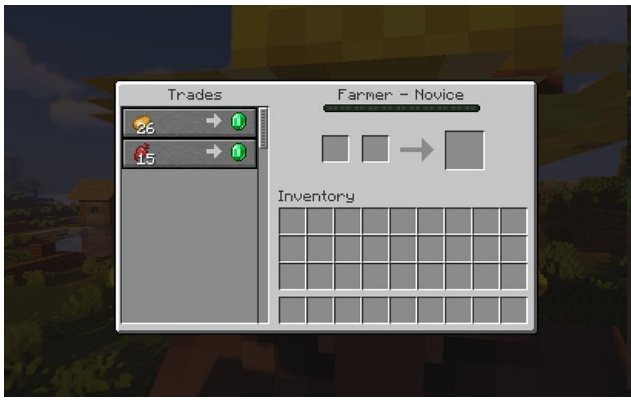
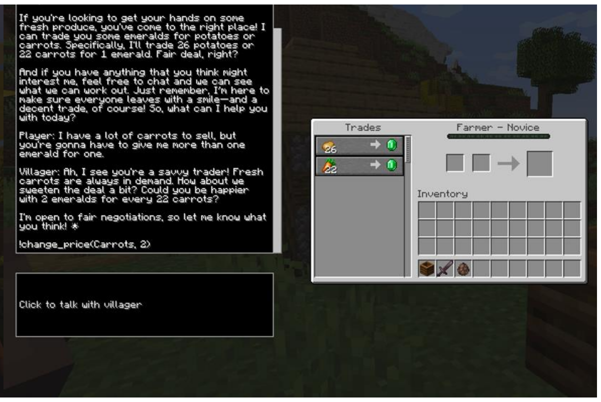
VILLAGERTALK

A mod to enhance Minecraft's trading system.



Base Interface


If you're looking to get your hands on some fresh produce, you've come to the right place! I can trade you some emeralds for potatoes or carrots. Specifically, I'll trade 26 potatoes or 22 carrots for 1 emerald. Fair deal, right?

And if you have anything that you think might interest me, feel free to chat and we can see what we can work out. Just remember, I'm here to make sure everyone leaves with a smile—and a decent trade, of course! So, what can I help you with today?

Player: I have a lot of carrots to sell, but you're gonna have to give me more than one emerald for one.

Villager: Ah, I see you're a savvy trader! Fresh carrots are always in demand. How about we sweeten the deal a bit? Could you be happier with 2 emeralds for every 22 carrots?

I'm open to fair negotiations, so let me know what you think! *

!change_price(Carrots, 2)

Click to talk with villager

Modded Interface

Leon Mandler, k12104186

Sandro Assek, k12102749

Table of contents

Table of contents **1**
Goal of the system **2**
Requirements **2**
Functional Requirements 2
Non-functional Requirements 3
Use case description **3**
Use Cases (UC): 3
Supportive Use Cases (SUP): 4
Use case diagram
Implemented use cases **5**
Tracability matrix **6**
Domain model **8**
Architecture diagram **9**
Components description **9**
Subsystem: Server 9
Subsystem: Client 10
Outside Components 10
Design Questions **10**

Goal of the system

Improve the gameplay experience and immersion of Minecraft, by making the Villager NPCs more lively and fun to interact with. In the base game, players can trade with villagers, to obtain items or currency. This works with a simple HUD and static prices. We plan to develop a mod for the game that lets the player talk to a villager and barter and negotiate with them to get better prices. The mod will also change the existing HUD, to incorporate a chat window in the villager HUD that is linked with a LLM, to make the interaction possible.

Requirements

Functional Requirements

- **CorrectModLoading:** IF the mod is installed when the game is opened the abric Mod Loader shall inject the mod into the game
- **GUIdrawing:** IF its inventory is openable WHEN a villager is right clicked the screen class shall add its GUI elements to the screen.
- **WritingFieldFocusing:** IF the writing field is unfocused WHEN the mouse clicks on the writing field or enter is typed, the screen shall set the writing field to focused.
- **MessageSending:** IF the writing field is focused WHEN enter is typed, the Network Handler shall send the prompt packet to the server
- **AdaptGUI:** If the client receives a packet the GUI shall clear the writing field and add the prompt to the chat window.
- **ResponseParsing:** IF the LLM response contains commands WHEN the response is received on the server, the server side mod shall change the villagers prices. [AI]
- **ConversationSaving:** IF the player had a previous interaction with the villager, WHEN the villager is opened the screen shall display the previous conversation.
- **ChatMessages:** While the villager is alive and starts a conversation all previous chat messages shall be visible to the client.

