UTD Career Guidance Backend Overview

Overview

The repository implements a backend-only, multi-agent workflow that produces individualized career roadmaps for UT Dallas students. A lightweight Flask server in run_demo.py exposes a single POST endpoint. Every request delegates to a CareerPlannerAgent orchestrator, which chains three specialized Claude-powered agents (job market, course catalog, and project advisor) before synthesizing the final plan. The web layer is intentionally thin; all interesting behavior lives in the agents package and the Claude client.

Request Lifecycle (Backend Only)

1. Flask creates a single CareerPlannerAgent instance when run_demo.py is imported, so Claude sub-agent instances persist along with any cached scrape results.
2. When a user submits a goal, the server calls planner.run_with_trace(goal).
3. run_with_trace orchestrates the specialized agents in sequence, passing their structured outputs forward as contextual data.
4. The method returns both the raw outputs and a narration trace that the UI can render to explain which agent produced each artifact.

Agent Orchestration Details

CareerPlannerAgent (agents/career_planner_agent.py:11) subclasses BaseAgent. It defines role-specific instructions that tell Claude to synthesize job, course, and project insights into a unified roadmap. run_with_trace coordinates the pipeline:

- JobMarketAgent.run executes first, gathering live web signals and calling Claude to produce a hiring-trend summary. Its result is captured in job_insights.
- CourseCatalogAgent.run follows, mapping the job insights (and any provided background) onto UT Dallas coursework recommendations.
- ProjectAdvisorAgent.run produces portfolio project ideas that align with the market analysis and proposed courses.
- The orchestrator sends all three summaries back to Claude (via BaseAgent.run) to generate the multi-semester career plan returned to the caller.

Each stage logs a narration entry that documents intent, completion, and any payloads. Scrape metadata from JobMarketAgent.last_scrape is also surfaced when available.

Shared Base Agent

BaseAgent (agents/base_agent.py:9) centralizes prompt construction and the claude_chat invocation. Subclasses primarily override role prompts and build_prompt. BaseAgent.run normalizes the query, serializes optional context sections, and forwards the request to Claude with consistent max_tokens and temperature defaults.

JobMarketAgent

Located in agents/job_market_agent.py, this subclass adds lightweight web intelligence before delegating to BaseAgent.run:

- _gather_web_signals scrapes the Hacker News Who Is Hiring board and the IT Jobs Watch skills table using either requests or urllib.
- Summaries of trending roles and skills are injected into the context sent to Claude, alongside scrape

diagnostics kept in last_scrape.
- build_prompt enriches the user goal with geography preferences, industries to inspect, and any pre-existing market research so Claude can produce a concise market snapshot.

CourseCatalogAgent
This agent lives in agents/course_catalog_agent.py. It consumes job_insights, student background, and degree level to instruct Claude to produce a course road map. The prompt enforces a clear output template (core courses, electives, sequencing, and gaps that need advisor follow-up).

ProjectAdvisorAgent
Defined in agents/project_advisor_agent.py, it composes project ideas grounded in the earlier outputs. Context includes job_insights, the course plan, time availability, and student background. The resulting prompt asks Claude for scoped projects, relevant datasets or competitions, showcase suggestions, and success metrics.

Claude Integration
claude_chat (claude_orchestrator.py:22) is the single Bedrock integration point. Environment variables (loaded via dotenv) provide AWS credentials and region. boto3 creates a bedrock-runtime client targeting anthropic.claude-3-haiku-20240307-v1:0. Requests follow the Bedrock-specific JSON envelope: the system prompt is sent in the top-level system field, and messages is a single user entry containing the constructed prompt. Responses stream back as JSON, and the first text block becomes the agent output. claude_client.py simply re-exports claude_chat for cleaner imports within the agents.

Utility Scripts and Data
- run_demo.py wires the agents into a Flask app (create_app) and exposes a single route for submitting goals. AGENT_META is a static mapping the UI can use to label trace entries.
- agentcore_demo.py offers a minimal standalone ping against Bedrock without the orchestrator stack, useful for validating credentials.
- agents/job_market/agent.py contains an alternate, data-centric JobMarketAgent that reasons over pre-defined job postings (data/job_postings_sample.json). It is not currently wired into the Flask flow but can support offline analysis or tests.

State and Error Handling
CareerPlannerAgent persists its sub-agents, so expensive web scrapes and caches (e.g., JobMarketAgent.last_scrape) survive across requests. Error handling is intentionally simple: the Flask layer catches exceptions from planner.run_with_trace and returns the message to the UI, while individual scrapers annotate issues in scrape_notes rather than raising.

Current Focus
The backend is fully concentrated on orchestrating Claude prompts and retrieving structured guidance. No database or long-term storage exists yet; every request is stateless beyond in-memory agent reuse. The frontend templates (ignored here) simply render the trace and outputs provided by run_with_trace.