**Product Design for Legal Assistance Software**

**1. Overview**

This software assists legal workers and civilians by summarizing legal documents from the Kerala High Court website and enabling users to find similar cases based on natural language queries. The main functionalities include:

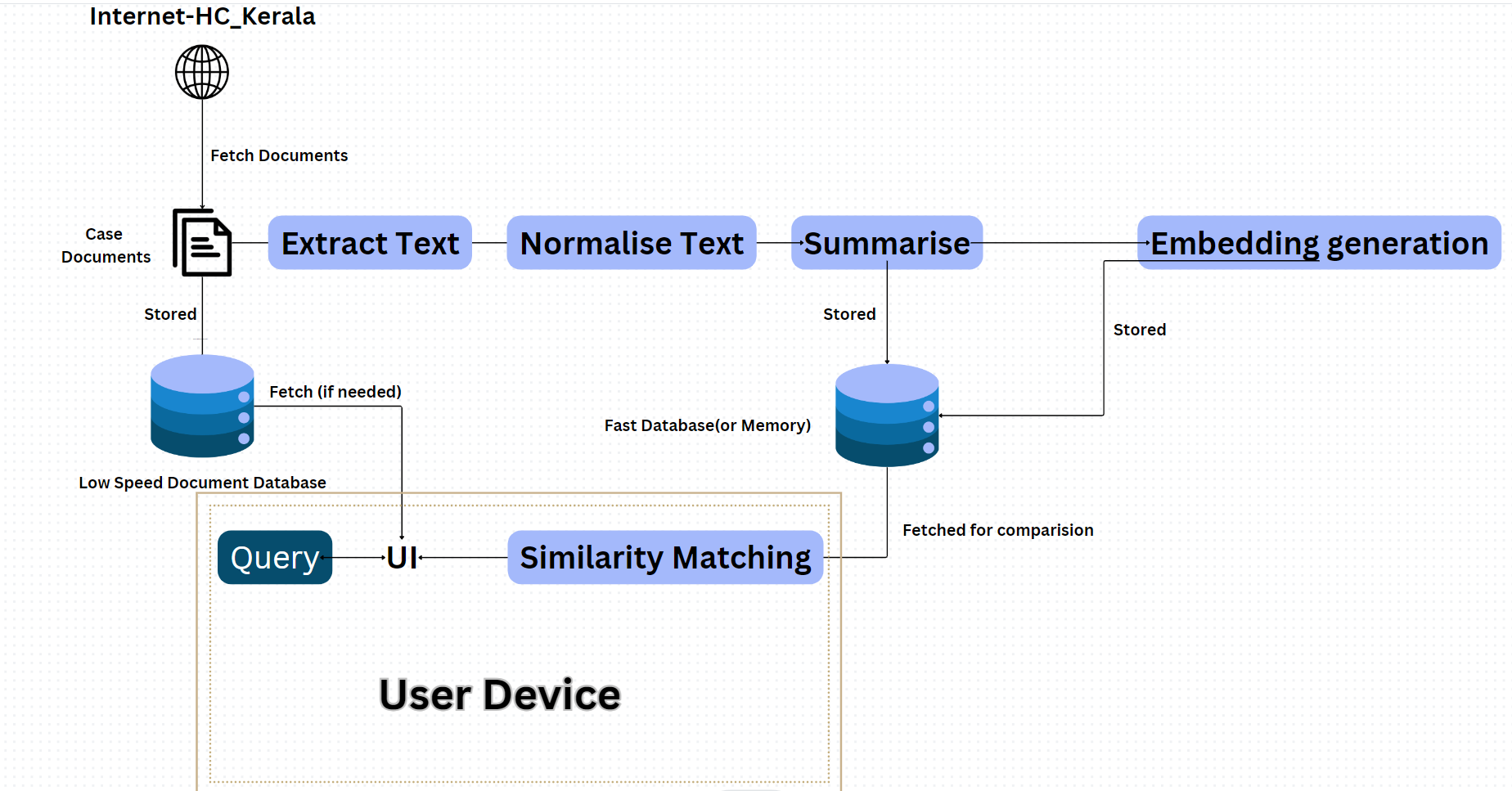
1. **Document Retrieval:** Automatically fetch legal documents.
2. **Text Extraction:** Extract text from PDF documents.
3. **Summarization:** Summarize the documents, extracting key details.
4. **Storage:** Store summaries and text embeddings in a fast database and full documents in a slower database.
5. **Query Processing:** Allow users to query the system using natural language.
6. **Similarity Matching:** Use cosine similarity to find and present summaries of similar cases.

