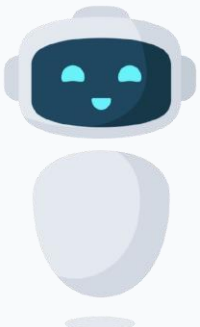
AI-Powered FAQ Chatbot Overview



An internal AI assistant that allows users to upload organizational documents (PDF, TXT, MD) and query them using natural language

> **Retrieval-Augmented Generation (RAG) Pattern:** Retrieves relevant document chunks from embeddings and generates grounded answers with citations via OpenAI GPT-4o through LangChain4j

Spring Boot + Java 21 | React + Tailwind CSS | Docker Containerization | GitHub Actions CI/CD

Use of

RAG

Retrieval-Augmented Generation

AI pattern that enhances LLMs by retrieving relevant information from external knowledge sources before generating responses. Improves accuracy by grounding answers in specific document content.

LangChain4j

Java LLM Framework

Java library that simplifies integration of large language models into Java applications. Provides tools for working with LLMs, embeddings, and vector stores in Spring Boot apps.

Docker

Containerization Platform

Platform that packages applications and dependencies into standardized containers. Ensures consistent operation across different environments and simplifies deployment to the cloud.

CI/CD

Continuous Integration/Deployment

Automated development workflow where code changes are regularly integrated, tested, and deployed. With GitHub Actions, it automates builds, tests, and deployments to the cloud.

These technologies work together to create a seamless, maintainable, and scalable FAQ chatbot system

Technology Stack

Frontend

React 18: Modern JavaScript library for building user interfaces
Tailwind CSS: Utility-first CSS framework for rapid UI development

Uses hooks only (no Redux) with custom components

Backend

Spring Boot 3.3+: Java framework for microservices
Java 21: Latest long-term support Java version

RESTful API endpoints with Swagger documentation

AI/LLM

LangChain4j: Java library for LLM application development
OpenAI GPT-4o: Advanced multimodal AI model

Handles chat generation and embeddings creation

Vector Store

InMemoryEmbeddingStore: Simple in-memory vector database
Stores and retrieves document embeddings without requiring external DB setup

Enables semantic search functionality

File Processing

Apache PDFBox: PDF text extraction
Apache POI: Microsoft doc formats

Containerization

Docker: Container platform
Docker Compose: Multi-container orchestration

CI/CD

GitHub Actions: Automates build, test, Docker image creation
Streamlines deployment to cloud environments

This comprehensive technology stack ensures a modern, scalable, and maintainable FAQ chatbot solution

Project Folder Structure

```
FAQ-ChatBot/
  backend/
    pom.xml
    Dockerfile
    src/
      main/java/com/capstone/faqbot/...
      main/resources/application.yml
  frontend/
    package.json
    Dockerfile
    src/... ← React + Tailwind
  docker-compose.yml ← runs both containers
  README.md
```

Backend

- Spring Boot 3 + Java 21
- LangChain4j with OpenAI GPT-4o
- In-memory vector store (no DB needed)
- API docs with Swagger UI

Frontend

- React 18 + Tailwind CSS
- Hooks-based (no Redux)
- Two-panel design: upload & chat
- Theme toggles & animations

Containerization

- Each service has dedicated Dockerfile
- docker-compose orchestrates deployment
- Enables cloud deployment via GitHub Actions

Clean separation of concerns with independent frontend, backend, and containerization configuration

Backend Architecture

The backend leverages Spring Boot with LangChain4j to provide a powerful RAG-based chatbot system that processes documents and generates accurate, context-aware responses.

