Task 1….

1. Choose Two Generative AI Apps

A. OpenAI's

- **Functionality:** It's used for chatbots, code generation, writing assistance, education, customer service, and more. It takes text inputs and generates relevant text-based responses, maintaining conversational flow and context.
- **Core Features:**
    - Text-based conversation ○ Memory of context within a session
    - Language translation, summarization, and code generation

B. DALL·E

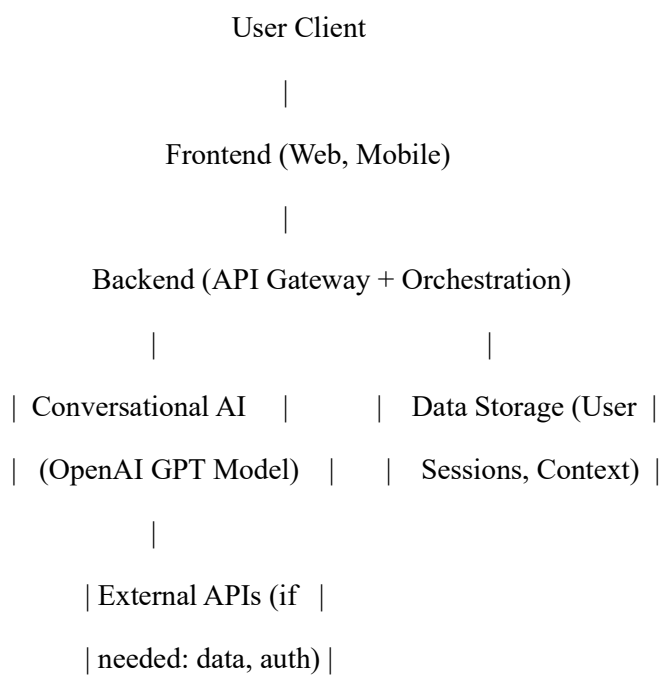
- **Functionality:** DALL·E is an image generation model that takes text descriptions and generates corresponding images. It allows users to create unique images based on detailed textual prompts. The generated images can be highly customized based on the input provided.
- **Core Features:**
    - Text-to-image generation ○ Style transfer based on user input
    - Image modification based on new text prompts

2. High-Level Architecture Design

- **Components:**
    - Frontend (Web UI/Mobile App): Users interact via text input.
    - Backend (API Server): Manages user requests, sessions, and data validation. ○ Model Layer (GPT-4 Model): Processes text inputs to generate responses.
    - Database: Logs conversations and user data.
    - Content Moderation Layer: Filters harmful or inappropriate content. ○ Cache Layer: Speeds up frequently accessed responses.

- Logging & Monitoring: Tracks API performance and issues.

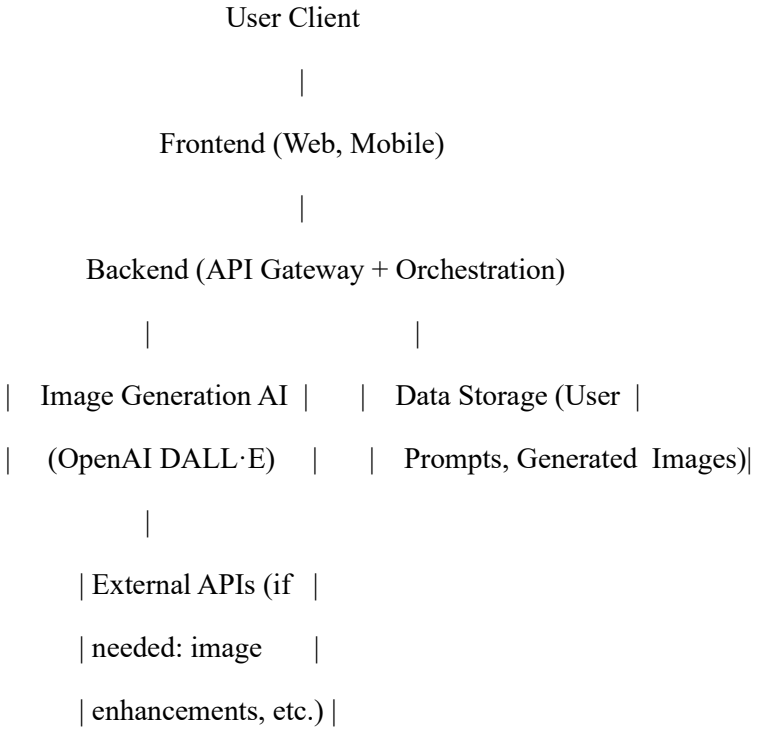
Data Flow



B. Architecture Overview for DALL·E:

- **Components:**
  - Frontend (Web UI/Mobile App): Users submit text-based image prompts. ○ Backend (API Server): Receives prompts and forwards them to the model. ○ Model Layer (DALL·E): Converts text prompts to images. ○ Database: Stores generated images, logs, and user preferences.
  - Content Moderation Layer: Scans generated images for inappropriate content.
  - Cloud Storage: Stores generated images.
  - Logging & Monitoring: Monitors service performance and logs outputs.

