Video Link:

https://www.youtube.com/watch?v=qsWvrsTMn3A

VOID IDE

CONCRETE ARCHITECTURE

Presented By: Emily Cheng, Jeff Hong

GROUP MEMBERS

- Jinpeng Deng 22SSl17@queensu.ca – (Group Leader) Abstract, Introduction, Overview, Reference Check.
- Jeff Hong 22RQ20@queensu.ca – (Presenter) Grammar Review, Definitions, Conclusion, Lessons Learnt, AI Report, slides.
- Emily Cheng 22TWS1@queensu.ca – (Presenter) Overall+Essential Use Cases, Sequence Diagrams, Box-and-Arrow diagrams, editing, slides.
- Zijie Gan 21ZG6@queensu.ca – Subsystem-Level Reflexion of Components, Conceptual Architecture.
- Nathan Daneliak 22TMX2@queensu.ca – Understand, Concrete Architecture, Key Components, Derivation Process, Detailed Inner Subsystem.
- Zipeng Chen 23QHl0@queensu.ca – High-Level Subcomponents, Interactions, Subsystem Dependency, Evolution.

01 CONCEPTUAL ARCHITECTURE BREAKDOWN

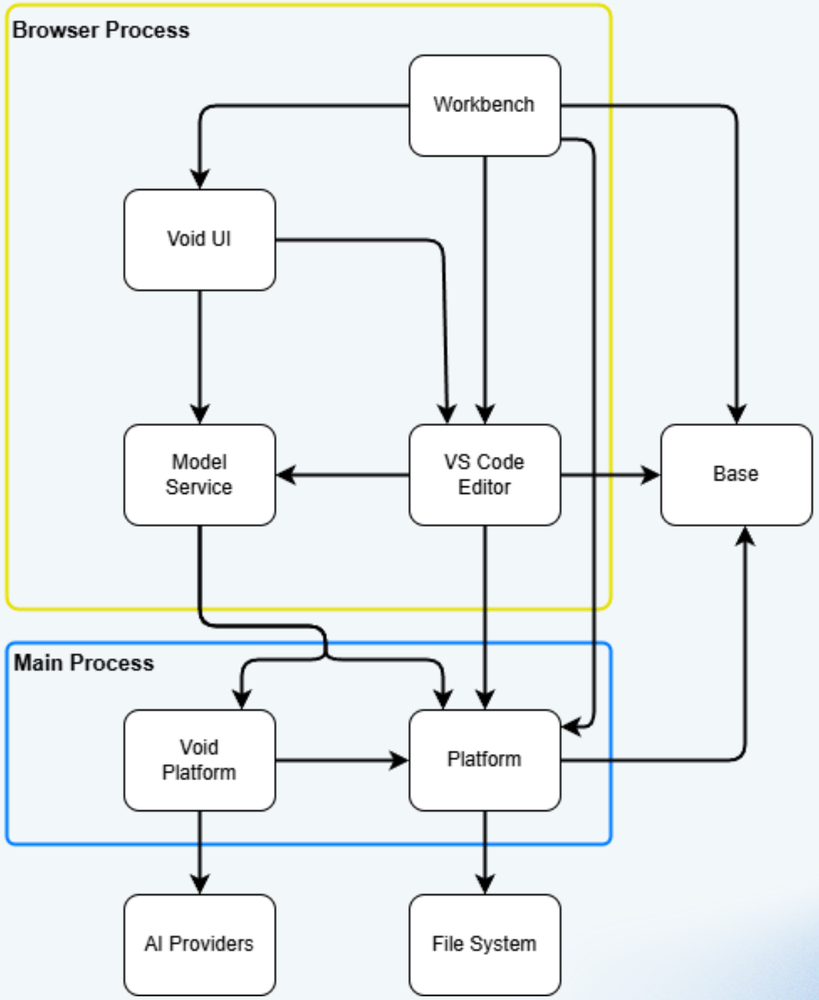
RATIONALE + COMPONENTS

Core Components

- Void UI (Browser): Manages chat, quick edits, and user interactions.
- VS Code Editor & Workbench: Host Monaco, coordinate extensions and layout.
- Model Service: Orchestrates prompts, maintains context, selects AI provider. Bridges interface and backend.
- Void Platform (Main): Connects browser ↔ AI Providers through IPC, handles formatting and authentication.

