

Google A.I Agents Capstone Project

Saul - Stack

saul.saul.stack@gmail.com

November 2025

Introduction

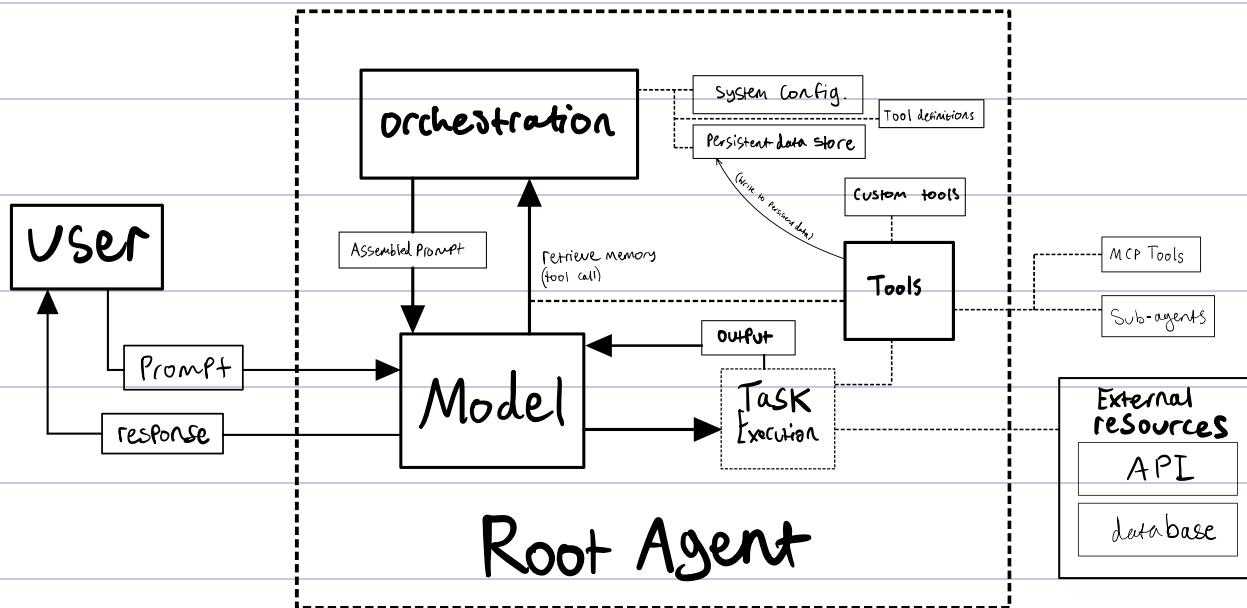
This project was completed as part of the 5-day Google A.I Agents Intensive web course, in the category of “Concierge Agents” Built using Google’s Agent Development Kit (ADK), it aims to provide users with a practical tool to manage their calendar, schedule tasks, and free up more time to improve their work–life balance.

The tool is primarily aimed at individuals who experience challenges with executive functioning—such as those with ADHD who struggle with time management, planning, or organisation. It is equally valuable for other people who simply want to create structure in their routines, build consistency around habits, or integrate long-term personal goals into their daily schedule.

I was motivated to create this system because I personally find it difficult to manage tasks that lack clear timeframes. As a result, I often miss opportunities such as keeping in touch with friends or fail to keep on top of tasks such as personal admin or maintaining long-term goals. I had the aim of creating a system that could provide value by automatically planning flexible tasks around existing commitments, and help me maintain recurring activities such as running.