Data Flow



3. API Endpoint Doc

Endpoint: /v1/completions

- **Method**: POST
- **Description:** Generates a text completion based on the provided prompt.

Code

```
{
  "model": "gpt-4",
  "prompt": "Explain the concept of neural networks.",
  "max_tokens": 100,
  "temperature": 0.7
}
```

Code

```
{
 "id": "cmpl-xyz123",
 "choices": [
  {
   "text": "A neural network is a computing system inspired by the biological neural networks...",
   "index": 0,
   "finish_reason": "length"
  }
 ],
 "usage": {
  "prompt_tokens": 10,
  "completion_tokens": 90,
  "total_tokens": 100
 }
}
```

- **Endpoint: /v1/models**
  - **Method:** GET
  - **Description:** Fetches the list of available models.

Code

```
{
 "data": [
  {
   "id": "gpt-4",
   "object": "model",
   "created": 1677610602,
   "description": "GPT-4 base model"
  }
 ]
```

}

DALL·E API

- **Endpoint: /v1/images/generate**
  - **Method:** POST
  - **Description:** Generates an image based on a text description.

Code

```
{
 "prompt": "A cat playing a piano in space",
 "size": "1024x1024"
}
```

Code

```
{
 "id": "img-abc123",
 "url": "https://generated-images.com/cat-piano-space.jpg",
 "created": 1677610602
}
```

- **Endpoint: /v1/images/variations**
  - **Method:** POST
  - **Description:** Creates variations of an existing image.

Code

```
{
 "image_id": "img-abc123",
 "n": 3 }
```

Code

```
{
```

```
 "data": [
  {
   "url": "https://generated-images.com/cat-piano-space-variation1.jpg"
  },
  {
   "url": "https://generated-images.com/cat-piano-space-variation2.jpg"
  }
 ]
}
```


Task 2……

1. Backend Structure

Objective:
The backend's objective is to handle all data processing, API endpoints, and database management to ensure smooth communication between the frontend and the server.

Technologies:

- **Language**: Java (Spring Boot Framework)
- **Database**: MySQL
- **API**: RESTful APIs with JSON
- **Authentication**: JWT (JSON Web Tokens) for secure user authentication.

Structure:

- **Controllers**: These handle incoming HTTP requests. Each controller corresponds to a specific module (e.g., UserController for handling user-related actions).
- **Services**: Contain the core business logic of the application, processing data and coordinating interactions between controllers and repositories.
- **Repositories**: Communicate directly with the database, using JPA (Java Persistence API) to perform CRUD operations on the database tables.

Database Design:

- **User Table**: Stores user data, including username, password (encrypted), email, and roles (admin, user).

- **Stock Table**: Manages stock-related data, such as stock names, buy/sell/hold zones, and price ranges.
- **Recommendation Table**: Logs user recommendations, associated stocks, and uploaded research reports.

---

2. Frontend Technologies and UI Design

Objective:
To provide users with a simple, clean, and responsive interface for interacting with the application.

Technologies:

- **Framework**: React.js
- **CSS**: TailwindCSS for utility-first styling, with a focus on a modern, minimalist design.
- **State Management**: Redux for managing application-wide state.
- **HTTP Client**: Axios for fetching data from the backend.

UI Design and User Experience:

- **Layout**: Responsive layout with a mobile-first approach. Forms are designed with validation and user-friendly prompts. Input elements have rounded corners and gradient backgrounds, adhering to the design style the user prefers.
- **User Flow**: Users log in, view personalized stock recommendations, enter stock data, and upload research reports via an intuitive and fluid interface.
- **Visual Elements**: Rounded buttons, hover effects, and subtle animations ensure a modern and visually appealing user experience.

---

3. Hosting and Deployment

Objective:
Ensure a reliable and scalable environment to host the application, with continuous deployment for seamless updates.

Hosting Environment:

- **Backend**: Hosted on AWS EC2 with a reverse proxy setup using NGINX to route traffic.
- **Frontend**: Deployed on AWS S3 with CloudFront CDN for efficient static content delivery.

- **Database**: Hosted on AWS RDS (Relational Database Service) for easy scaling and management.

Deployment Process:

- **CI/CD Pipeline**: Configured using GitHub Actions. The pipeline automates building, testing, and deployment.
- **Version Control**: Git is used for source code management. Each push to the main branch triggers automated tests and deployment processes.
- **SSL & Security**: SSL certificates are applied via AWS ACM (AWS Certificate Manager) to ensure secure HTTPS communication.