Supporting Components

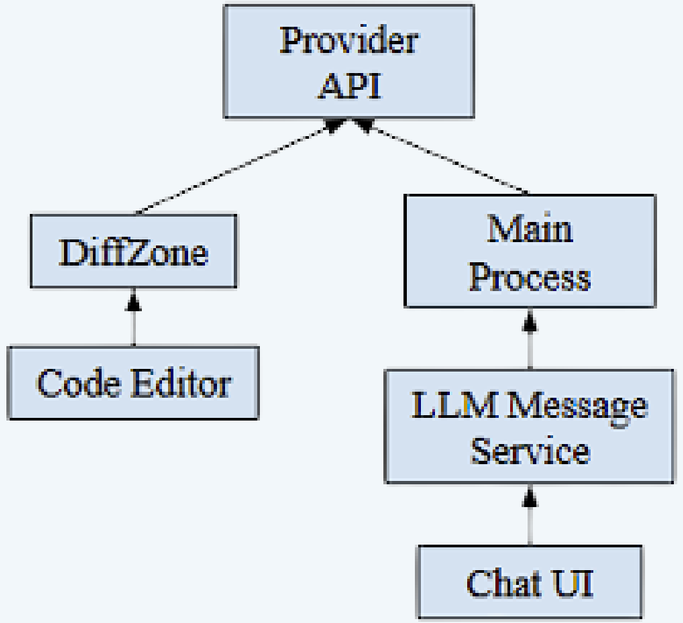
- Indexer & Repository: Tracks syntax and symbol updates to keep prompts accurate.
- Validation & Feedback Handling: Filter results and record feedback.
- File System: Persistence and data exchange, forms basis of file I/O.
- AI Providers: Links Void to local or remote LLM models through adapters.



02

CONCEPTUAL ARCHITECTURE BREAKDOWN (CONT.)

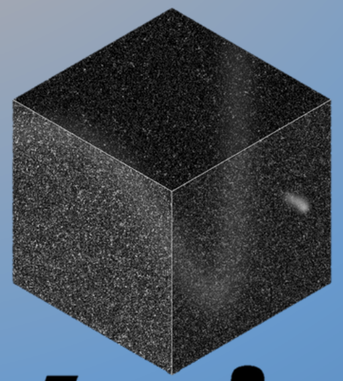
DIVERGENCE + REVISION NOTES



Assignment 1 ^

Updated from last assignment to build on Group 22's top-level model.

- Shifts from simplified layered → realistic, process driven design.
- Reflects how components actually exchange data and control.
- Captures cross-process interactions instead of abstracting them away in concrete design.



Void

03

ARCHITECTURE DERIVATION

OTHER GROUPS

Inspiration from the conceptual architecture of other groups provided an idea of what to look for in the source code.

SEPARATING CONCEPTUAL

Separated the components from the conceptual architecture into nodes tied to repository folders.

VOID REPOSITORY

Investigated the source code among the various Void repository files.

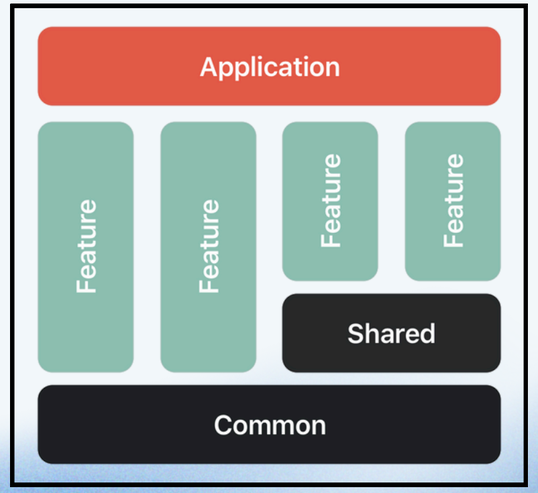
UNDERSTAND

Analyzed the connections between components and subsystems with the Understand software.

04

ALTERNATIVE ARCHITECTURE

Void's concrete architecture takes elements from the **layered architecture** style like in the conceptual architecture. However, the concrete architecture reveals it also has elements from the **modular architecture** style.



01 **Why Modular?**
Many VS Code components have a bidirectional dependence on each other.

02 **Arguments against Modular:**
The newly added Void components have a hierarchy based dependency structure.

03 **Rejecting Modular Architecture:**
Overall, the formatting and style of the newly added Void components are more important than the VS Code base.