**2. Functional Components**

1. **Document Retrieval Module(Needed to be ran only once for data):**
   * **Function:** Fetch documents from the Kerala High Court website.
2. **Text Extraction Module(Needed to be ran only once for data):**
   * **Function:** Extract text from PDF documents.
3. **Text Preprocessing Module(for both query and data):**
   * **Function:** Clean and normalize extracted text.
4. **Summarization Module( Only needed for the data, rarely needed for the query):**
   * **Function:** Extract key information and summarize documents.
5. **Database:**
   * **Function:** Store summaries and metadata in a fast database(MongoDB), full documents in a slower database(S3 or Postgress).
6. **Embedding and Similarity Matching Module:**
   * **Function:** Generate embeddings and find similar cases using cosine similarity.
7. **User Interface (UI):**
   * **Function:** Provide a platform for users to input queries and view results.

**3. System Architecture**

The architecture follows a modular design to ensure scalability and maintainability.



**4. Steps And Architecture Explanation**

**1. Document Retrieval**

**Function:** Automatically fetch legal documents from the Kerala High Court website(can be done manually for prototype).

* **Process:** The system periodically checks the Kerala High Court website for new documents, downloads them, and stores them locally for processing.

**2. Text Extraction**

**Function:** Extract text content from PDF documents.

Here it is done using the python module called pdfplumber.

* **Process:**
  1. Open the PDF document.
  2. Read each page and extract the text.
  3. Concatenate the text from all pages to form a single string.
  4. Store the extracted text for further processing.

**3. Text Preprocessing**

**Function:** Clean and normalize the extracted text to prepare it for summarization and embedding generation.

* **Process:**
  1. Remove special characters and digits.
  2. Convert text to lowercase.
  3. Remove stopwords.
  4. Lemmatize the text to reduce words to their base form.

**4. Summarization**

**Function:** Extract key details and create a summary of the document. We have to perform extractive summarization.

Here the google generative Gemini AI model is used.

* **Process:**
  1. Identify and extract key information such as judge names, lawyer names, case numbers, and important dates.
  2. Create a concise summary that captures the essence of the document.
  3. Store the summary and key details in a fast-access database.

**5. Storage**

**Function:** Store summaries and original documents efficiently.

For prototype, the chat initialization files(history.pkl in code) is store in local EC2 instance storage since, its highly compressed. A small summary of the general details of the cases are stored in a SQLite server running ChromaDB locally on the EC2 instance(cloud machine).

* **Process:**
  1. Store the summaries along with their metadata in a fast-access database (e.g., MongoDB).
  2. Store the full documents in a slower-access database (e.g., PostgreSQL or AWS S3) for archival purposes.

**6. Embedding Generation**

**Function:** Generate numerical embeddings for the text to capture its semantic meaning.

We using the chromaDB’s built in embedding generation capabilities.

* **Process:**
  1. Preprocess the text.
  2. Send the preprocessed text to an embedding generation service (e.g., GPT-4).
  3. Obtain and store the embeddings along with the summaries in the database.

**7. Similarity Matching**

**Function:** Find and present similar cases based on user queries.

For prototype the ChromaDB is doing this for us.

* **Process:**
  1. Preprocess the user query.
  2. Generate embeddings for the query.
  3. Calculate cosine similarity between the query embeddings and stored case embeddings.
  4. Retrieve and present the most similar cases to the user.

**8. Interactive Query System**

**Function:** Provide a web interface for users to input queries and view results.

Interative UI is made using the generative AI’s interative response shown in an interactive way

* **Process:**
  1. User inputs a natural language query through the search bar.
  2. The backend processes the query and finds similar cases.
  3. The frontend displays the summary of similar cases to the user.