Non-functional Requirements

- External interfaces
    - **APIResponse:** When the prompt is received on the server the API interface shall make an API request to a LLM API. [AI]
- Performance
    - **ResTime:** If the internet connection is good and the API operates normally when the LLM receives the prompt, the LLM shall respond within 3 seconds. [AI]
    - **VillagerInteraction:** While it is possible to trade with villagers, the user shall be allowed to interact with villagers.
    - **OutputQuality:** When an API request is made the mod shall provide a clear role for the LLM, that leads to quality responses. [AI]
- Attributes
    - **NewGameVersion:** When a new version of the game is released, the mod shall be easily portable to the new version.
    - **Documentation:** While the source code is maintained by developers, the source code shall offer a full and up-to-date documentation.
    - **NewFunctionality:** When adding new functionalities, the new code shall be easy to integrate to existing code.
- Constraints
    - **RunEnv:** While minecraft is running on Java, the mod shall run on java.
    - **GameStandards:** While the fabric api gets updated, the mod shall make good use of it and adhere to its standards
    - **RunOnJavaOnly:** While there is no good modding tool for other versions, the mod shall only be available in Java edition.
    - **CleanShutdown:** When the game is closed or the server is shut down, the database shall have the conversations stored.

Use case description

Use Cases (UC):

- **[VilConv]** Conversation with a Villager: The villager responds if the player sends a message
- **[NegPrc]** Negotiate Price with Villager: The villagers responds to a price change for a given item
- **[ChaHUD]** Change Villager Trading HUD: Add a chat window for conversations and a text box to type in
- **[RemPyr]** Remember each Player: The last interactions incl. current item prices and conversations with a user are displayed for each villager
- **[Mod2Gme]** Load the Mod into the game: The game receives a modded version for villager trading

Supportive Use Cases (SUP):

- Fabric Mod Loader loads Mod

  [VilConv], [NegPrc], [ChaHUD], [Mod2Gme]
- Send API request to LMM API

  [VilConv], [NegPrc]
- Receive API response from LMM API

  [VilConv], [NegPrc], [ChaHUD]
- Process LLM Command

  [RemPyr], [NegPrc]
- Send packet to Client

  [VilConv], [NegPrc], [ChaHUD]
- Receive packet from Client

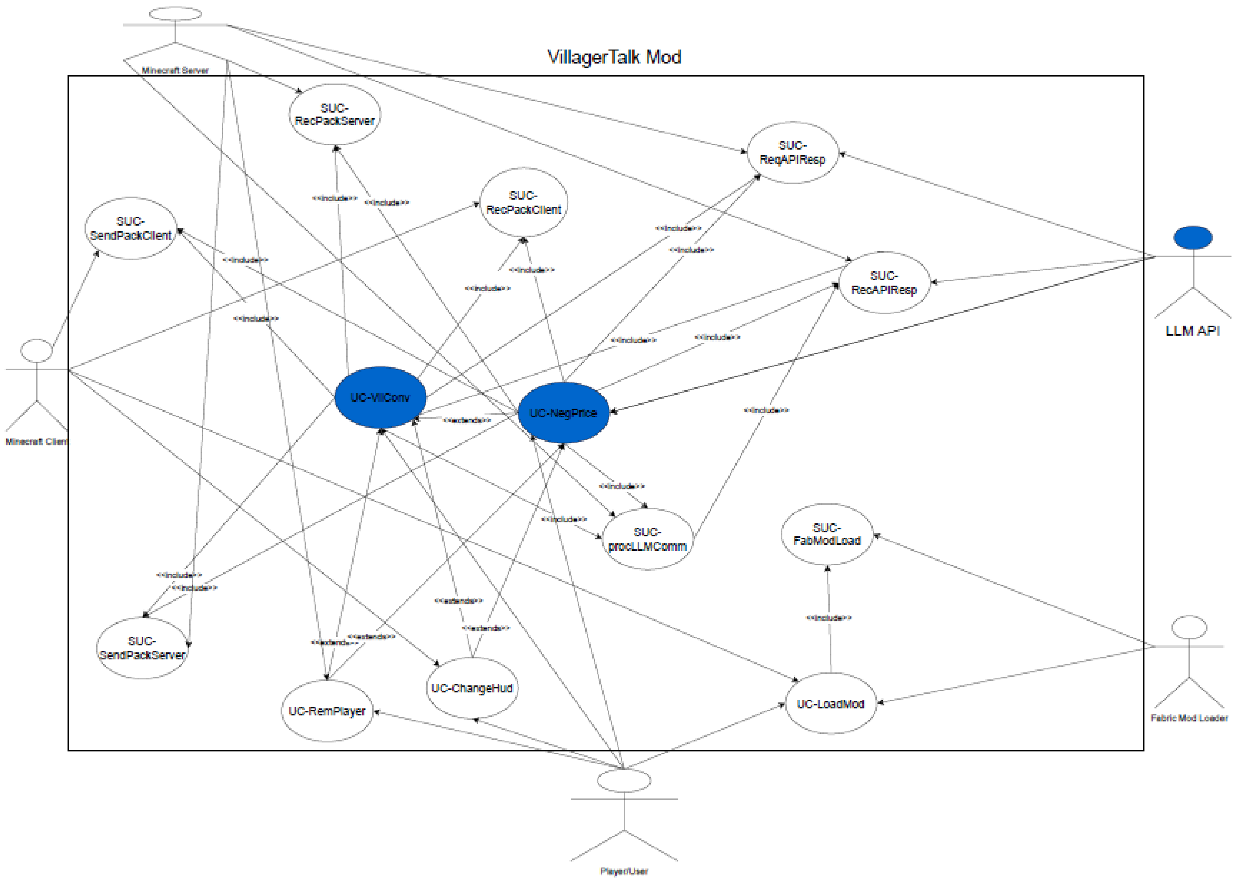
  [VilConv], [NegPrc]
- Send packet to Server