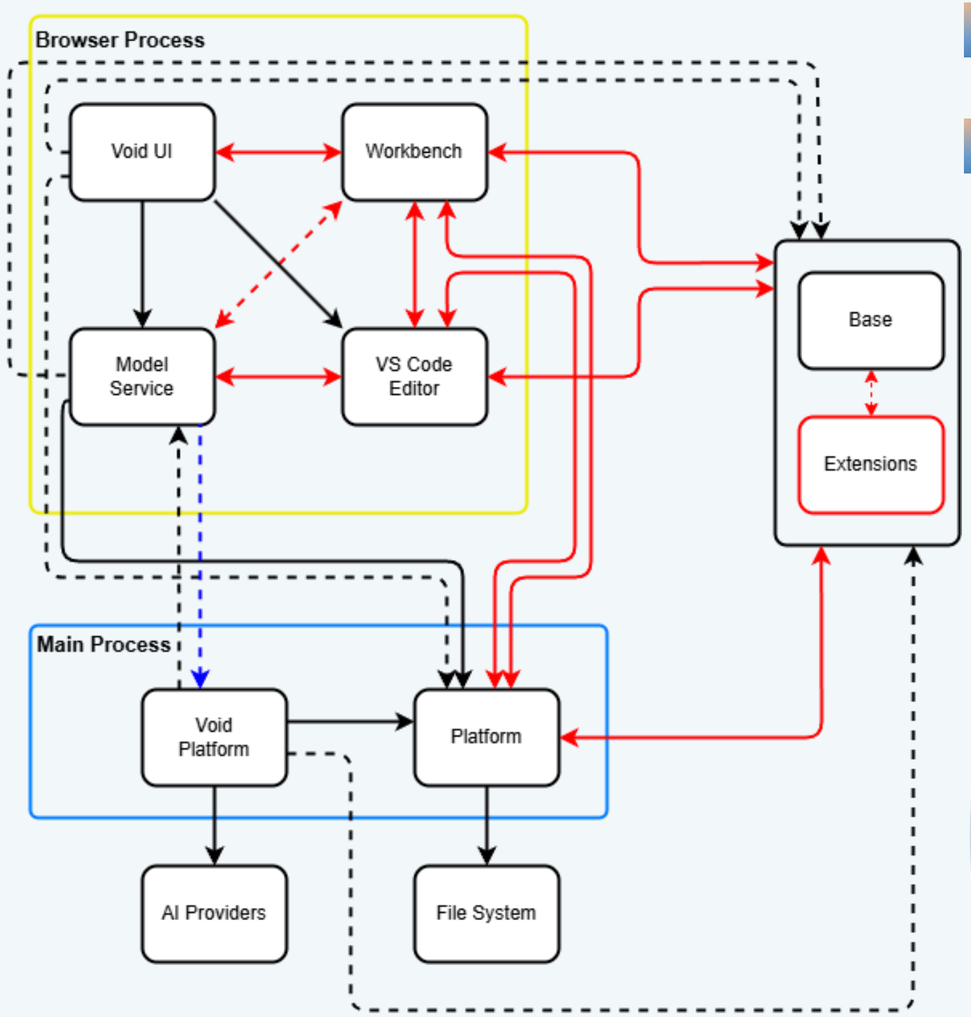
05 CONCRETE ARCHITECTURE

Logical decomposition → Components

Understand tool and directory mapping identifies subsystems that form the logical decomposition of Void.

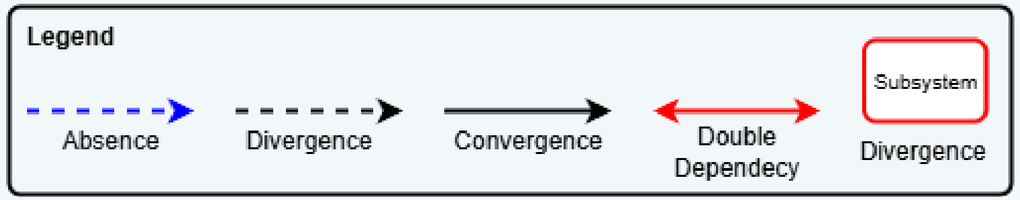
1. Workbench: Manages UI and logic in the browser process; shows minor upward coupling from debugging/terminal helpers.
2. VS Code Editor: Hosts Monaco core, depends on Model Service for AI-assisted editing.
3. Platform: Provides configuration, IPC services, and customization in the main process.
4. Base: Utility layer offering shared functionality across subsystems.
5. Extensions: Newly identified built-in layer (e.g. GitHub integration).
6. File System: Handles persistence + OS access, connects to the Platform via IPC.
7. Void UI: Implements chat, quick edit, and sidebar panels; depends on Workbench as well as Model Service.
8. Model Service & Void Platform: Form the AI pipeline: Model Service in browser, Platform in main process handling provider I/O.

CONCRETE ARCHITECTURE



REFLEXION + DIVERGENCES

- **Cross-Process AI Service:** Split between Model Service and Void Platform, diverging from the single-layered model.
- **Added Repository & Validation:** Confirmed as active runtime modules improving accuracy + feedback.
- **Bidirectional Dependencies:** Unexpected relations among Workbench, Editor, and Model Service due to shared APIs and two-way event subscriptions (verified in Understand).



Void's real architecture preserves its layered base but evolves into a modular, cross-process design optimized for reliability and maintainability.

07

2ND LEVEL SUBSYSTEM

CONCEPTUAL + CONCRETE ARCHITECTURE: **MODEL SERVICE**

A

Conceptual View
One cohesive orchestration unit managing prompts, conversation state, and provider communication.

B

Design Role
Acts between the interface and backend to format user actions into LLM requests.

