

Repository Architecture Diagramming



Statement of Purpose:

This project aims to turn code into clarity by automating architecture visualization, making complex systems understandable and collaboration effortless.



USER PERSONAS

Lily Chen

Beginning Software Developer

21, First-year CS student, San Diego

Needs and motivations

- Understand real codebases visually instead of reading dense source files
- Explore open-source projects for learning and inspiration

Pain Points

- READMEs are text-heavy and hard to follow
- Hard to see how modules and folders are connected

How We Can Help

Our tool automatically visualizes a repo's internal logic and structure, turning messy code into an intuitive diagram that helps Lily learn by seeing.

Eric Wang

Newly Onboarded Software Engineer

25, Junior software engineer, Seattle

Needs and motivations

- Quickly grasp core project logic and architecture
- Locate where assigned features are in the codebase

Pain Points

- Sparse onboarding documents
- Hard to link assigned tasks to actual code components

How We Can Help

By generating architecture diagrams and supporting task-focused prompts, our tool gives Eric a map of the codebase from day one.

Cassie Hang

Computer Science Researcher

31, University researcher, Minneapolis

Needs and motivations

- Analyze how structure and visualization affect codebase comprehension
- Study architecture complexity across branches and commits

Pain Points

- Manual diagram creation is slow and inconsistent
- Hard to align research metric across codebase evolution

How We Can Help

Our automated visualization engine provides consistent, analyzable mappings, helping Cassie transform code evolution into measurable insights.

MORE USER PERSONAS

Tyler Roberts
Product Manager

25, Product Manager, Los Angeles

Needs and motivations

- Visualize the architecture of a feature directly from the source code
- Ensure the user experience aligns with system capabilities and constraints

Pain Points

- Hard to explain different modules interact without relying solely on engineers' explanations
- Hard to visualize how backend components connect to the UI or API endpoints

How We Can Help

Our instant generated diagrams provides analyzable diagrams, helping help Tyler plan and run his meetings and his sprint reviews.

Blake Li
Software Team Lead

30, Senior software engineer, Irvine

Needs and motivations

- Visualize team codebase system architecture
- Track system architecture diagrams that evolve across sprints

Pain Points

- Hard to detect emergent technical debt or architectural drift early
- Difficult to confirm whether implementation matches the original design

How We Can Help

Our generated architecture diagrams provide Blake a mapping of the entire codebase to visualize and track the system architecture.

Freddie Reynolds
QA and Test engineer

27, QA engineer, Minneapolis

Needs and motivations

- Analyze development flow and identify potential risks to build better tests
- Visualize the branch structure and commits of a repo

Pain Points

- Difficult to track branch structure and commits of a repo
- Time-consuming to understand breakdown of each component

How We Can Help

Our automated visualization engine provides consistent, analyzable mappings, helping Freddie design better test cases to ensure the codebase implements the system architecture and design .

TOPICS OF RISK AND POTENTIAL CONCERNS

01

Essential Complexity

1. Cross-Language Parsing (Syntactic and Semantic nuances)
2. Repository Scale → Memory and Performance bottlenecks

02

Cognitive Complexity

1. Role Diversity → Varied Abstraction Levels
2. Large Diagrams → Reduced Visual Comprehension

03

Temporal Complexity

1. Version evolution → Repositories evolve continuously

04

Structural Complexity

1. Scale of Interconnections → Dependencies multiply rapidly with repository size
2. Large Diagrams → Reduced Visual Comprehension

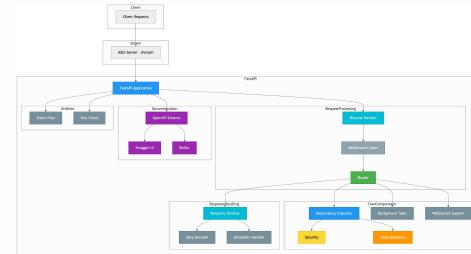


PREVIOUS WORK

Before starting on this project, we found some existing software related to this topic. Although they offer impressive functionality, they still have some inherent limitations.

