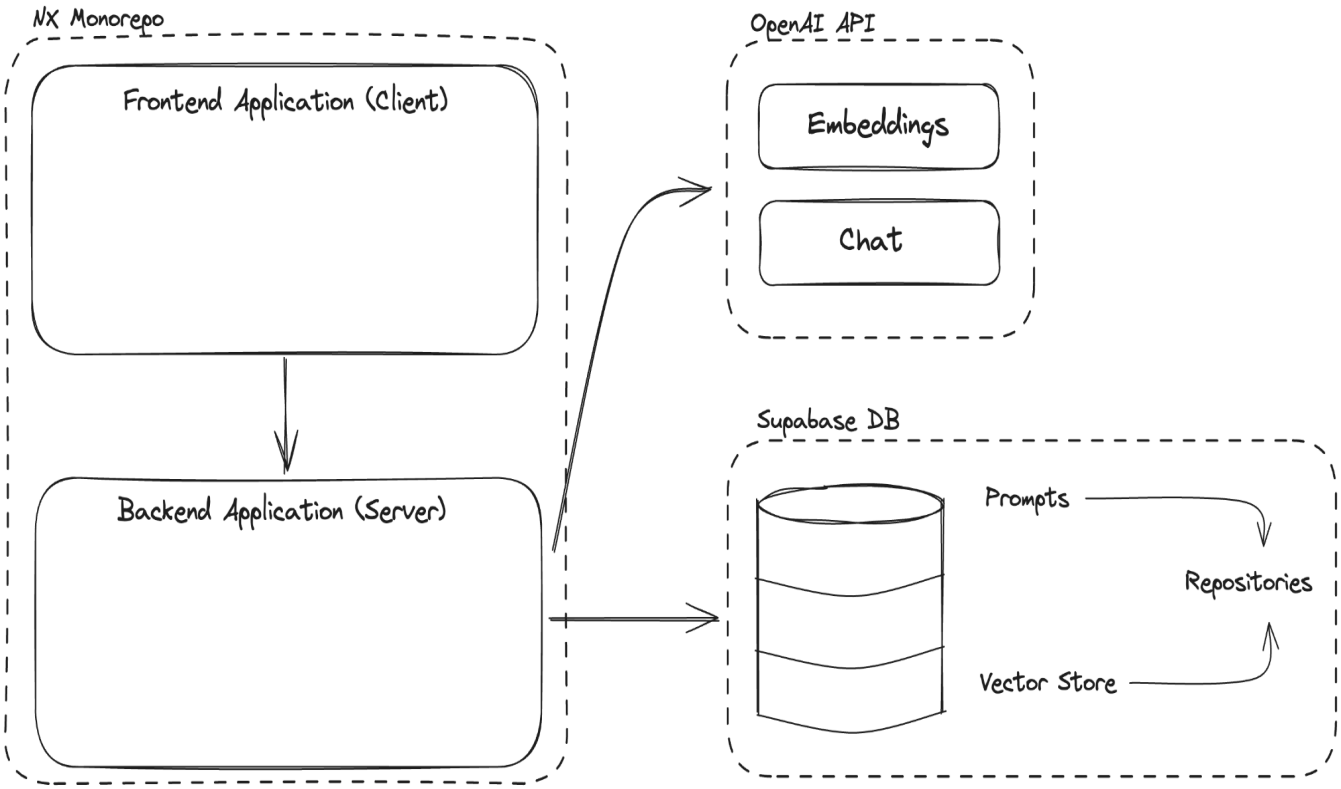
Repo Analysis Chatbot

System Design

Shai Ballali	207380379
Yarin Barnes	318851300
Noy Keren	313307316



Frontend Application (Client):

Technologies:

- TypeScript
- React

Responsibilities:

- User interface for interacting with the chatbot
- Sending user queries to the backend application (server)
- Displaying responses received from the backend
- Handling user authentication and authorization using OpenAI API keys, ensuring secure access to the chatbot's features and resources

Backend Application (Server):

Technologies:

- TypeScript
- Node.js
- NestJS
- LangChain (LLM Framework)

Responsibilities:

- Handling incoming requests from the frontend application
- Communicating with the OpenAI API for embeddings and chat functionalities
- Interacting with the Supabase database for storing and retrieving data

Key Services:

- Code Analyzer Service
  - Responsible for scanning repositories
  - Embeds the files using OpenAI Embeddings API
  - Saves the embedded files in a vector store in Supabase
- Chat Service
  - Retrieves relevant information from the vector store based on user queries
  - Generates responses to user questions using a given prompt
  - Utilizes the Retrieval-Augmented Generation (RAG) pattern and LangChain framework

Database (Supabase):

Responsibilities:

- Storing repository data, including file contents and metadata
- Storing embedded file vectors in a vector store
- Storing prompts and their rankings

OpenAI API:

Responsibilities:

- Generating embeddings for repository files using the OpenAI Embeddings API
- Providing the language model for generating responses to user queries

Relationships between Components:

1. The frontend application communicates with the backend application (server) via API calls, sending user queries and receiving responses.
2. The backend application interacts with the OpenAI API to generate embeddings for repository files and to generate responses to user queries.
3. The backend application, specifically the Code Analyzer Service, scans repositories, embeds files, and saves the embedded vectors in the Supabase vector store.
4. The backend application, specifically the Chat Service, retrieves relevant information from the Supabase vector store based on user queries and generates responses using the RAG pattern and Langchain framework.
5. The Supabase DB stores repository data, embedded file vectors, prompts, rankings, and user information.

Additional Details & Summary:

The RAG (Retrieval-Augmented Generation) pattern is employed in the Chat Service. It involves retrieving relevant information from the vector store based on user queries and using that information to augment the language model's knowledge during response generation. The LangChain framework is utilized to facilitate the integration of the RAG pattern and the interaction with the OpenAI API.

The OpenAI API plays a crucial role in generating embeddings for repository files, enabling efficient retrieval of relevant information from the vector store. It also powers the language model used for generating responses to user queries.

By leveraging the power of TypeScript, React, Node.js, NestJS, Supabase, and the OpenAI API, along with the RAG pattern and LangChain framework, The Repo Analysis Chatbot is designed to provide users with an intuitive and effective way to analyse and understand code repositories, enhancing developer productivity and collaboration.