**UML Diagram Generator for Java Code – Requirement Document**

**1. Introduction**

**1.1 Purpose**

This tool aims to analyze Java source code and generate UML diagrams automatically. It enables software engineers, architects, and students to visualize Java code structure, relationships, and interactions efficiently.

**1.2 Objectives**

* Automate the creation of UML diagrams from Java source code.
* Support multiple diagram types, including **Class Diagrams, Sequence Diagrams, and Package Diagrams**.
* Provide export options for **PlantUML, Graphviz, XMI, and image formats (PNG, SVG, PDF, etc.)**.
* Offer an interactive **GUI/Web interface** along with **CLI support**.

**1.3 Target Users**

* **Software Architects & Developers** – To document and analyze software design.
* **QA Engineers** – To validate system architecture and dependencies.
* **Students & Educators** – For learning and teaching UML concepts.
* **DevOps & Security Teams** – To review architecture and security risks.

**2. Use Cases**

| **ID** | **Use Case Name** | **Description** |
| --- | --- | --- |
| UC-01 | Load Java Files | User provides Java files (single file or directory) for parsing. |
| UC-02 | Parse Java Code | Tool extracts class, methods, relationships, and dependencies. |
| UC-03 | Generate Class Diagram | Creates UML Class Diagrams showing classes, attributes, and methods. |
| UC-04 | Generate Sequence Diagram | Identifies method calls to generate sequence diagrams. |
| UC-05 | Generate Package Diagram | Shows dependencies between Java packages. |
| UC-06 | Export to PlantUML | Converts extracted data to PlantUML format. |
| UC-07 | Export to Graphviz | Outputs a .dot file for visualization in Graphviz. |
| UC-08 | Export to XMI | Generates an XMI file for integration with UML modeling tools. |
| UC-09 | Generate Image Files | Outputs UML diagrams as PNG, SVG, or PDF. |
| UC-10 | Provide CLI Interface | Allows command-line execution with arguments for input/output. |
| UC-11 | Provide GUI Interface | User-friendly UI for interactive diagram generation. |
| UC-12 | Provide Web Interface | Exposes API for web-based access and integration. |
| UC-13 | Analyze Dependency Graph | Detects dependencies between classes and modules. |
| UC-14 | Generate Metrics Report | Produces reports on code structure, complexity, and dependencies. |

**3. Functional Requirements**

**3.1 Core Functionalities**

✔ **Java Code Parsing**

* Analyze class definitions, interfaces, methods, fields, and annotations.
* Detect relationships: **Inheritance, Implementation, Association, Aggregation, Dependency, Composition**.
* Identify **static and dynamic method calls** for Sequence Diagrams.

✔ **UML Diagram Generation**

* Convert parsed Java code into UML notation.
* Support **Class Diagrams, Sequence Diagrams, and Package Diagrams**.
* Provide **editable PlantUML, Graphviz, and XMI outputs**.
* Support export as **PNG, SVG, PDF**.

✔ **Multiple Input Methods**

* Accept **single Java files, entire directories, or compressed projects (ZIP)**.
* Support **Gradle/Maven-based projects**.
* Detect and include **third-party dependencies** if required.

✔ **Output & Integration**

* Provide **CLI-based execution** with arguments for specifying **input and output formats**.
* Generate **interactive diagrams** in GUI/Web mode.
* Provide an **API for integration with other tools**.

✔ **Error Handling & Logging**

* Handle **syntax errors gracefully** and notify users.
* Provide **detailed logs for debugging and analysis**.

✔ **Performance Optimizations**

* Handle **large Java projects** efficiently.
* Optimize parsing using **multi-threading**.
* Implement **caching** for frequently accessed files.

**4. Non-Functional Requirements**

✅ **Performance**

* Must process **large Java projects (1000+ classes)** within reasonable time constraints.
* Use **parallel processing** where applicable.

✅ **Scalability**

* Should support **small, medium, and enterprise-level projects**.
* Web-based version should handle **multiple concurrent users**.

✅ **Extensibility**

* Allow **future support for additional languages (Kotlin, Scala, etc.)**.
* Support **plug-ins** for additional diagram types.

✅ **Usability**

* **GUI/Web interface should be intuitive** with drag-and-drop support.
* CLI must provide **clear documentation and examples**.

✅ **Platform Independence**

* Must run on **Windows, Mac, and Linux**.
* Should support **containerized deployment (Docker, Kubernetes)**.

✅ **Security**

* Prevent execution of **malicious Java code**.
* Implement **sandboxing mechanisms** for uploaded projects.

**5. Constraints & Assumptions**

🔹 **Constraints**

* **Requires Java 17+** for execution.
* **Limited to static code analysis** (dynamic method calls in Sequence Diagrams may not be detected).
* **No runtime monitoring** – tool works on source code, not compiled binaries.

🔹 **Assumptions**

* Users will provide **valid Java source code**.
* Users will specify **which diagrams they need**.

**6. Architectural Considerations**

**6.1 System Architecture**

* **Modular architecture** with separate components for **parsing, analysis, and output generation**.
* **Microservices-based approach** for web deployment.

**6.2 Technology Stack**

* **Backend:** Java 21, Spring Boot
* **Parsing Engine:** JavaParser, Eclipse JDT, or Spoon
* **Diagram Generation:** PlantUML, Graphviz, or custom rendering engine
* **CLI:** Apache Commons CLI
* **GUI:** JavaFX or Electron-based Web UI
* **Web API:** RESTful API with Spring Boot

**7. Potential Challenges**

| **Challenge** | **Mitigation Strategy** |
| --- | --- |
| Large projects take too long to process. | Implement multi-threading, caching, and lazy loading. |
| Complex Java features (e.g., generics, reflection) are hard to parse. | Use a robust parsing engine like JavaParser. |
| Sequence Diagrams require dynamic analysis. | Use **instrumentation-based approach** for dynamic call tracking. |
| Users require customization of UML output. | Provide a **configuration file** to control diagram generation. |

**8. Future Enhancements**

* **Support for additional languages** (Kotlin, Scala, Python).
* **Cloud-based version** for remote usage.
* **Integration with CI/CD pipelines** (e.g., GitHub Actions, Jenkins).
* **Live UML rendering** for real-time code visualization.