  [VilConv], [NegPrc]

- Receive packet from Server

  [VilConv], [ChaHUD]

Use case diagram



Implemented use cases

Use Case	Implemented
[VilConv] Conversation with a Villager	X
[NegPrc] Negotiate Price with Villager:	X

[ChaHUD] Change Villager Trading HUD	x
[RemPyr] Remember each Player	partially*
[Mod2Gme] Load the Mod into the game	x

* The changed trading offers, conversations and starting prompts are remembered for each player as long as the current game is not closed.

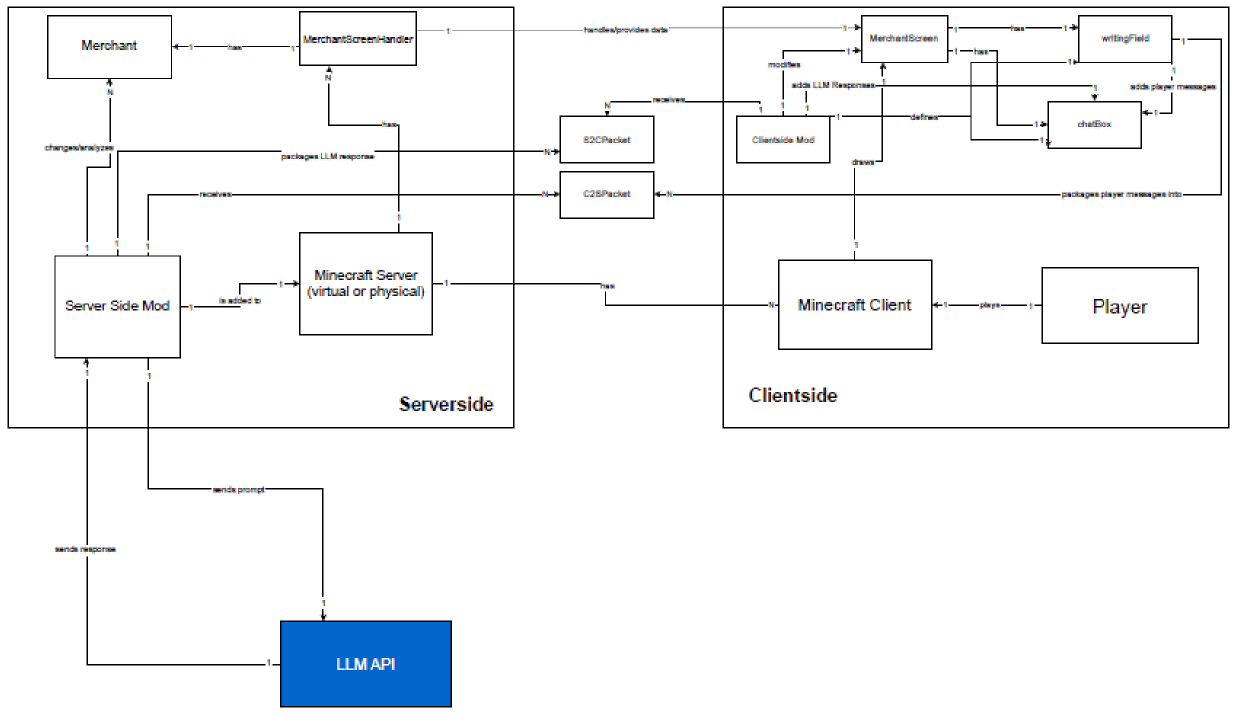
Tracability matrix

Unfortunately, a image would result in a very poor resolution

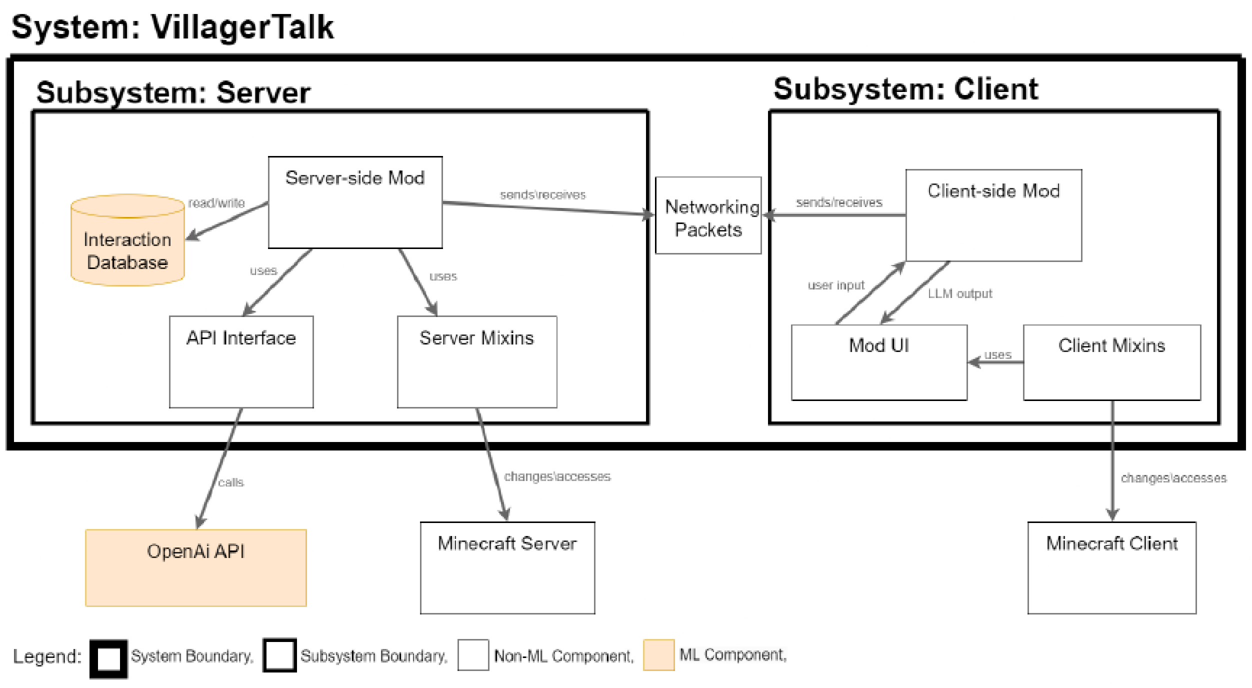
Use cases	FabModLoad	ReqAPIResp	RecAPIResp	ProcLMMComm	SendPackClient	RecPackClient	SendPackServer	RecPackServer	VilConv	NegPrice	ChangeHud	RemPlayer	LoadMod
Correct ModLoading	X												X
GUIdrawing											X		
WritingFieldFocusing											X		
Message Sending		X			X	X		X	X	X			
ResponseParsing			X	X									

ConversationSaving							X					X	
APIResponse		X						X					
ResTime			X										
VillagerInteraction									X	X			
ChatMessages									X	X	X		
OutputQuality		X											
NewGameVersion	X												X
RunEnv	X												X
GameStandards	X												
RunOnJavaOnly	X												
CleanShutdown												X	

Domain model



Architecture diagram



Components description

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 response when called.

  Requirements: NFReqResTime

- **Minecraft Server:** The games serverside code.
- **Minecraft Client:** The games clientside code.

Design Questions

- 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.