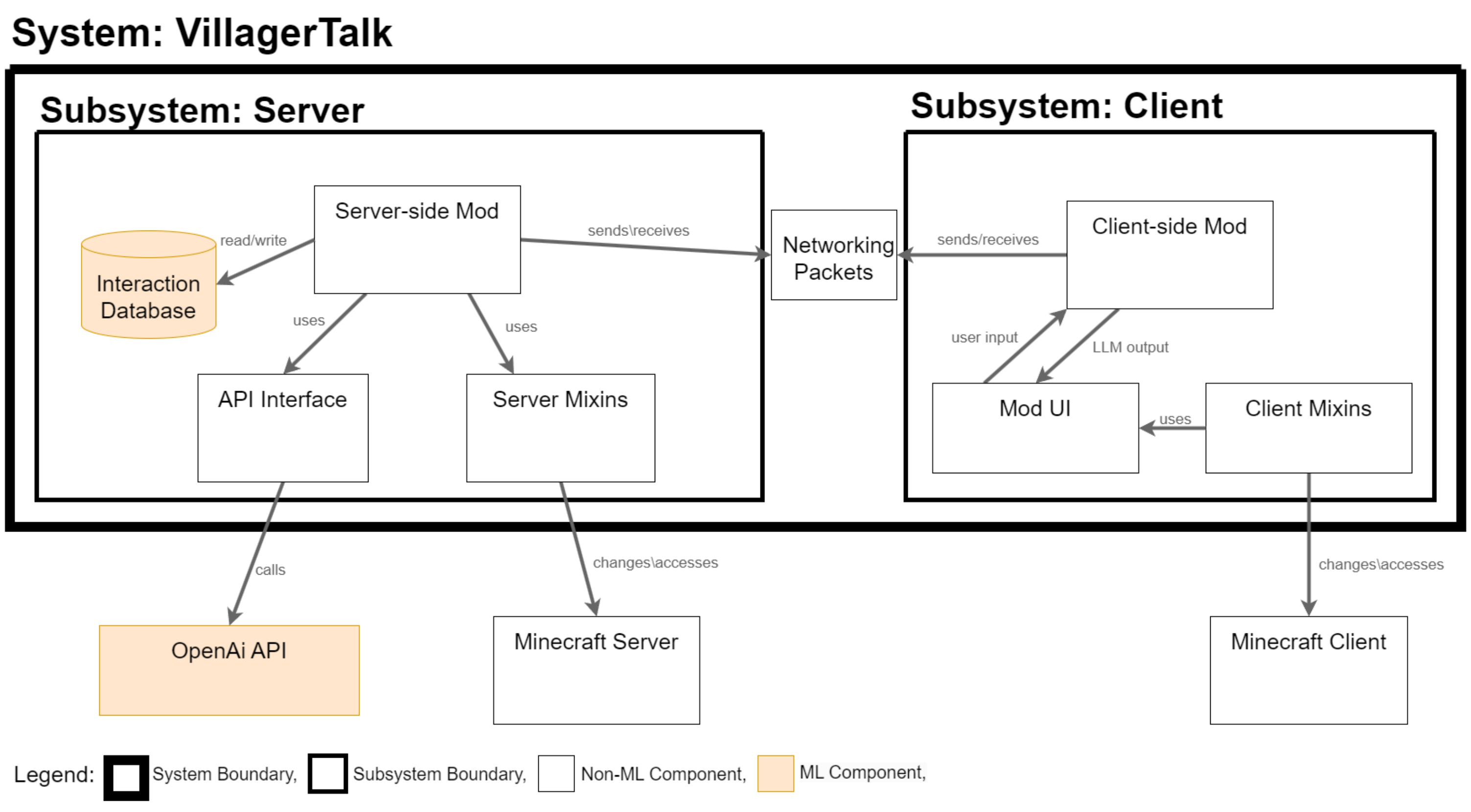

Components

Subsystem: Server:

- Server-side Mod: The Serverside mod, is the part of the system that extends the Minecraft Server, and runs in its Environment. It is the central system of the mod, and uses almost all other components. Requirements: ReqResponseParsing, NFReqOutputQuality, NFReqNewFunctionality.
- Interaction Database: A Database for storing previous Villager-Player Interactions. It is used for storing an interaction when it has ended, and constructing a paragraph in the system prompt for the LLM, that describes its relation to the player. Requirements: ReqConversationSaving
- API Interface: A java library, used for calling the OpenAi API. Requirements: NFReqAPIResponse,
- Server Mixins: A mixin is used to change the Minecraft Source code, or access private fields/methods. The Server-side mod uses them to change some behaviours of the game. Requirements: ReqCorrectModLoading

Subsystem: Client

- Client-side Mod: The client side mod handles the creation and change of the UI, and runs locally on the client. It manages interaction with the Server-side Mod. Requirements: NFReqVillagerInteraction,

- Mod UI: The user interface of the mod. It defines what is added to the users screen, and handles user input and output to the user. Requirements: NFReqChatMessages, ReqMessageSending
- Client Mixins: Changes the Minecraft clients source code, to include the Mod UI. Requirements: ReqGUIdrawing, ReqCorrectModLoading

Networking Packets: Packets used for communicating between Clients and Server

Outside Components:

- OpenAI API: provides LLM responsed when called. Reqirements: NFReqResTime
- Minecraft Server: The games serverside code.
- Minecraft Client: The games clientside code.

Design Questions and Answers

- How can the Villager interactions feel unique for each Villager and Player?

  Multiple parameters, that are input into the system prompt (e.g. Villager profession, biome, age, previous interactions, randomized mood or other character traits)
- How can game performance impacts from the mod be minimized?

  Run as much as possible on the server, only run when a villager is open.
- How can we ensure maximum compatibility with other mods?

  Keep the mixins (changes to the source code) light, and inform the users about potential conflicts.
- How do we ensure LLM output quality and realisticity?

  Test different system prompts, and see how they influence the AI's behaviours.
- How well will the server-side mod handle multiple users talking to villagers at the same time?

  Depends on the server performance, try to optimize by running each player interaction in a seperate thread.
- How easy will it be to port the mod to a new game version?

  As long as Villagers aren't changed, it shouldn't be too hard.
- How do we handle API costs, do we require users to input their own API Key?

If the mod goes public, we will need to ask the users to input their own API Key, as the potential costs are not forseeable.

- How do we make the mod customizeable for user, e.g. what configuration options?

  Let the user set a maximum discount, LLM Model thats used, maybe even starting prompt parameters.

- How do we handle Errors (e.g. Network Errors, API downtime, game crashes)?

  Implements solid Exception logging, to help debug problems.

- How do we stop users from abusing the mod, since LLMs are easily gaslit and jailbroken?

  Design a solid starting prompt, that prevents the LLM to change its role and be too easily convinced about changing prices.