C

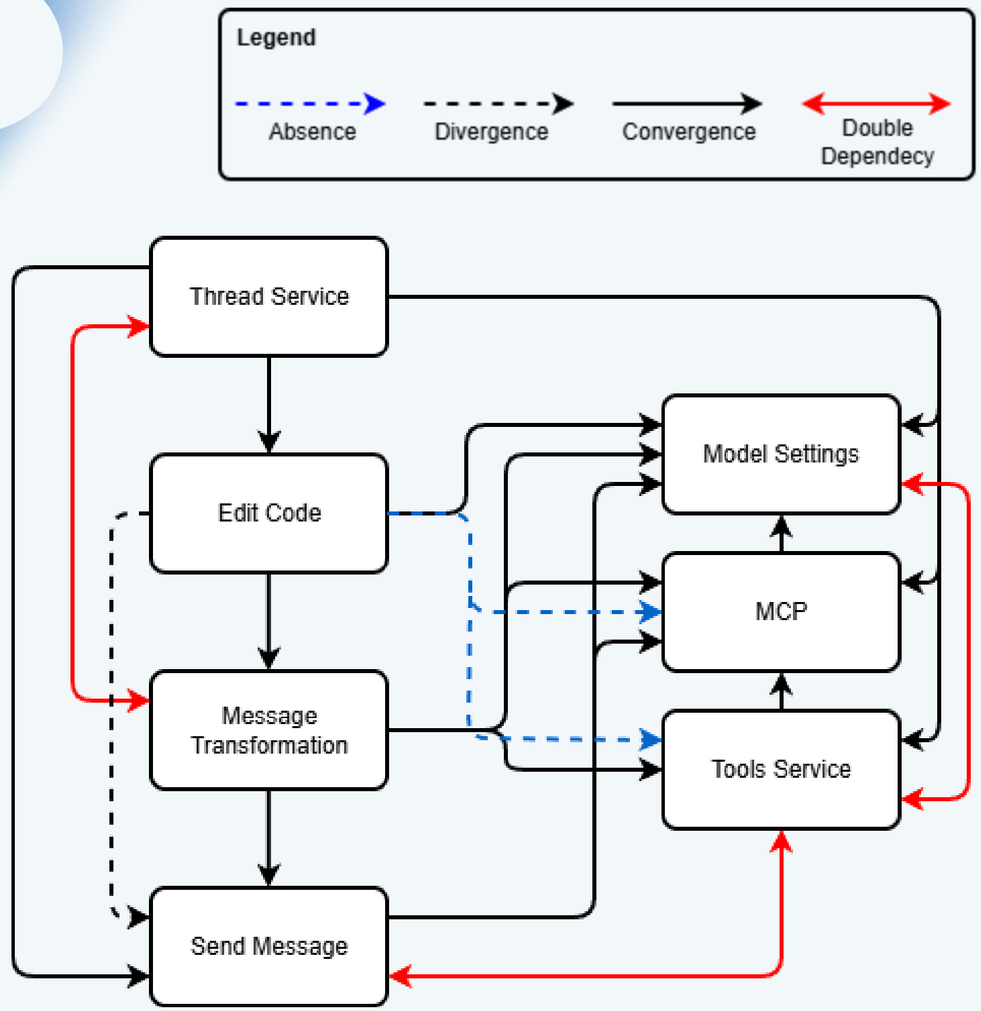
Initial Assumption
Responsibilities handled within a single module (no clear subcomponent boundaries).

Concrete implementation reveals a modular structure: this single conceptual unit decomposes into several cooperating sub-components.

ThreadService
Manages conversations and agent-mode tasks, keeping conversation history and context synchronized.

EditCode/MessageTransformation
Applies edits and converts chat or code requests into model-specific formats.

SendMessage/ModelSettings/MCP
Sends requests, manages model configuration, and integrates external tools.



01 **Unexpected Relation/Dependency**
ThreadService ↔ MessageTransformation reference cyclically. Adds internal complexity (compared to conceptual one-way flow).

02 **Absences (Missing Connection)**
Manages provider configurations and model settings. Acts as an internal dependency supporting all other core services.

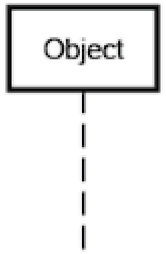
03 **Rationale**
Conceptual model assumed linear, top-down data flow. Concrete is asynchronous/message-driven with runtime feedback loops.

04 **Takeaway**
Model Service is a network of linked modules spanning browser and main processes. Greater responsiveness, adaptivity, maintainability.

MODEL SERVICE + REFLEXION ANALYSIS DIVERGENCES

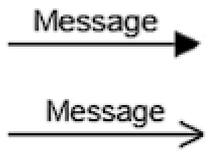
09 SEQUENCE DIAGRAM LEGEND

Actor: The user (developer) who interacts with the Void IDE system and thus its objects, through the interface.



Object: Represents a component or class of Void's concrete architecture.

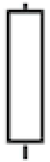
Lifeline: Vertical dashed line from each box, represents existence and activity over the passage of time as it goes downwards.



Message: Interaction or communication between objects, represents a method call, signal, data exchange between components.

