> GEMINI

https://youtu.be/9squ1uApOtY

Gemini CLI Conceptual Architecture

CISC 322

Scott Rosenberg Fan Club

Callum McIntyre - **Group Leader** - Abstract, Architecture, Use Cases

Isaac Ouellette - **Presenter** - Use Cases, Slide Show, Script

Nicholas Saloufakos - **Presenter** - Use Cases, Derivation Process, Slideshow, Script

Jordan Bouckley - Architecture, AI Collaboration, Introduction, Overview

Andreea Cobzaru - Architecture, External Interfaces, Naming Conventions, Data Dictionary, Lessons Learnt, Conclusions

Zachary Gazaille - Architecture, AI Collaboration, References

What is Gemini CLI?

- Command line interface AI model
- Powered by Google Gemini LLM
- 3-tiered system | The CLI, The Core, The API
- Allows you to use AI within your command line/terminal
- Bridges the gap between your files and commands, and AI

Data Dictionary

- **Child Process:** an independent OS made by another process - executes shell commands in parallel
- **Context Window:** maximum amount of text the API can process
- **Event-Driven Model:** control flow pattern used by the system
- **Human-in-the-loop:** Safety design, human permission needed to execute tools
- **Non-deterministic:** property of LLMs where the same input can produce a different output
- **Promises:** unit of concurrency to handle asynchronous operations
- **Prompt engineering:** process performed by PromptConstructor, formats user input, history and instructions to optimize performance
- **ReAct pattern (Reason + Act):** logic flow used by the core, LLM makes a thought, uses a specific tool, observes the output and continues the reasoning
- **Selective context pruning:** performance optimization used by ContextManager, summarizes conversation history and removes irrelevant parts
- **Singleton pattern:** Architectural pattern to make sure a class has one instance
- **Streaming:** data transfer method, instead of waiting the API sends data in chunks
- **Token:** Smaller units that text is broken down into, so the LLM can process it
- **Tool:** function or script that the AI model can request to be executed.

Naming Conventions

- **CLI:** Command line interface - the user side
- **API:** Application protocol interface - the brain of the operation, ran within google on the server side
- **JSON:** JavaScript object notation
- **LLM:** Large language model
- **UI:** User interface

Architectural Styles

Style One: **Layered**

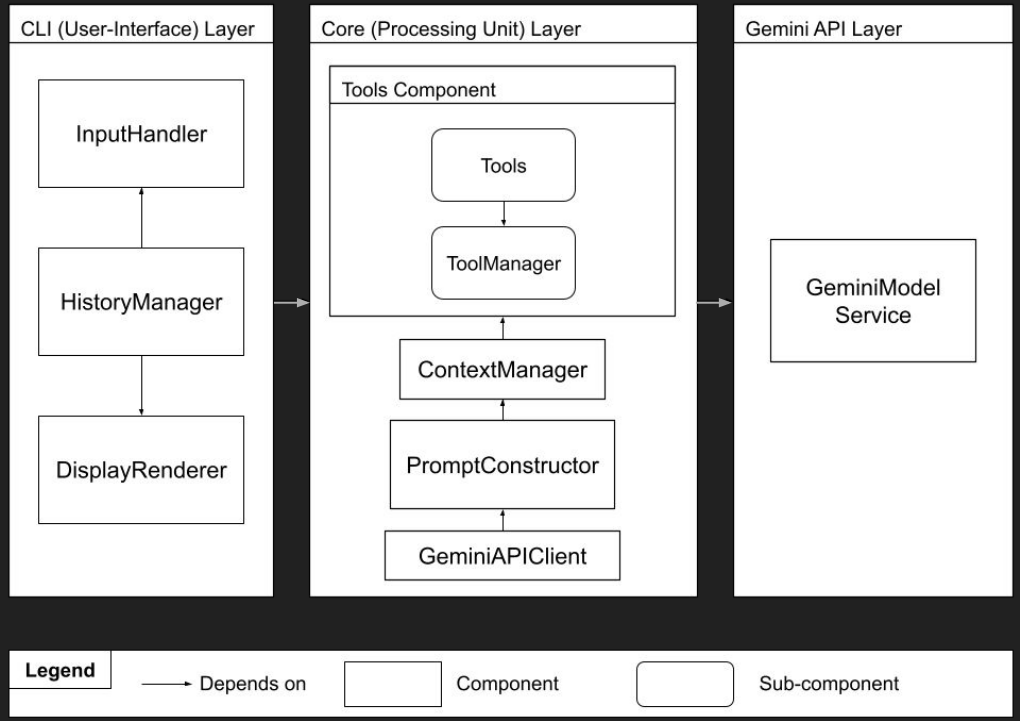
- 3 main layers | CLI - Core - API
- Organized hierarchically
- Each layer having specific set of info
    - Relies on other layers if not able to complete task
- Interacts only with layer adjacent
- Highly Modular → Evolvable