GitDiagram

Instant repo-structure diagrams — closest competitor; validates demand but lacks deeper semantic + version insights.



DeepWiki

AI-generated repo documentation + structure views — inspires automated understanding but limited architectural depth.

[Deep Dive](#) [Microservices](#) [Service Mesh](#)

Last updated: 2020-08-06

[View on GitHub](#) [Report an issue](#) [Search](#) [Help](#) [About](#)

[Code Architecture Overview](#)

[Application Structure and Process Architecture](#)

[Deployment](#)

[Build System and Pipeline](#)

[Testing](#)

[Health Model and Configuration](#)

[Microservices](#)

[External API and API](#)

[License and Agreement](#)

[Bugs or Issues](#)

[Code Examples](#)

[Microservices Examples](#)

[Microservices Integration](#)

[Layout System and Global Styles](#)

[Components](#)

[Atomic Models](#)

[Tree and List Components](#)

[Integrated System](#)

[Form and Editor Components](#)

[Chart and Diagram](#)

[Chart Widgets and Session](#)

[Purpose and Scope](#)

This document provides an overview of the Code Architecture, focusing on the code process design, core patterns, and major components involved. It explains the evolution between the initial process, modern process, evolution path, and the specified process, as well as the key concepts and components involved in each stage and throughout the evolution.

For detailed information about specific subtopics:

- Application runtime and interaction with Application Structure and Process Architecture
- Build system details: see [Build System and Pipeline](#)
- Deployment details: see [Deployment](#)
- External architecture: see [Microservices](#)
- Modeling and design: see [Application Structure and Process Architecture](#)
- Monitoring and observability: see [Metrics and Log Infrastructure](#)

Multi-Process Architecture

12 Factor Code is a multi-process system application with three separation of concerns for security, stability, and extensibility.

```

graph TD
    AC[Application Container] --- Persistence[Persistence]
    AC --- Configuration[Configuration]
    AC --- Network[Network]
    subgraph NP [Processors]
        Persistence
        Configuration
        Network
    end
    Persistence --- PersistenceProcessor[Persistence Processor]
    Configuration --- ConfigurationProcessor[Configuration Processor]
    Network --- NetworkProcessor[Network Processor]
    PersistenceProcessor --- AC
    ConfigurationProcessor --- AC
    NetworkProcessor --- AC

```

The Application Container interacts with the Persistence, Configuration, and Network Processors. Each processor has its own specific processor.

[View on GitHub](#) [Report an issue](#) [Search](#) [Help](#) [About](#)

[Feedback](#) [File a bug](#) [Suggest a feature](#) [Ask a question](#)

[On this page](#)

[12 Factor Code Architecture Overview](#)

[Purpose and Scope](#)

[Build System and Pipeline](#)

[Deployment](#)

[Microservices](#)

[Core Microservice Architecture](#)

[Domain Model](#)

[External API](#)

[Metrics and Log Infrastructure](#)

[Processors](#)

[Data Sources](#)

[Data Services](#)

[Data Service Interactions](#)

[Editor and Message Passing](#)

[Extensibility](#)

[Global State](#)

[Build System and Packaging](#)

[Key Relational Architectures](#)

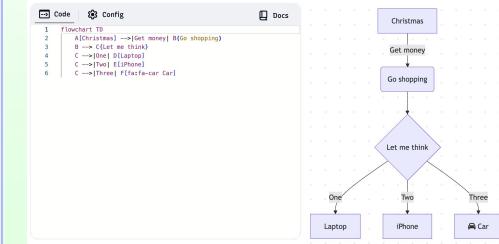
[Non-relational Architectures](#)

[Service Registration Events](#)

[Pattern Summary](#)

MermaidJS

Text-to-diagram rendering — useful
formatting backend, but requires
manual authoring, not automatic
extraction.



awesome-diagramming

A curated index of visualization tools — shows the ecosystem but doesn't analyze or auto-diagram code.