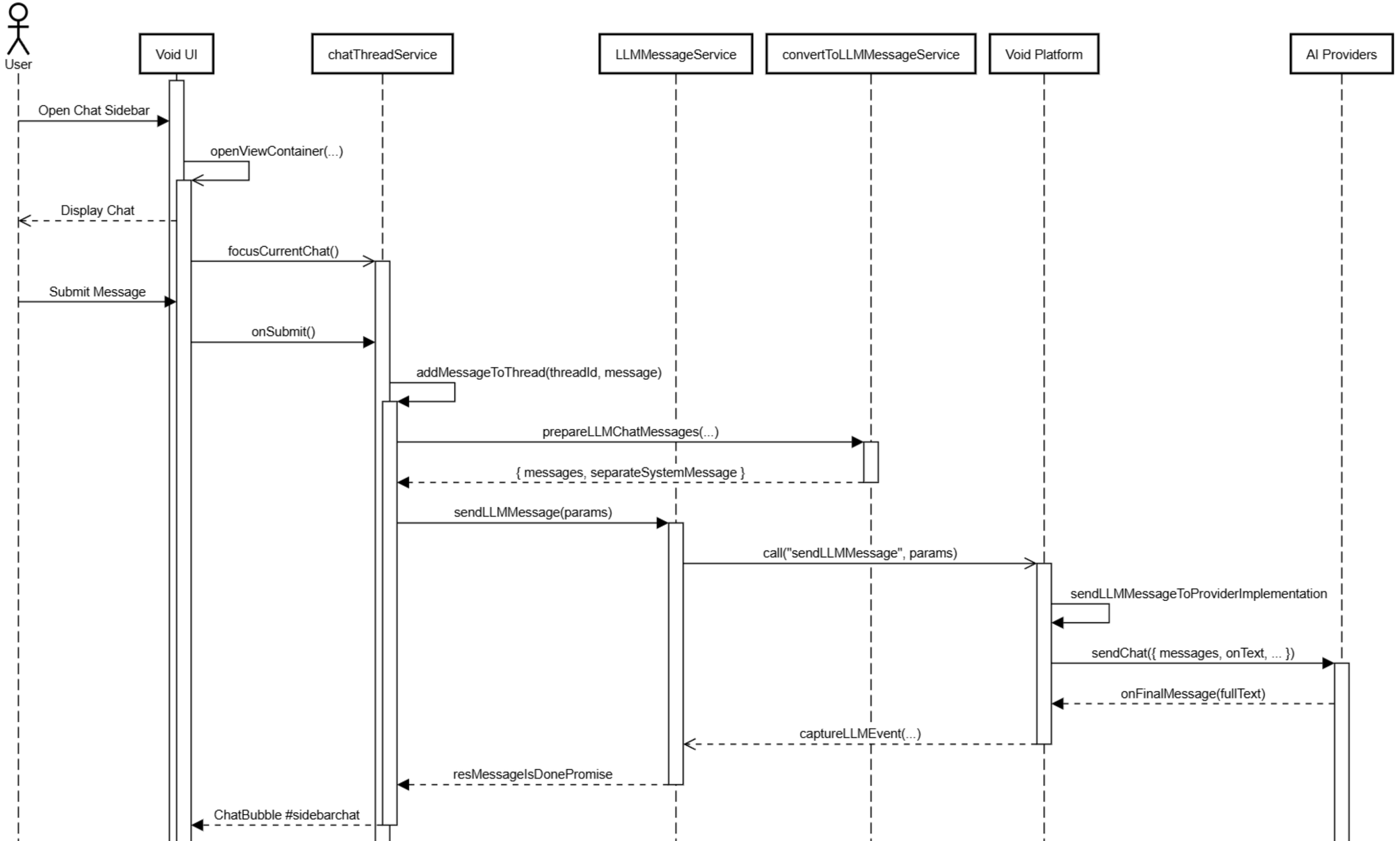
- **Synchronous:** Normal method call where the sender waits for a response.
- **Asynchronous:** For background tasks or events, the sender doesn't wait.

Reply: A return message, response sent from the receiver of a message when it has finished processing.

