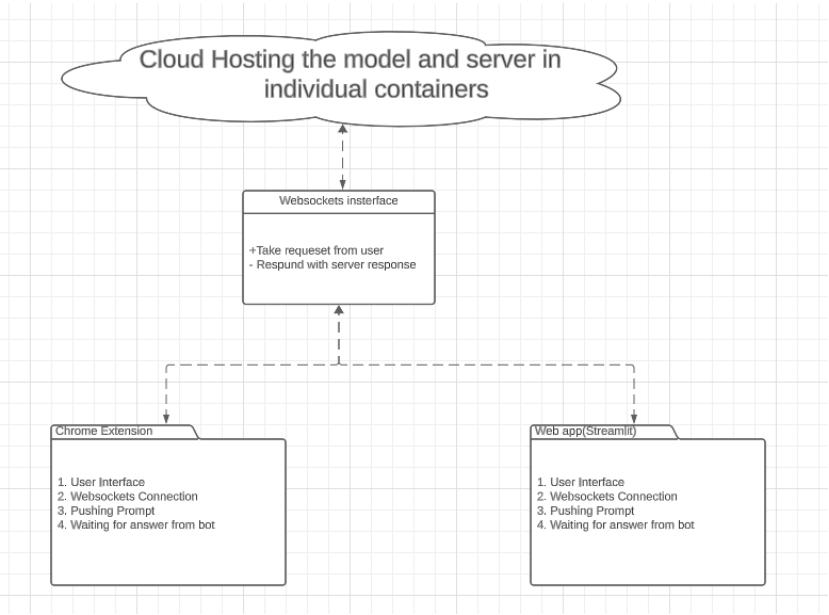
ChatFi

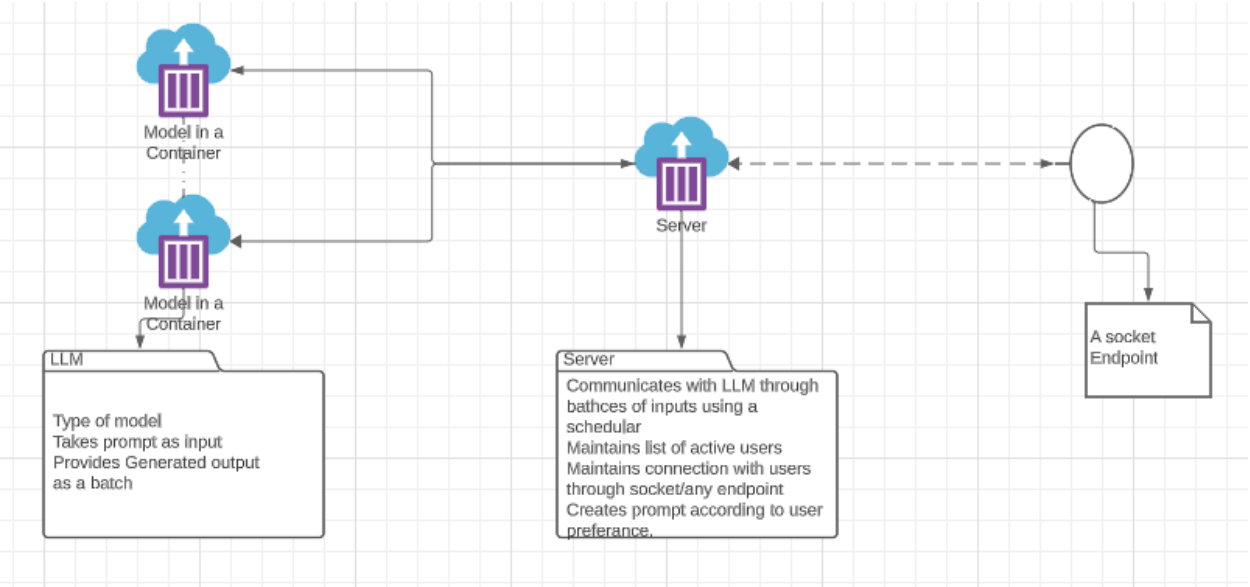
Document to set the vision and structure of the product.

This will be as an open to all on freemium service,whose chats can be read by the owners.

Basic Structure:



Structure of Cloud



1. **Model in a Container:** Container hosted in the cloud holds the model, there can be multiple containers holding one type of model each.
   A. Each container is connected to the central server.
   B. Just takes prompt as input and outputs the generated response.
   C. Runs according to the scheduler present in the server.
   D. Isolation helps in easier updation and maintainance.

   Attributes: A. Server Endpoint Connection,
   B. [{'Identifier': 'Prompt'},....]
   C. [{'Identifier': 'Response'},....]
   Methods : A. Generate Response ([prompt])
   Constraints : This is a very high RAM and GPU intensive container as it holds the complete model.

2. **Server in a Container: This is the core of the service.** It maps the input form the user , generates a valid prompt accordingly , passes it to the selected model container , accepts response and routes it back to the concerned user.
   a. Server connects to the user and authenticates it.
   b. Creating and structuring prompt:
      Types of prompts :
      i. Without context normal Q/A
      ii. With Context Q/A:
      If the user chooses with context Q/A then and uploads documents , then the server creates embeddings and stores it in vector store, which then are retireived using similarity search.

These documents as vector stores are saved in the server until the user logs out or deletes them.

iii. Summarization of the given context : This is the most intensive procedure because due to limitations of length of generation large documents can't be given in contexts. Therefore we need to create chunks of the documents and summarize each chunk . Then use these summaruized chunks as documents and continue until we donot get the summary in the required number of words.
Batch generation of responses helps speed up the process but still it is the most resource intensive.

iv. Recommendation prompt :
    1. Recommendation prompt can be based on context or off context.
    2. These types of prompt work very well when the LLM is finetued on the domain .
    3. If relevant context is provided then recommendations can be generated without finetuning as well.
    4. The risk of hallucinations is highest in this type of requests if finetuning is not done properly.

v. FAQ : These are helpful while describing the working of the project given a list of question and answers . These are the most simplest kinds of prompts and can even be used by just doing a similarity search without running the LLM.

c. Choosing which type of question to ask can be done in two ways 👍
    i. Accepting it from the user.
        1. Pros : Simpler chain
        2. Cons : User forgets chainging the type of prompt can result irrelevant responses from the LLM.
    ii. Automatically classifying the user input by vectorizing then classifying the user input.
        1. Pros: Minimal user input required .
        2. Cons : Complicated chain.
d. Connections between Server and Clients can be through sockets, https or REST apis.
e. Handling the traffic:
    i. Low traffic : Normally passing the formatted prompt directly to selected LLM .
    ii. Medium traffic : Scheduling according to the priority of the client and without starving any of them.
    iii. Medium high traffic : Scheduling and batching the prompts

iv. High traffic : creating another container of the same LLM and diverting the traffic after scheduling.

f. Chaining the chats: If the user wishes then the chats can be chained, but this takes up valuable length of context as well as the space in output result.

g. Attributes:

```
Active users:{
        'USER_ID':{
                'user info':{},
                'Documents':[],
                'Vectorstore':vectorStore(),
                'Chats':[]
        }}}
Active LLMS:{'LLM_ID':(link to llm container)}
```

h. Methods:

i. Schedule([prompts,..])
ii. Authenticate user
iii. Get_user input
iv. Respond to user

3. **Socket Endpoint:** The endpoint through which the user connects to the server. Recommended for low latency: websocket.
   Usuall doesnt matter as the most timeintensive task is the response generation and comparatively network latency has minimum impact .
4.