Technical Documentation – AssurFlow Project

1. Overview

The **AssurFlow** system is designed to automate **personalized insurance contract recommendations** for clients based on their profile data and predefined insurance offerings. The system uses multiple AI agents running in parallel and interacting sequentially, supported by a frontend for client/advisor interaction and a backend orchestrating data flow.

2. System Architecture

The system is composed of the following components:

- **Frontend (Streamlit UI)**: A web interface for user login, data interaction, and result visualization.
- **Backend (Python logic)**: A controller that triggers AI agents and manages data flow.
- **AI Agents (Claude, Mistral, etc.)**: Autonomous modules performing analysis, recommendation synthesis, validation, and email drafting.
- **Database Layer (Redshift + SQLite)**: Client data is stored and retrieved from AWS Redshift; user credentials are managed in a local SQLite database.
- **Documents**: Includes insurance contract documentation (PDF), and column explanations for model context.

3. Components Description

3.1 Frontend – frontend.py

Built using Streamlit, the frontend handles:

- **Authentication**: Login interface for users with roles (admin/client).
- **Client Dashboard**: Allows clients to view and update personal and lifestyle data.
- **Advisor Dashboard**: Provides advisors with a button to trigger the AI recommendation pipeline and view results.
- **Session Management**: Uses st.session_state for persistent user interactions.
- **Dynamic Forms**: Lets clients modify fields such as housing, employment, and health history.

3.2 Backend – backend.py

This module defines a single key function: orchestrate_agents.

It matches the given client_id against a CSV file and triggers the full agent pipeline via final_run() from ai_agents.py.

3.3 AI Agents – ai_agents.py

This module is the core of the decision pipeline and includes:

Agent A1 – Claude (Anthropic)

- Generates contract suggestions based on client data and insurance contract documentation.

Agent A2 – Mistral

- Provides an independent recommendation using a separate model.

Agent B – Consensus Synthesizer

- Compares A1 and A2 outputs.
- Resolves discrepancies and finalizes a recommendation.

Agent C – Reverse Validation Auditor

- Re-evaluates the final recommendation independently based on client data and insurance criteria.
- Confirms or proposes alternatives with justification.

Agent D – Email Generator

- Produces a formal recommendation email to be sent to an insurer.

Mail Sending

- Uses **MailHog** (SMTP on localhost:1025) to simulate email delivery.

3.4 Database Initialization – init_db.py

- Connects to AWS Redshift (clients_database table).
- Creates a local SQLite database (users.db) with user credentials and role information.
- Adds:
    - One admin account (advisor / adminpass)
    - Multiple client accounts based on the Redshift dataset

4. AI Workflow

The final_run() function performs the following sequential steps:

1. Fetches updated client data from Redshift.
2. Loads supporting documents:
    - Column descriptions
    - Insurance contract definitions
3. Builds a unified prompt.
4. Executes A1 (Claude) and A2 (Mistral) in parallel.
5. Runs Agent B to resolve their outputs.
6. Runs Agent C to validate the consensus.
7. Agent D crafts a professional message.
8. The message is sent via MailHog.

Each step uses Bedrock-hosted LLMs with appropriate prompt engineering.

5. Insurance Contracts Reference

The system uses a domain knowledge PDF with 4 categories of insurance:

- **Auto Insurance**: Basic, Comprehensive, Premium
- **Home Insurance**: Renter's, Standard, Comprehensive
- **Health Insurance**: Basic, Family, Premium
- **Life Insurance**: Term, Whole, Investment-linked

Each type includes eligibility, target demographic, and feature criteria. These are parsed from the PDF and included in the AI prompt.

6. Technologies Used

Component	Technology
Frontend UI	Streamlit

Component	Technology
Backend Logic	Python
AI/LLM Inference	AWS Bedrock (Claude, Mistral)
Database (Users)	SQLite
Database (Clients)	AWS Redshift
PDF Parsing	PyPDF2
Parallel Execution	ThreadPoolExecutor
Email Simulation	MailHog (SMTP)

7. Security and Access Control

- Only authenticated users can access dashboards.
- Clients only view/update their own data.
- Advisors can run system-wide AI recommendations.
- Login credentials are stored in SQLite (plaintext – should be encrypted in production).

8. Known Limitations

- Email delivery is simulated (no real SMTP).
- AI decisions rely heavily on prompt accuracy.
- The system currently supports batch recommendations based on client updates (Last_Account_Update = 1).
- Column documentation is expected from a missing file: Explanations_about_each_columns_of_the_clients_dataset.pdf.