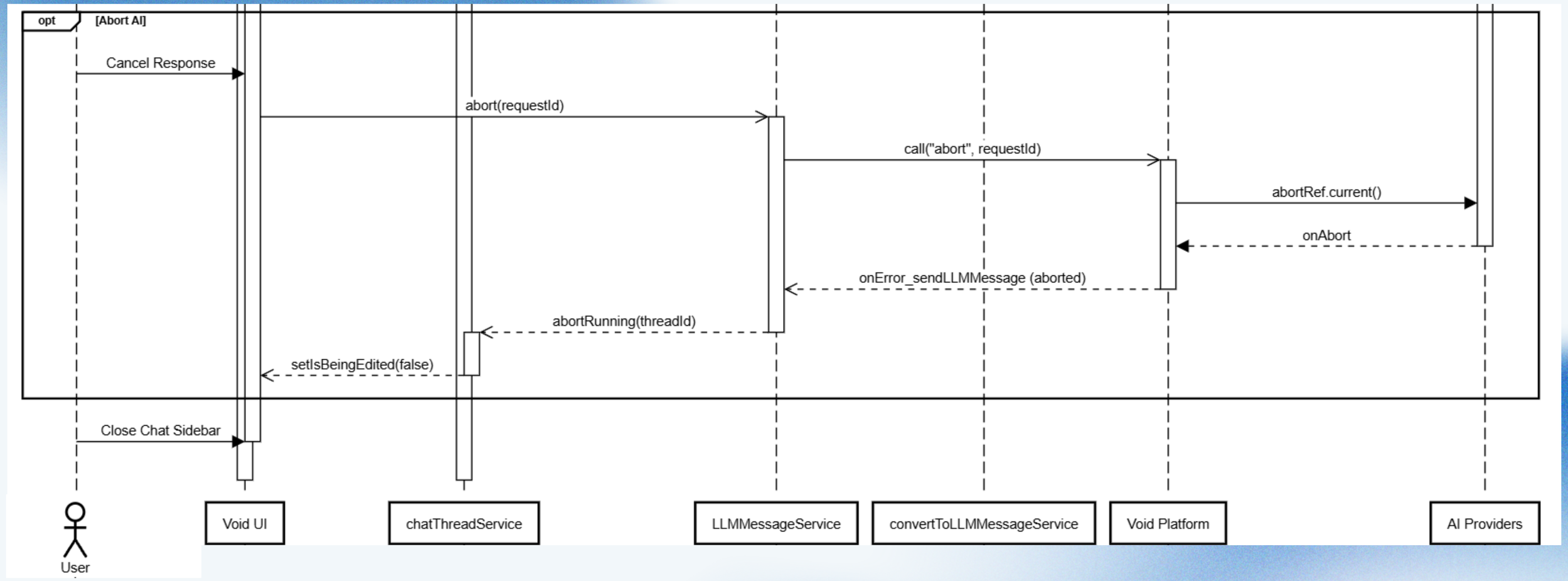


Activation Box: Rectangular box along an object's timeline that represents the time it takes for a task to be completed.

10 SEQUENCE DIAGRAM



SEQUENCE DIAGRAM (CONT.)



BOX-AND-ARROW DIAGRAM

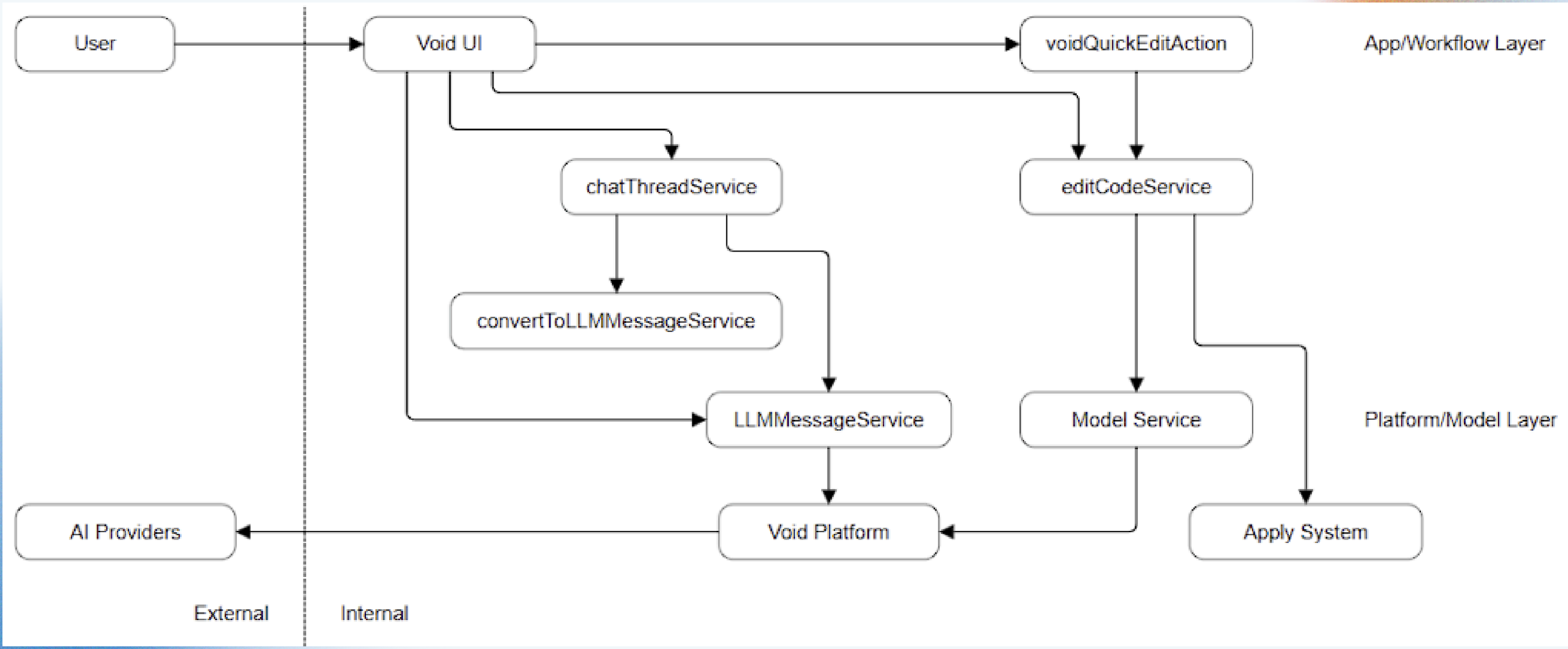
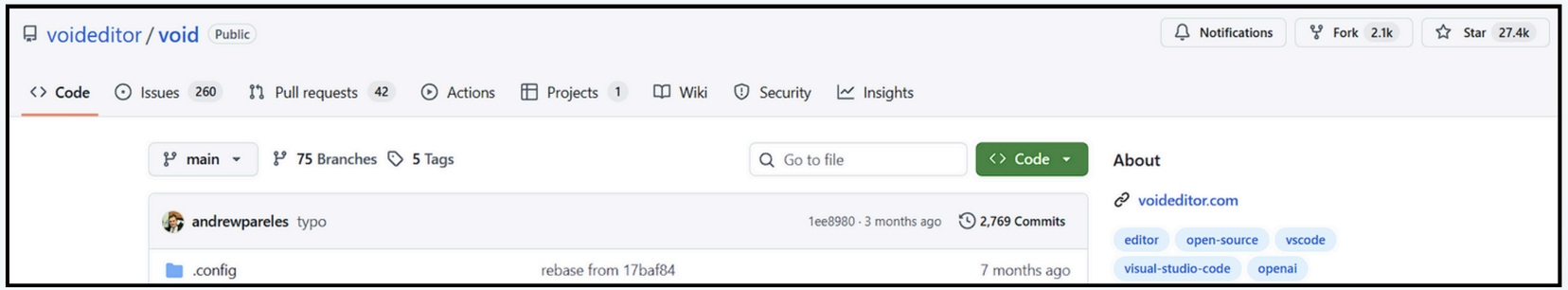