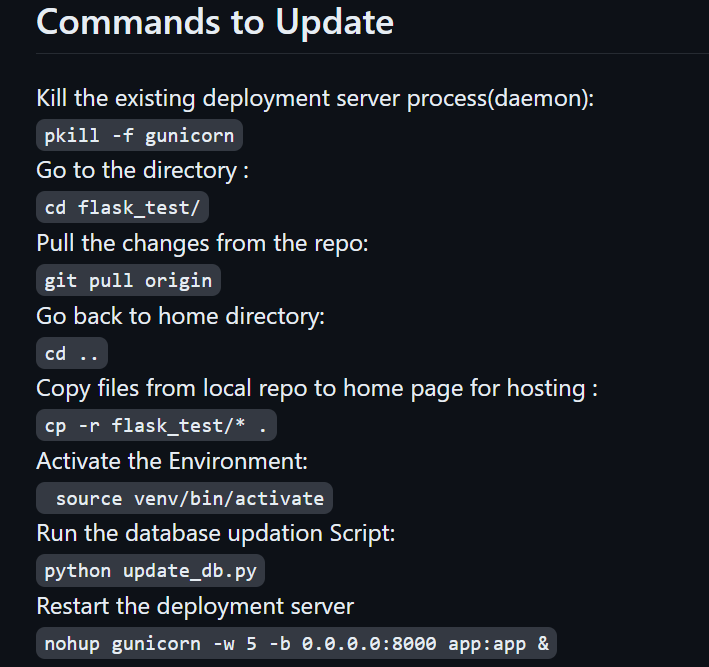
**DETAILS ON PROTOTYPE:**

The prototype functions in the following way:

1)The new case files are uploaded into a cases file in the a local github repo.

2)The file admin.py, automatically extracts all the relevant data of a particular case and constructs a chat environment file(chat initialization file called history.pkl). and also generated an updation packet for the chromaDB hosted in the cloud for the new cases.  
The admin.py is a highly flexible python code, it is able to recognize if there are more than 1 interim order or absence of presence of the Judgment. It should work in all possible scenarios and extract the relevant data.

3)When the case environment files for all the cases are generated and the updatation packet is made. Push the changes into the GitHub.

4)Log and Connect to your EC2 instance(use ssh, PuTTY or instance connect).  


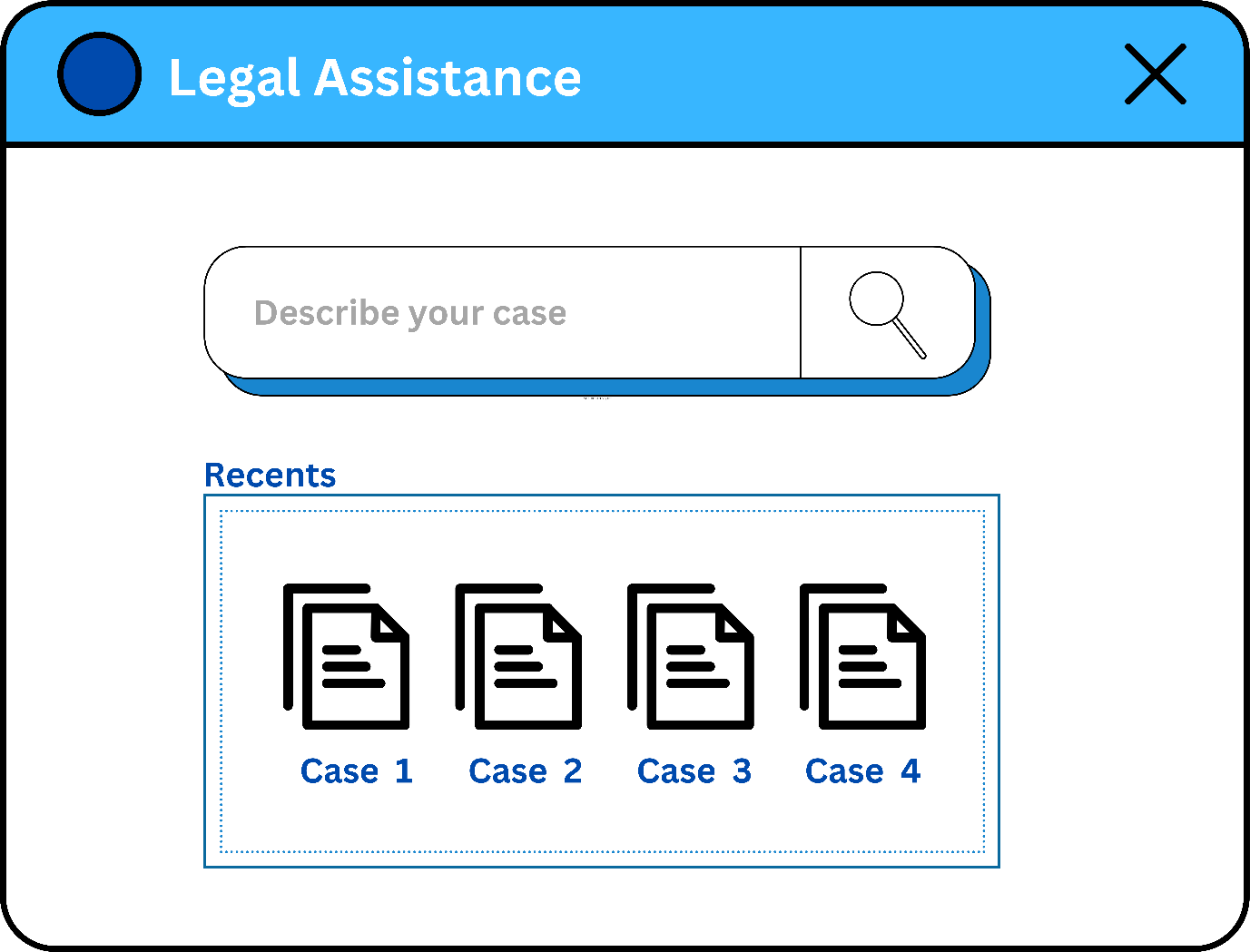
Assuming the server was already setup during development

