
  Loads and displays the content of the README.md file.

Maven Build Pipeline

- **RunMavenBuild**
  Triggers the Maven build using mvn install. Handles output and progress indication.
- **PomHelper**
  Detects the pom.xml file and extracts metadata such as the application name and main class.
- **JarFinder**
  Locates the built .jar file in the target/ directory after build.

Installer Builders

- **CreateInstaller**
  Orchestrates installer creation based on current platform and user inputs.
- **WindowsInstaller**
  Creates .lnk shortcuts using the mslinks library for Desktop and Start Menu.
- **MacInstaller**

Builds .app bundles using shell scripts and AppleScript. Optionally pins to Dock.

- **LinuxInstaller**
  Attempts Flatpak build; falls back to .desktop launcher creation.
- **CreateInstallerHelper**
  Provides shared helper logic for all platform-specific installer modules.

Helper Utilities

- **Constants**
  Centralized storage for UI text, filenames, or platform-specific defaults.
- **RepositoryHelper**
  Handles repository path parsing, project name extraction, and folder logic.
- **OSChecker**
  Detects if the system is Windows, macOS, or Linux.
- **ConsoleLogAppender**
  Outputs and optionally saves messages to file or console area in the GUI.
- **ConfigImporter**
  Loads saved configuration files to auto-fill fields.
- **Cleaner**
  Optional cleanup logic (temporary or build folders).

Entry Point

- **Main**
  Launches the system check and initializes the selected UI mode.

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