Spring Boot 3 + Java 21

- Modern Java framework providing robust backend infrastructure
- RESTful API endpoints for document upload and queries
- Handles asynchronous document processing
- Efficiently manages API rate limits with OpenAI

LangChain4j + GPT-4o

- Java-native RAG implementation for AI integration
- Connects with OpenAI's GPT-4o model for embeddings and responses
- Manages context windows and prompt engineering
- Handles citation generation from source documents

In-memory Vector Store

- InMemoryEmbeddingStore eliminates database dependencies
- Stores document chunks with OpenAI-generated embeddings
- Performs semantic similarity search during queries
- Simple deployment with no external DB configuration

Swagger UI Integration

- Interactive API documentation at /swagger-ui.html
- Allows testing document uploads and queries
- Built with Springdoc OpenAPI
- Simplifies developer onboarding and API exploration

Frontend Architecture

React 18 + Tailwind CSS

- Modern SPA (Single Page Application) architecture
- Tailwind CSS for utility-first styling
- Responsive design for various screen sizes
- Fast rendering with React 18's concurrent features
- Lightweight build with optimized bundle size

Two-Panel UI Design

- **Left Panel:** Document upload section with drag & drop
- **Right Panel:** Chat interface with question input & response display
- Clean separation of concerns for improved UX

Hooks-Based Architecture

- Uses React hooks exclusively (no Redux)
- useState for local state management
- useEffect for API calls and side effects

Interactive Features

- **Avatars:** User and AI assistant visual identities
- **Typing Animation:** Realistic word-by-word response display
- **Theme Toggle:** Light/dark mode support
- **Upload Progress:** Visual feedback for document processing
- **Citation Highlights:** Source references in responses

The frontend is designed for intuitive interaction with the AI chatbot while providing a seamless document upload experience

RAG Pattern Data Flow

How the Retrieval-Augmented Generation process works in our FAQ chatbot system

API Endpoints & Data Models

Key Endpoints

POST `/api/documents/upload`

Upload & ingest document (PDF/TXT/MD)

POST `/api/ask`

Ask a question to the AI chatbot

GET `/swagger-ui.html`

Interactive API documentation

Data Models

QuestionRequest

String question
int topK
Input for /api/ask endpoint

AnswerResponse

String question
String answer
List<String> citations
LLM response with source citations

DocumentMetadata

String filename
int chunks
Document ingestion result details

Docker Containerization

Backend Dockerfile

```
FROM eclipse-temurin:21-jdk
WORKDIR /app
COPY target/*.jar app.jar
EXPOSE 8080
CMD ["java", "-jar", "app.jar"]
```

Builds a container for the Spring Boot backend using Java 21 JDK, exposing port 8080.

Frontend Dockerfile

```
FROM node:18 as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
```

Multi-stage build: builds React app with Node.js, then serves it with Nginx.

docker-compose.yml

```
version: '3'
services:
  backend:
    build: ./backend
    ports: ['8080:8080']
  frontend:
    build: ./frontend
    ports: ['3000:80']
    depends_on: [backend]
```

Orchestrates multi-container application, defining service dependencies and networking.

Benefits

- **Consistency**: "It works on my machine" problem solved
- **Portability**: Run anywhere Docker is installed
- **Isolation**: Separate environments for each component
- **Scalability**: Easy horizontal scaling in cloud environments
- **CI/CD Integration**: Seamless automation with GitHub Actions

Docker containerization enables consistent development, testing, and production environments

Automated Build & Deployment Pipeline

GitHub Actions automates the entire workflow from code commit to cloud deployment, ensuring quality and consistency.

1 Code & Test

- Triggered on push or PR
- Unit & integration tests
- Ensures code quality

2 Build & Package

- Spring Boot app build
- React app build
- Docker image creation

3 Deploy

- Push to Docker Hub
- Cloud deployment
- Environment variables config

Key Benefits

- Automated testing prevents bugs
- Consistent build environment
- Rapid deployment cycles
- Simplified rollbacks if needed

