Backend Onboarding Guide (FastAPI)

# Project Overview

The CAP Backend is a FastAPI-based backend service that powers the Client Action Plan (CAP) web application. It handles APIs for client memos, profiles, AI agent workflows, and database interactions. The backend integrates with Generative AI agents and PostgreSQL for semantic search and memo management.

# Key Features

* FastAPI for high-performance APIs
* PostgreSQL with async queries (asyncpg, sqlalchemy)
* pgvector support for RAG (Retrieval-Augmented Generation)
* Integration with Generative AI Agents
* Secure Authentication via Azure AD / MSAL
* Async background tasks
* Logging and error handling
* Unit/Integration tests

# Tech Stack

* Framework: FastAPI (Python 3.10+)
* Database: PostgreSQL (with pgvector)
* ORM/Queries: SQLAlchemy / asyncpg
* Auth: Azure AD (MSAL / JWT)
* Agents: Custom LLM agents (RAG, summarization, control, etc.)
* Testing: Pytest
* Build/Run: Uvicorn + Docker
* Task Queue: (Optional) Celery / BackgroundTasks

# Getting Started

1. Raise IDM Requests: Request access to GitHub repo, DB credentials, Azure AD client details.
2. Clone the Repo:  
    git clone https://github.com/org/repo-name.git  
    cd repo-name
3. Set up Virtual Environment:  
    python -m venv venv  
    source venv/bin/activate # Linux/Mac  
    venv\Scripts\activate # Windows
4. Install Dependencies:  
    pip install -r requirements.txt
5. Configure Environment: Create a .env file with DB, API, and Azure AD configs.
6. Start the App:  
    uvicorn src.main:app --reload --port 8000
7. Open in Browser:  
    http://localhost:8000/docs

# Project Structure

src/  
 ├── agents/ # AI agents (control\_agent, writer\_agent, etc.)  
 ├── api/ # FastAPI route handlers  
 ├── utils/ # Helpers, logging, validation  
 ├── config.py # App configuration  
 ├── sql\_queries.py # Raw SQL queries  
 ├── variables.py # Constants / env variables  
 ├── main.py # FastAPI entry point  
tests/ # Unit/Integration tests  
requirements.txt # Python dependencies  
.env.example # Example environment variables

# Development Workflow

* Branching: Use feature branches (feature/<name>, bugfix/<name>)
* Commits: Clear, descriptive commit messages
* Pull Requests: Open PRs for review before merging
* Code Reviews: Peer review for consistency and best practices

# API Design

All APIs follow REST conventions. Routes are defined in src/api/. Example:

@router.get('/memos/{id}')  
async def get\_memo(id: int, db: Session = Depends(get\_db)):  
 ...

# Testing

Framework: Pytest

Run tests:

pytest -v

# Best Practices

* Use async/await for DB & API calls
* Keep functions small and focused
* Centralize config in .env + config.py
* Log important events (logging)
* Use Pydantic models for request/response validation
* Handle exceptions gracefully with FastAPI exception handlers

# Troubleshooting

* DB errors: Check DATABASE\_URL in .env
* Auth errors: Verify Azure AD configuration
* Agent errors: Check OpenAI API key and agent logs
* Import errors: Ensure src/ is in PYTHONPATH

# Useful Commands

* uvicorn src.main:app --reload # Start local dev server
* pytest -v # Run tests
* docker-compose up --build # Run with Docker
* black src/ tests/ # Format code

# Resources

* FastAPI Docs: https://fastapi.tiangolo.com/
* SQLAlchemy Docs: https://docs.sqlalchemy.org/
* Pytest Docs: https://docs.pytest.org/
* MSAL Python Docs: https://learn.microsoft.com/en-us/azure/active-directory/develop/msal-python-overview