Style 2: **Client-Server**

- 2 Sides:
    - “Client” - CLI + Core
    - “Server” - API
- Client side takes and manages input
- Server side the brain of the operation
    - User prompts, CLI + Core contextualizes, sends to API
    - API sends structured response
- Logic held on server level, so local updates not usually needed
- Very low strain on local devices

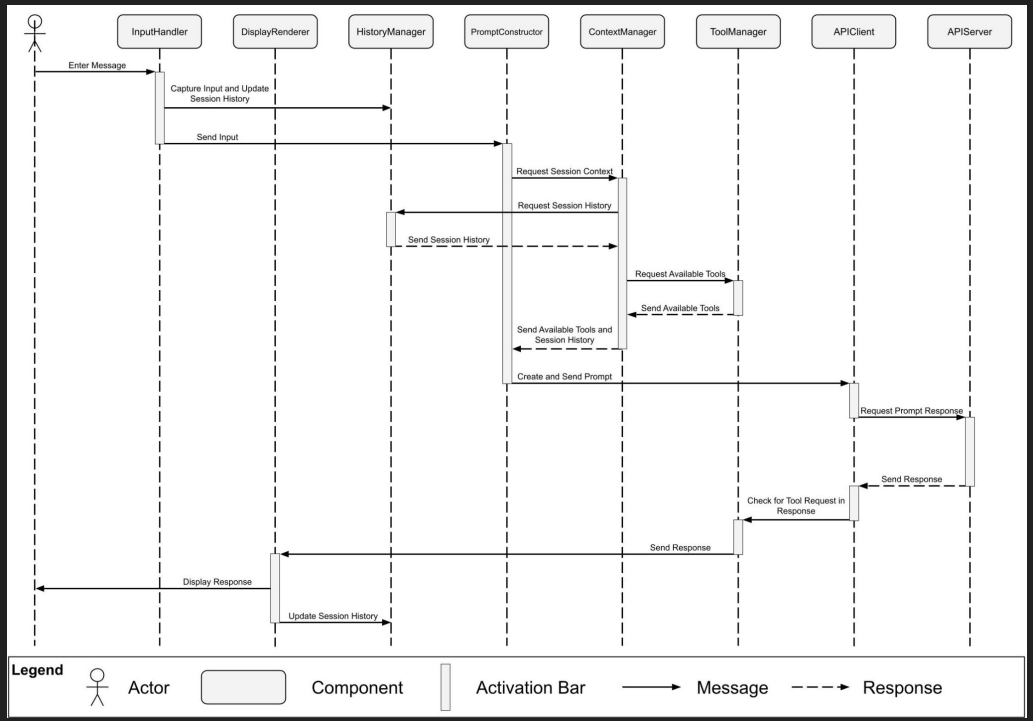