FEATURES AND PRIORITIES

Feature	Core	Medium Priority	Exploratory
Input & Access	<ul style="list-style-type: none">Import via GitHub linksZip file upload: Allow users to upload source ZIPs as an alternative to GitHub links	<ul style="list-style-type: none">Share to tools like slack, jira, githubPrivate Access Tokens: Allow users to analyze private repositories.	
Repository Insights	<ul style="list-style-type: none">List all components and latest commit author/date	<ul style="list-style-type: none">Repo metrics (file count, language, folder depth)Provide a brief summary of what each component contains	<ul style="list-style-type: none">Coding language analysis for the repo
Visualization	<ul style="list-style-type: none">Generate an architecture diagram for the repo, able to select specific commit/branch	<ul style="list-style-type: none">Dropdown to select branches from repo to displayNavtools to move to parts of the codebase in the diagram quickly	<ul style="list-style-type: none">Diagram change history (diff view)Export Mermaid JS codeBuilt-in option to copy/edit Mermaid JS code
Usability & Personalization	<ul style="list-style-type: none">Export diagram as PNG/SVG or embedded linksCached repo history: Save previously analyzed repositories.	<ul style="list-style-type: none">Construct in-depth documentation based on the diagram	<ul style="list-style-type: none">Annotate diagrams with notes and highlightsUser-type based specialized analysis

WIREFRAMES

SYSTEM DESIGN

SWE Diagramming

Enter Github Repo URL OR Upload Zip file Generate Diagram History

Repository: [Repository Name] Branch: [Branch Name] Export as Image ↴

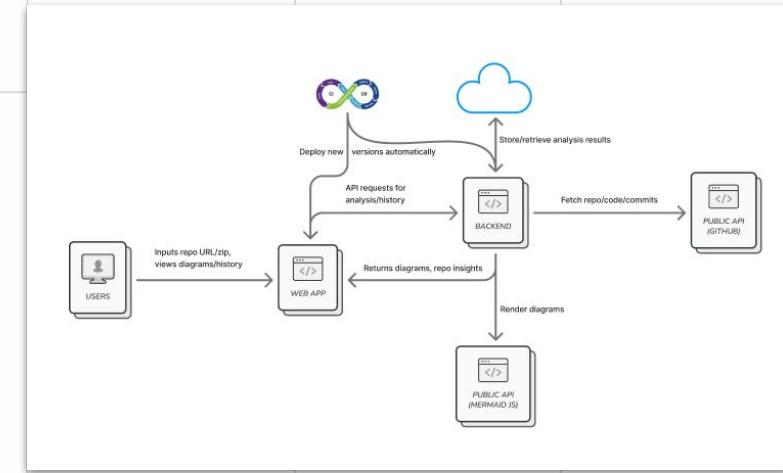
File Structure

```
tree
  i18n
    messages.properties
    messages_da.properties
    messages_de.properties
    messages_es.properties
    messages_fr.properties
    messages_it.properties
    messages_ja.properties
    messages_nl.properties
    messages_pt_BR.properties
    messages_pt_PT.properties
    messages_ru.properties
    messages_th.properties
    messages_zh_CN.properties
  services
  config
    TestTagLib.groovy
    com
      shri
        webalbum
        tablib
  util
    com
      shri
        webalbum
        util
          PasswordCodec.groovy
```

SWE Diagram

```
graph TD
    Start([START Input Product Request]) --> Fetch[Fetch inventory for product from DynamoDB]
    Fetch --> Decision{Items remaining?}
    Decision -- Yes --> Select[Select next available inventory item]
    Select --> Mark[Mark inventory item as Sold in DynamoDB]
    Mark --> Decision
    Decision -- No --> Error[Send No Items Remaining Error]
    Error --> End([END])
    Select -- Transaction fails --> Decision
    Mark -- Transaction fails --> Decision
    Select -- Transaction Success! --> Publish[Publish PRODUCT_PURCHASED event]
    Mark -- Transaction Success! --> Publish
    Publish --> End
```

Took X seconds to generate



THANK YOU

