**AI-Assisted Legacy Migration Agent Suite**

**Current Progress**

You have successfully built the GitHub Repo Analyzer Agent. This agent:

- Automatically triggers on GitHub repo pages

- Summarizes the repository purpose using an LLM

- Extracts and categorizes file structure

- Provides a styled and interactive UI with copy/download features

**Next Agent: Architecture Inference Agent**

Purpose: Infer high-level architecture of the repo.

Input:

- Output from Repo Analyzer Agent

- Raw code/config files (optional)

Output:

- Architecture type (e.g., monolith, layered, microservice)

- Stack used (e.g., Java + Spring Boot)

- Layers/components inferred

- Entry points (e.g., /login, /register)

Sample Output:

{

'architecture': 'Spring Boot monolith',

'layers': ['Controller', 'Service', 'Repository'],

'tech\_stack': ['Java', 'Spring Boot', 'JPA']

}

**Future Agents (Roadmap)**

**3. Code Flow Agent**

**Purpose:**  
Understand the logical flow of the application — from user input to system output.

**Input:**

* Categorized source code files (from the GitHub Parser Agent).
* Specific entry point file (e.g., main.java, app.py, etc.).

**Output:**

* Step-by-step trace of how data flows through the codebase.
* Key classes, methods, and their interaction sequence.

**Sample:**

yaml

CopyEdit

Input: GET /user/profile

→ Router → AuthMiddleware → UserController → UserService → UserRepository → DB

**Use Cases:**

* Identify tightly coupled logic.
* Understand dependencies and flow for migration planning.

**4. Risk Assessment Agent**

**Purpose:**  
Detect potentially risky or fragile parts of the code that may break during modernization.

**Input:**

* Parsed repo structure
* Dependency and usage metrics (e.g., frequency of file edits, LOC, cyclomatic complexity).

**Output:**

* Heatmap-style risk report highlighting:
  + Outdated libraries
  + Large or complex files
  + Deprecated APIs

**Sample:**

json

CopyEdit

{

"high\_risk": [

"UserService.java",

"AppConfig.java",

"pom.xml (uses log4j 1.x)"

],

"medium\_risk": ["AuthController.java"],

"low\_risk": ["README.md"]

}

**Use Cases:**

* Prioritize parts of the system for refactoring.
* Flag migration blockers early.

**5. Migration Strategy Generator**

**Purpose:**  
Generate an initial draft of how to split or refactor the monolithic application.

**Input:**

* Code flow mapping
* Risk report
* Business context tags (optional)

**Output:**

* Microservice candidates
* Suggested API boundaries
* Communication method (REST/gRPC/events)
* Potential deployment suggestions (container, serverless, etc.)

**Sample:**

yaml

CopyEdit

Microservices:

- UserService

- AuthService

- NotificationService

Boundaries:

- Extract NotificationService from UserService (low coupling)

- Separate AuthService due to high call frequency

Recommended:

- Deploy using Docker + K8s

- Communication via REST

**6. Stack Upgrade Agent**

**Purpose:**  
Modernize the tech stack and suggest newer alternatives.

**Input:**

* Detected dependencies from pom.xml, package.json, etc.

**Output:**

* Table of outdated libraries/frameworks
* Recommended upgrades
* Compatibility notes

**Sample:**

markdown

CopyEdit

| Current | Latest | Recommendation |

|---------|--------|----------------|

| Spring 4.3 | Spring Boot 3.x | Major upgrade required |

| Log4j 1.2 | Logback | Security risk: must upgrade |

| Java 8 | Java 17 | Performance & security |

**7. Human Feedback Integrator**

**Purpose:**  
Allow humans to interactively review, edit, and approve each AI-generated suggestion.

**Input:**

* Suggestions from previous agents
* Admin/editor responses

**Output:**

* JSON/Markdown record of approved, rejected, or modified items
* Tracked rationale (optional)

**Sample:**

json

CopyEdit

{

"AuthService Split": "approved",

"Upgrade Java 8 to 17": "approved",

"Move all controllers to API Gateway": "rejected",

"Extract PDFService": "modified → Merge into DocumentService"

}

**Use Cases:**

* Enable collaboration with domain experts.
* Ensure AI proposals align with business needs.

**8. Blueprint Generator**

**Purpose:**  
Generate the final migration blueprint as a structured, exportable document.

**Input:**

* Aggregated and finalized outputs from all other agents
* Human-reviewed feedback

**Output:**

* Downloadable .md or .pdf file
* Clear migration plan with steps, diagrams (optional), service breakdowns

**Sample TOC (Markdown/PDF):**

markdown

CopyEdit

# Legacy Modernization Blueprint

1. Executive Summary

2. Codebase Analysis

3. Identified Risks

4. Suggested Microservices

5. Stack Upgrade Plan

6. Migration Phases

7. Approval Log

8. Appendix (File structure, flow diagrams, etc.)

**Agent Integration Vision**

Each agent acts in a pipeline:

- GitHub Parser -> Architecture Agent -> Code Flow Agent -> ...

- Human can inspect/edit results at each step

- Final product: a complete, semi-automated legacy migration blueprint