Box-and-Arrow Diagram

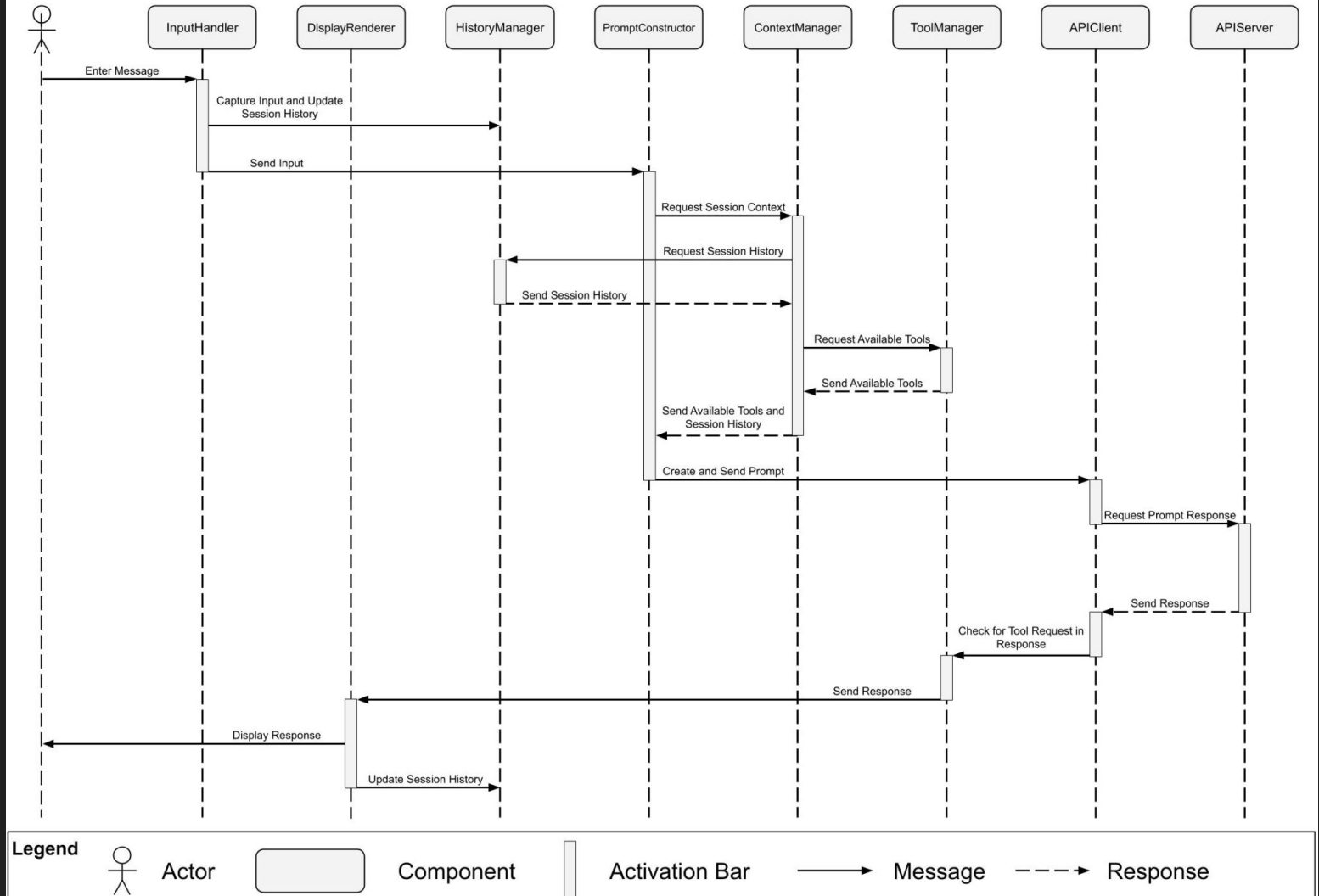
- CLI → Core → API
- CLI handles the input, user history, and display
- Core houses the tools
- Creates context with user history
- Prompts generated with PromptConstructor
- GeminiAPIClient sends prompts to server
- GeminiModelService Does the deep thinking, reasoning, and decides what tools to use