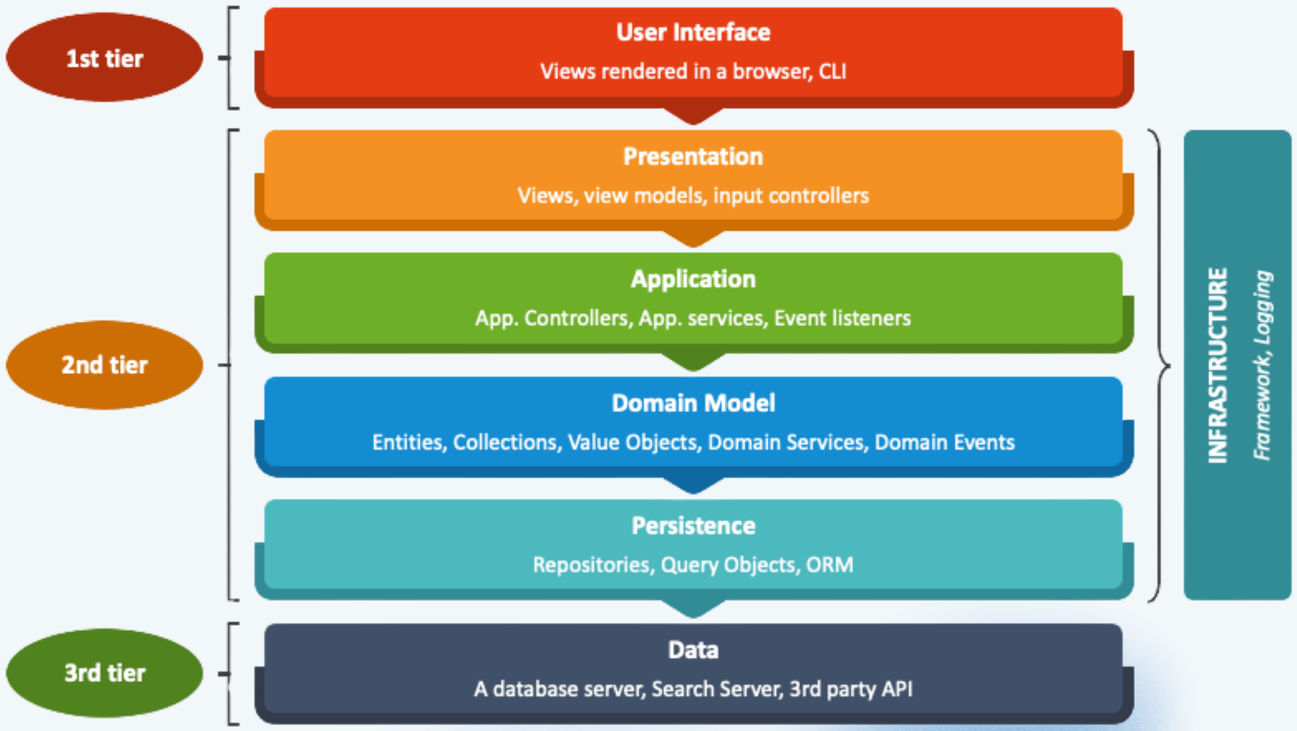
DIAGRAM METHODOLOGY

TRACED CONCRETE METHOD CALLS FROM VOID'S GITHUB CODEBASE /SRC/VS/WORKBENCH/CONTRIB/VOID/*

- Analyzed TypeScript files such as sidebarActions.ts and chatThreadService.ts to identify synchronous and asynchronous flows.
- Used Understand to visualize dependencies and confirm subsystem relationships.
- Cross referenced with **official Void documentation** on LLM Message Pipelines and architecture.
- Ensured sequence → box-and-arrow diagrams accurately matched real implementation behavior.