NOTE: The above commands can be completely automated using GitHub actions to SSH into the AWS EC2 instance using the security credentials and run the said commands when a change is pushed into main branch.

**UI Design**

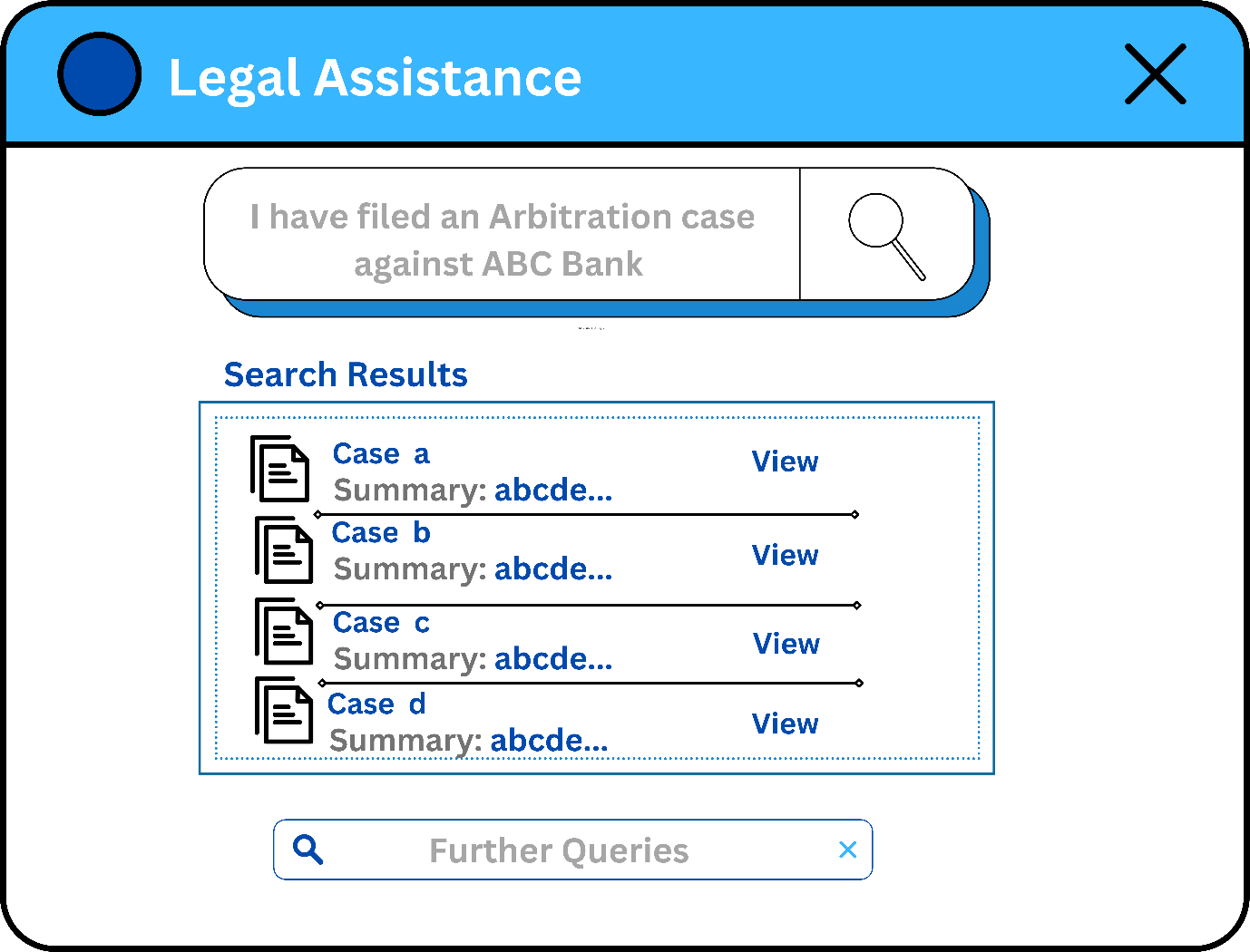
**Home Page**

* **Search Bar:** Allows users to input their query in natural language.
* **Recent Cases:** Displays a list of recently summarized cases for quick access.



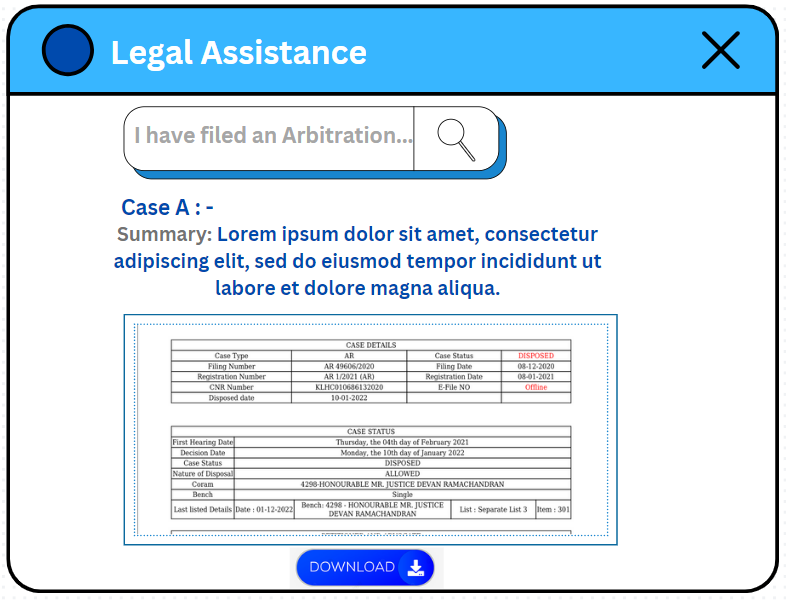
**Search Results Page**

* **Results List:** Shows a list of cases that match the query based on cosine similarity.
* **Summary View:** Brief summaries of each case with key details.



**Case Details Page**

* **Summary:** Detailed view of the selected case summary.
* **Download Option:** Option to download the full document or summary.
* **Related Cases:** List of other cases with high similarity.



**Financial Aspect**

**Subscription Model(can be flexible):**

1. **Free Tier**:
   * **Access**: Limited to basic features with usage restrictions.
   * **API Calls**: Up to 60 requests per minute during the preview period.
   * **Summary**: Users can perform a limited number of document extractions and queries.
   * **Price**: Free
2. **Enterprise Pro Tier**:
   * **Access**: Full access to advanced features, including more extensive usage of the Gemini API.
   * **API Calls**: Charges of approximately $0.00025(approx. Rs. 0.021) per 1,000 characters for text input.
   * **Summary**: Unlimited document extraction and advanced search capabilities for big legal enterprises.
   * **Price**: Estimated starting at $20(Rs. 1700) per month based on Google's new AI Premium plan.