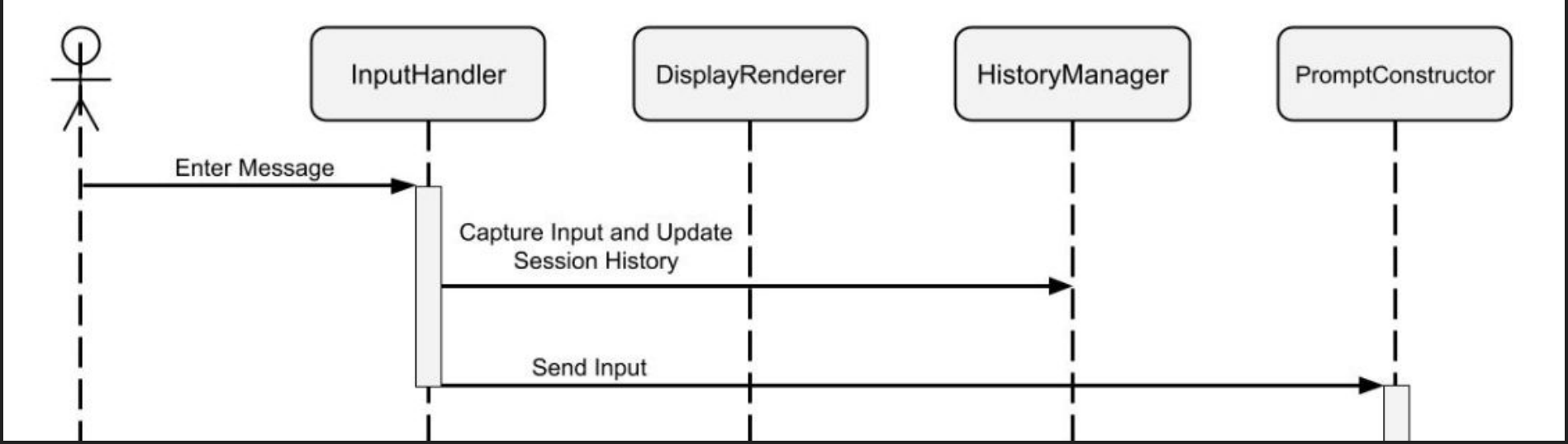


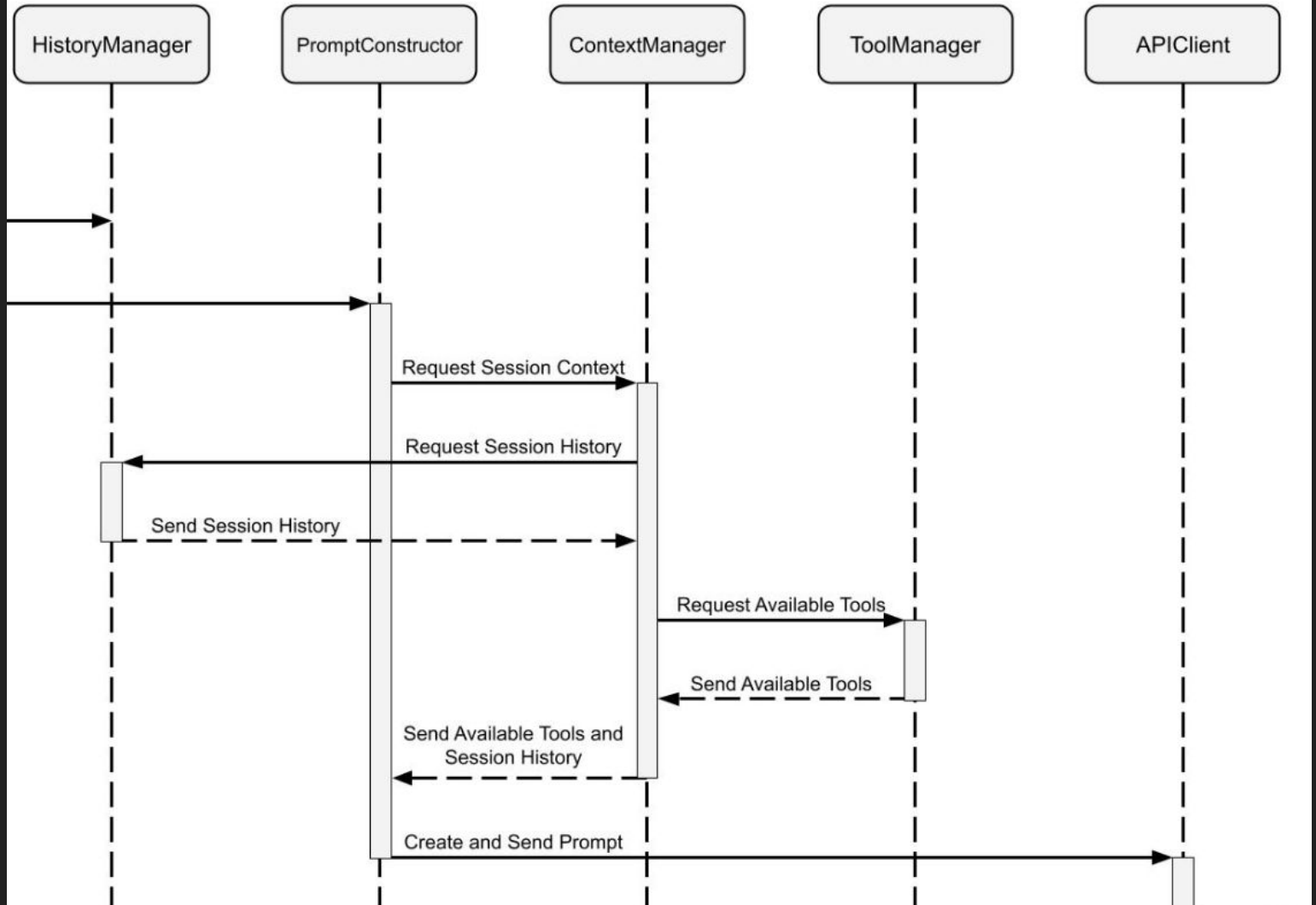
Sequence Diagram

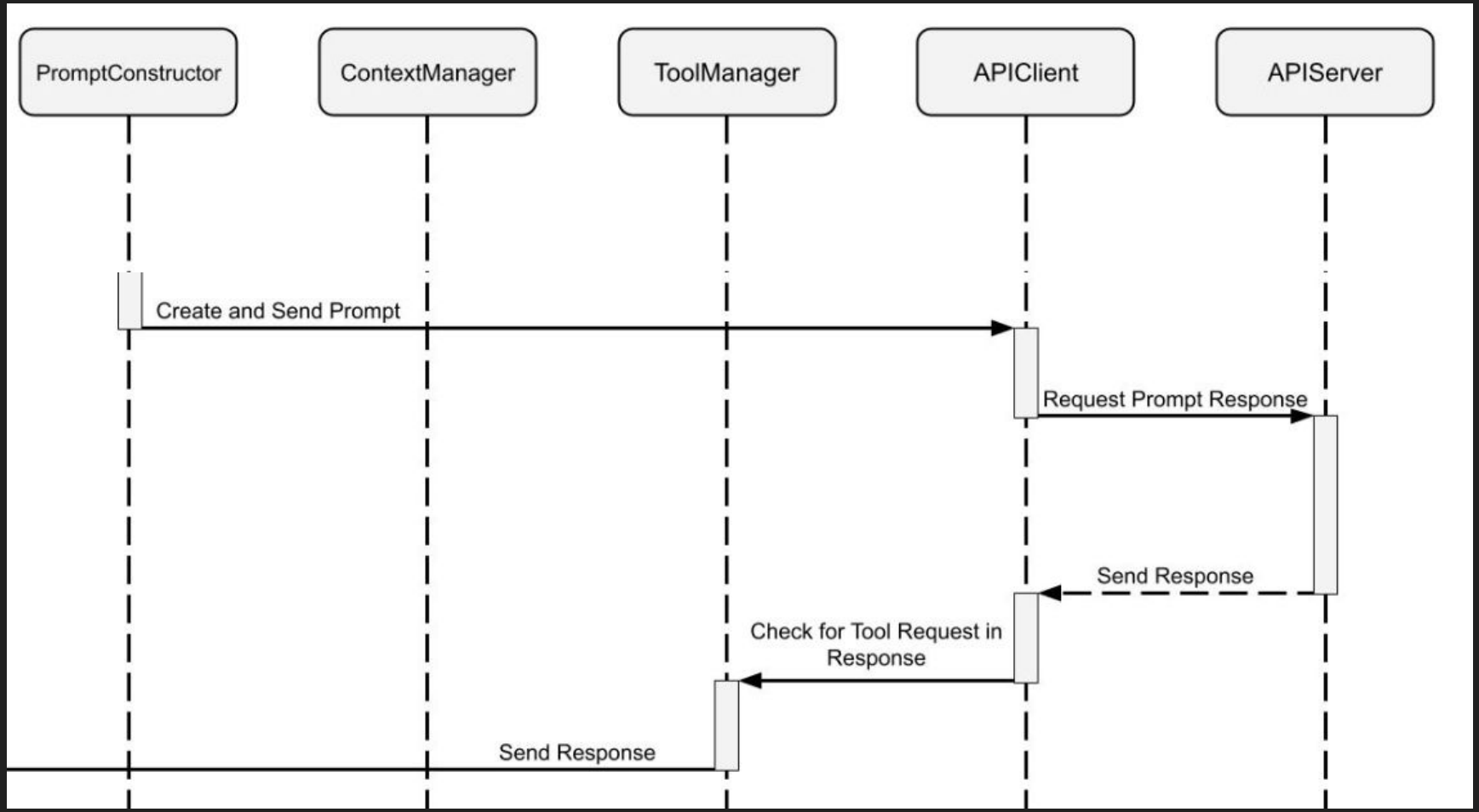
User Prompt Without Tool Execution:

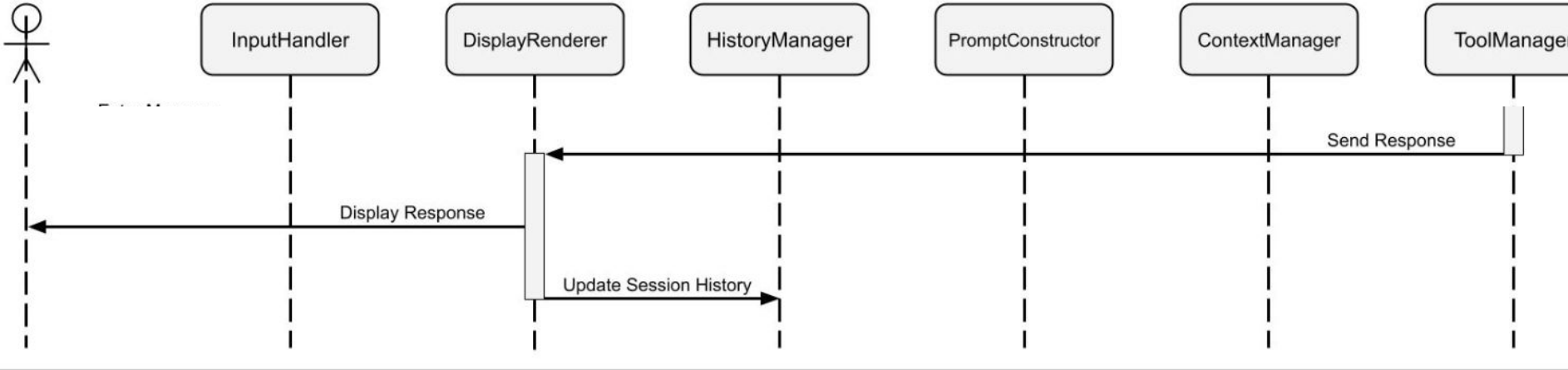



InputHandler
DisplayRenderer
HistoryManager
PromptConstructor
ContextManager
ToolManager
APIClient
APIServer
Enter Message
Capture Input and Update Session History
Send Input
Request Session Context
Request Session History
Send Session History
Request Available Tools
Send Available Tools
Send Available Tools and Session History
Create and Send Prompt
Request Prompt Response
Send Response
Check for Tool Request in Response
Send Response
Display Response
Update Session History
Legend
Actor
Component
Activation Bar
Message
Response