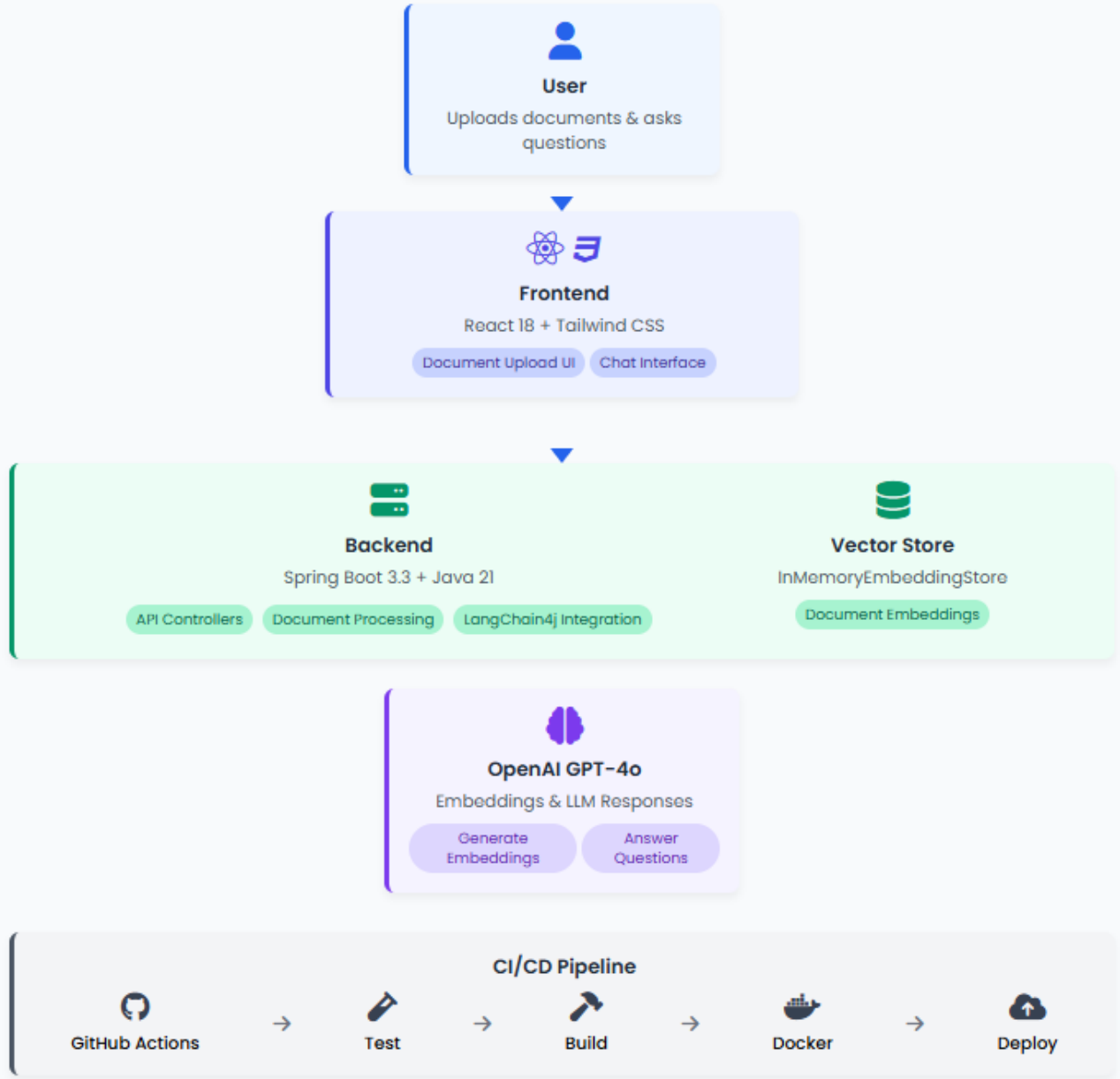
Workflow Configuration

```yaml
name: Build & Deploy
on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - checkout
      - test
      - build-images
      - push-to-registry
      - deploy-cloud
```

System Architecture Diagram

High-level overview of the FAQ Chatbot system components and interactions



Thank You!

Find us at

github.com/faq-chatbot

localhost:3000

Have questions? Our team is ready to assist you with implementation or customizations.