AllocAIte - Supporting Documentation

1. Project Overview

AllocAIte is a full stack staffing system designed to help organisations match technical employees to tasks using AI. The project is built with a FastAPI backend (Python 3.9 / PostgreSQL) and a React frontend.

Users upload an Excel spreadsheet containing employee information. The system processes the data, stores it in the database, and provides features such as:
- AI-driven task recommendations
- Availability dashboard
- Chatbot for quick queries
- Task assignment and management
- User profile and settings
- Authentication (login/register)

2. Backend (FastAPI)

2.1 Application Entry Point

backend/main.py handles the setup of the API:
- Configures CORS
- Initialises PostgreSQL connection
- Registers routers for upload, recommendations, tasks, dashboard, chatbot, settings and authentication

2.2 Database Access

backend/db.py provides:
- A get_connection() function for connecting to PostgreSQL
- Table creation and schema initialisation
- Tables include: Users, Uploads, Employees, Assignments, Recommendations, UserSetting

2.3 Upload Processing

processing/upload_processing.py processes Excel spreadsheets using pandas and openpyxl:
- Validates column names
- Extracts employee rows
- Inserts employees and assignments into the database
- Links uploaded files to Users through the Uploads table

2.4 Recommendation Engine

The recommendation engine uses Sentence-BERT to understand the meaning of task descriptions.

Key modules:

- processing/nlp/task_matching.py
Loads employees, encodes task and employee text using SentenceTransformer (all-MiniLM-L6-v2), and computes similarity, skill and experience signals.

- processing/nlp/task_scoring.py
Normalises values, applies weighted formulas and generates explanations for the ranking.

- processing/recommend_processing.py
Coordinates the process and produces the final ranked list.

API endpoints:

- /api/recommend - generate recommendations
- /api/recommend/assign - create an assignment record

2.5 Availability and Dashboard Logic

- processing/availability_processing.py calculates availability windows from assignment dates.
- processing/dashboard_processing.py summarises team workload for the dashboard view.
- These power the /api/dashboard endpoints.

2.6 Authentication and Settings

- routers/auth.py implements registration and login with SHA-256 password hashing.
- routers/settings.py handles updating theme, font size and profile data.
- processing/settings_processing.py updates the database.

2.7 Chatbot System

processing/nlp/chatbot_processing.py includes:
- Date parsing (python-dateutil)
- Rule-based intent detection
- Queries to check availability, skills or assignments
- Responses are returned through /api/chatbot

2.8 Other Backend Modules

- processing/task_processing.py handles CRUD operations for assignments.

3. Frontend (React)

3.1 Application Entry

- src/index.js loads the user's saved theme and font preferences.
- App.js defines all application routes.

3.2 Layout and Theming

- Navigation menu: Pages/Menu.jsx
- Global colour and font variables in index.css
- Page-specific CSS files such as Assignments.css, Recommendations.css, etc.

3.3 Core Pages

- Upload.jsx - uploads Excel file to /api/upload
- Assignments.jsx - form for requesting recommendations
- Recommendations.jsx - displays ranked employees and calls /api/recommend/assign
- Dashboard.jsx - shows team availability and workload
- Tasks.jsx - displays tasks from the backend
- Chatbot.jsx - sends queries to /api/chatbot
- Settings.jsx - updates user settings via /api/settings
- Login.jsx / Register.jsx - handles authentication

3.4 API Helper

src/api.js centralises all backend communication.

4. Tech Stack

Backend: Python 3.9, FastAPI, Pydantic, PostgreSQL, pandas, openpyxl, sentence-transformers,
python-dateutil, psycopg2, Uvicorn
Frontend: React, JSX, CSS modules, LocalStorage
Database: PostgreSQL with schema defined in backend/db.py

5. System Workflow (High-Level)

1. User registers or logs in.
2. User uploads an Excel spreadsheet.
3. Backend validates and stores employees, skills and assignments.
4. User requests recommendations for a new task.
5. Backend generates embeddings, scores employees and returns ranked results with explanations.
6. User reviews recommendations and assigns the task.
7. Dashboard summarises overall workload and availability.
8. Chatbot answers availability/skill queries using the same dataset.
9. User settings update both backend and frontend state.