**Cost Analysis**

**Gemini API Usage**:

* + For the Pro tier, the cost per 1,000 characters is around $0.00025(approx. Rs. 0.021). Assuming an average legal document is 10,000 characters, each document extraction will cost approximately $0.0025(approx. Rs. 0.021).
  + If an average user processes 100 documents per month, the API cost will be $0.25(approx. Rs. 21) per user.

**Storage Costs:**

* + **AWS:** Using AWS S3 for storage can vary based on usage but averages around $0.023 per GB per month for standard storage. For fast database storage, you might use AWS RDS which can range from $0.025 to $1.413 per GB per month depending on the instance type and performance. For slower storage, perhaps AWS Glacier could be utilized, which is around $0.004 per GB per month.
  + **GCP:** GCP offers similar services. For example, GCP Cloud Storage Standard costs around $0.020 per GB per month. For fast database storage, Cloud SQL could be used, ranging from $0.0151 to $2.622 per GB per month. For slower storage, GCP offers Nearline Storage at around $0.010 per GB per month.

**Infrastructure Costs:**

* + Both AWS and GCP have similar pricing models for hosting and infrastructure. This can vary widely based on the specific services used, such as virtual machines, networking, and data transfer costs. For simplicity, let’s assume this to be $1(Rs. 84) per user per month.

**Other Expenses:**

* + Customer support, marketing, and other operational expenses could vary based on the scale of operations and specific strategies employed. Let’s assume this to be $1(Rs. 84) per user per month.

Adding these up:

* **Storage Costs:** Let's assume a total of $0.05 per GB per month for a combination of fast and slow storage.
* **Infrastructure Costs:** $1 per user per month.
* **Other Expenses:** $1 per user per month.
* **API Costs:** $0.25 per user per month.

So, the total operational costs per user per month would be around $2.30 (Storage) + $1 (Infrastructure) + $1 (Other Expenses) + $0.25 (API Costs) = $4.55(Rs. 380).

To maintain profitability and ensure a reasonable profit margin, as mentioned, a minimum subscription fee of around $5(Rs. 420) per user per month would be appropriate.

We can achieve these figures by either charging more from paying customers for more feature and give limited access to the free tier customers or we can average it out, depending on the management’s decision.

To estimate the company's earnings under the provided subscription model, we'll need to consider the number of users subscribing to each tier and their respective pricing.

Let's assume the following:

* **Free Tier:** Initially, let's say 10,000 users sign up for the free tier during the preview period.
* **Enterprise Pro Tier:** Let's conservatively estimate 1,000 users subscribing to the Enterprise Pro Tier.

Now, let's calculate the earnings:

1. **Free Tier:** Since it's free, there's no direct revenue from these users but it can serve as a good hook for the product and can be converted into leads based on the marketing team’s capability.
2. **Enterprise Pro Tier:** Each user is paying approximately $20 per month.

Total Revenue from Enterprise Pro Tier = Number of users \* Price per user per month = 1,000 users \* $20 = $20,000(Rs. 16,69,873.00) per month

So, with the provided assumptions, the company could earn approximately $20,000 per month from the Enterprise Pro Tier alone, with an estimated annual base income of 2 CR (realistically, it will be more than this as new and new paying customers may enter into the client space).

**Profit Margin Calculations:**  
Given the revenue generated from the Enterprise Pro Tier is $20,000 per month and assuming the total operational costs per user per month are $4.55, we can calculate the total operational costs for 1000 users as follows:

Total operational costs per user per month = $4.55 Total operational costs for 1000 users per month = $4.55 \* 1000 = $4550

Now, let's calculate the profit:

Total Revenue = Revenue from Enterprise Pro Tier Total Revenue = $20,000

Total Costs = Total operational costs for 1000 users per month Total Costs = $4550

Profit = Total Revenue - Total Costs Profit = $20,000 - $4550 = **$15,450**

**Annual Profit:** $15,450 \* 12 = $185,400**(Rs. 1,54,77,377.40** ≈ **1.5CR)**

Now, let's calculate the profit margin:

Profit Margin = (Profit / Total Revenue) \* 100 Profit Margin = ($15,450 / $20,000) \* 100 Profit Margin ≈ 77.25%

So, approximately, the profit margin for this revenue stream would be around **77.25%.**