InputHandler
DisplayRenderer
HistoryManager
PromptConstructor
Enter Message
Capture Input and Update
Session History
Send Input


HistoryManager
PromptConstructor
ContextManager
ToolManager
APIClient
Request Session Context
Request Session History
Send Session History
Request Available Tools
Send Available Tools
Send Available Tools and Session History
Create and Send Prompt


PromptConstructor
ContextManager
ToolManager
APIClient
APIServer
Create and Send Prompt
Request Prompt Response
Send Response
Check for Tool Request in Response
Send Response


InputHandler
DisplayRenderer
HistoryManager
PromptConstructor
ContextManager
ToolManager
Send Response
Display Response
Update Session History

Interactions Between Subsystems

- **User Input:** InputManager → HistoryManager + PromptConstructor
- **State Recovery:** ContextManager → HistoryManager
- **Context Constructing:** PromptConstructor → ContextManager
- **Prompt Sending:** APIClient → PromptConstructor
- **Response Interpreting:** ToolManager → APIClient
- **Human-In-The-Loop:** ToolManager → User/DisplayRenderer
- **Remote Comms:** API Server → APIClient
- **Tool Execution:** Tools → ToolManager

Derivation Process

1. Gemini CLI Documentation
    - Layered and Client / Server

2. Gemini CLI Demo Video & Use Case Diagrams
    - User Entering a Prompt

3. Box-and-Arrow Diagram
    - Dependencies between Components

Alternative Architectures

Pipe and Filter

- System viewed as series of independent transformations
- Input = data stream
- Goes through core then to Gemini API
- Improves concurrency
- Less interactive, no back and forth discourse
- Overall less efficient

Interpreter

- Core = execution engine
- Gemini API = instruction set
- CLI takes human texts, makes it executable actions
- Follows interpreter process
- CLI abstracts LLM logic
- Adds much latency
- Instruction set is non deterministic

Lessons Learned

Necessity of Strict Interfaces

- Maintaining boundaries between CLI, Core, API were paramount
- Prevents blurring between layers

Asynchronous Requirements Add Complexity

- Event driven model important, but makes control flow more complexed
- ReAct loop pauses and resumes many times

Lessons Learned (cont’d)

Safety Must be a Priority

- “Human-in-the-loop” fundamental requirement
- Since LLMs are non deterministic, human review is required
- Security needs to be implemented deep within

Visualization Driven Discovery

- Took creating the sequence diagram to understand components
- Visualizing data flow revealed missing dependencies
- Allowed us to add ContextManager, HistoryManager

Limitations of Findings

- All similar to the Lessons Learnt slide
- Limited understanding of inner workings of API
- Potentially other components missing from sequence diagram
- Impossible to map the AI behaviour due to non-deterministic nature

AI Collaboration

LLMs used: GPT-4, Claude 4.5, Gemini 3 Pro

- Using Gemini for Gemini CLI was important (up to date)

Used For:

- Rubric Compliance
- Architectural Analysis
- Technical Articulation

AI Collaboration

- Used ‘driver-navigator’ approach - designated person to manage the interactions. Helped with consistent workflow and consistent prompts
- Prompting Strategy: word limits, specifying formatting. Injecting more information
- Verification Strategy: checked claims with CLI documentation. Made sure all information matched
- Attributed to ~25% of the final deliverable
- Streamlined workflow, improving pace
- Sometimes would hallucinate features