EFFECTS OF CONCURRENCY AND PROJECT ARCHITECTURE



AI Pipeline Functionality:
Concurrently processes and displays tokens returned from the AI while the AI continues to process

Import Structures in the Source Code:
The imports come from inner layers of the code influenced by the layered architecture

Increased Dependencies on Outer Layers:
Since the layers depend on each other going outwards, outer layers need more dependencies

LIMITATIONS IN FINDINGS

CANNOT CONTACT OTHER DEVELOPERS

Should we need any clarification regarding the code, we cannot get an answer.

VASTNESS OF VOID'S CODE REPOSITORY

Due to the many files in Void's source code, some details may be missed or misunderstood.

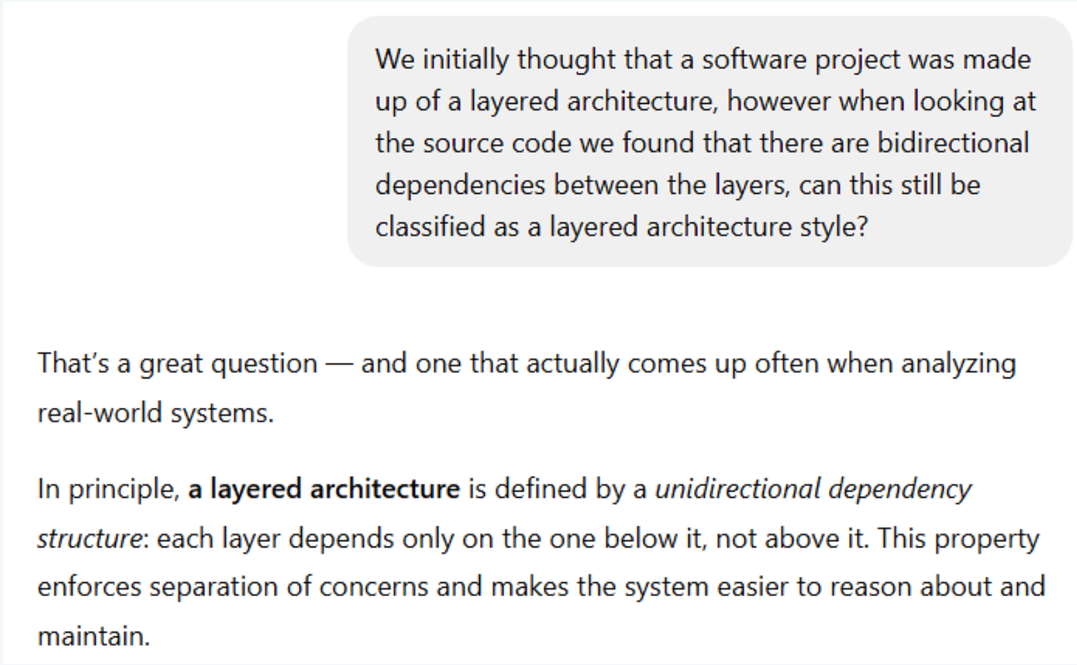
INSPIRATION FROM OTHER GROUPS

Since we took inspiration from other groups, any mistakes they made could have affected the baseline for our research.

AI REPORT

16

Group No. 7 | The Trailblazers | 2025



ChatGPT-5, August, 2025 Model Chosen:

- Satisfied with its performance in the first assignment.
- Claims to be special trained for coding and code related tasks.

Tasks Given:

- Grammar checking and review.
- Tips on organizing and drawing diagrams.
- Clarifying questions about architecture.

Prompting Strategy:

- Group members individually log their chats with the AI in a shared group chat.
- AI chats are double checked and fact checked after the initial report draft.

Impact:

- Helped streamline the workflow process.
- Let us think of more options for the flow of the text.

17

LESSONS LEARNED

01 The layers of a software with layered architecture may not always be so clear cut.

02 Visualization and communication of findings is crucial when analyzing such a large code base.

03 Void’s incremental processing of AI tokens allows for quicker responses to create a better feel for the user.

THANK YOU FOR LISTENING!

REFERENCES

A. Kumar, “Void IDE: The Comprehensive Guide to the Open-Source Cursor Alternative,” Medium, Mar. 24, 2025.

A. Perkaz, “Advanced Electron.js architecture,” LogRocket Blog, Oct. 5, 2023.

Electron, “Electron Documentation,” Electron.

Microsoft, “Visual Studio Code Extension API,” Visual Studio Code.

P. Powell and I. Smalley, “What is monolithic architecture?,” IBM Think.

Void Editor, “Void Official Website,” Void Editor.

Zread.ai, “Architecture Overview | Voideditor/Void,” Zread, Jun. 25, 2025.

Microsoft, “Source Code Organization,” GitHub, Oct. 11, 2025.

“Introduction - Model Context Protocol,” Modelcontextprotocol.io, 2025.

Zread.ai, “Apply System (Fast vs Slow) | Voideditor/Void,” Zread, Jun. 25, 2025.

Zread.ai, “LLM Message Pipeline Architecture | Voideditor/Void,” Zread, Jun. 25, 2025.