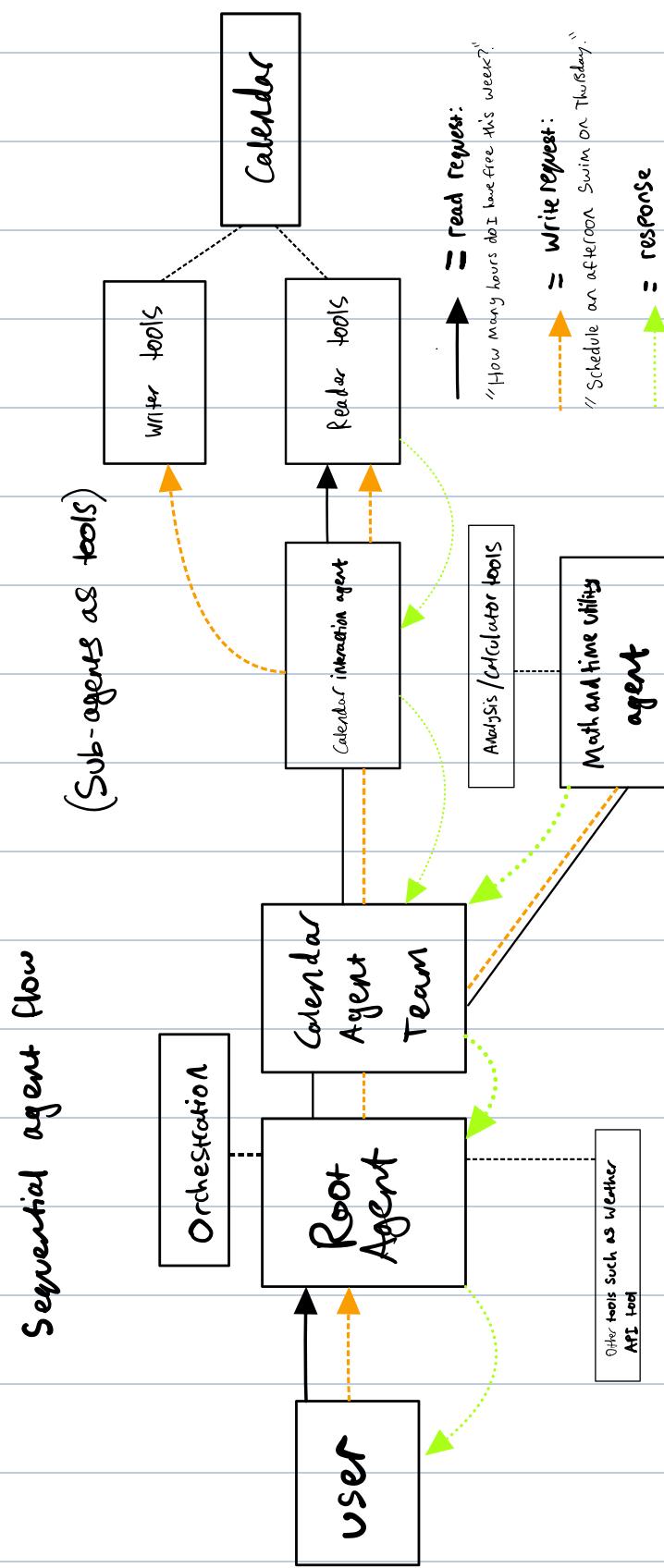
Technical Overview

Root Agent Architecture



This agent architecture follows modern design principles used in OpenAI Agents and MCP-based systems. The model is a stateless inference engine. Memory, system configuration, and tool interfaces are external and orchestrated programmatically in Python. The agent interacts with external state through tool calls, and context windows are assembled dynamically each turn.

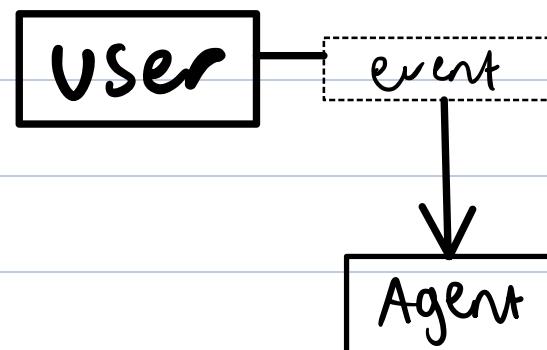
Multi-Agent Architecture and Sub-agents as Tools



High-level Event Logic Flow

Example 1- Determining Timetable Availability for a New Event Without an Explicit Time-slot

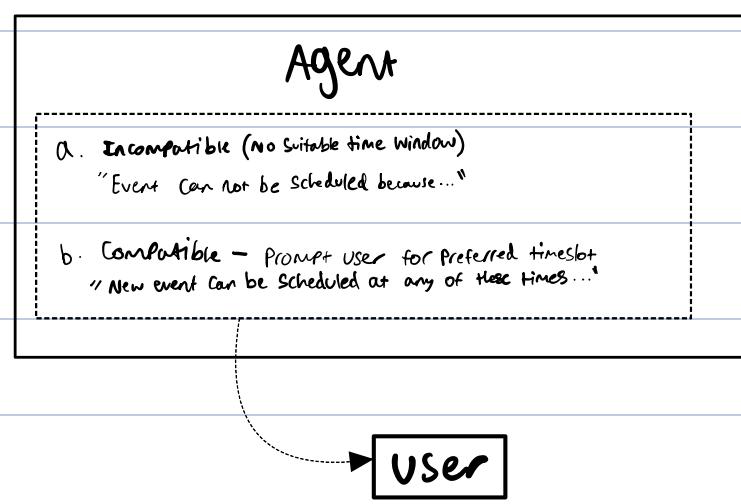
1. User asks agent to find a suitable time for scheduling a new event



2. Agent calls tools and reasons to determine timetable compatibility

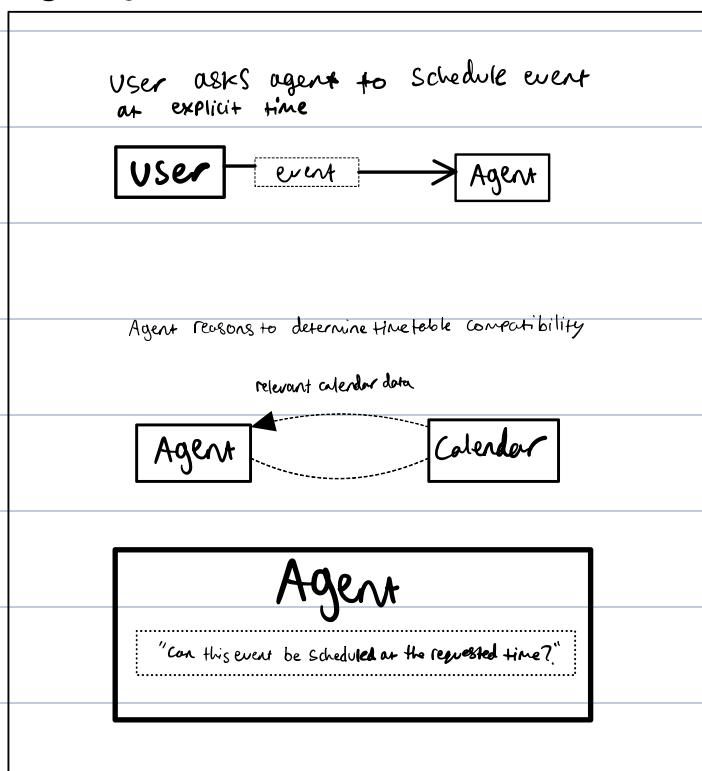


3. Agent parses result and responds to user

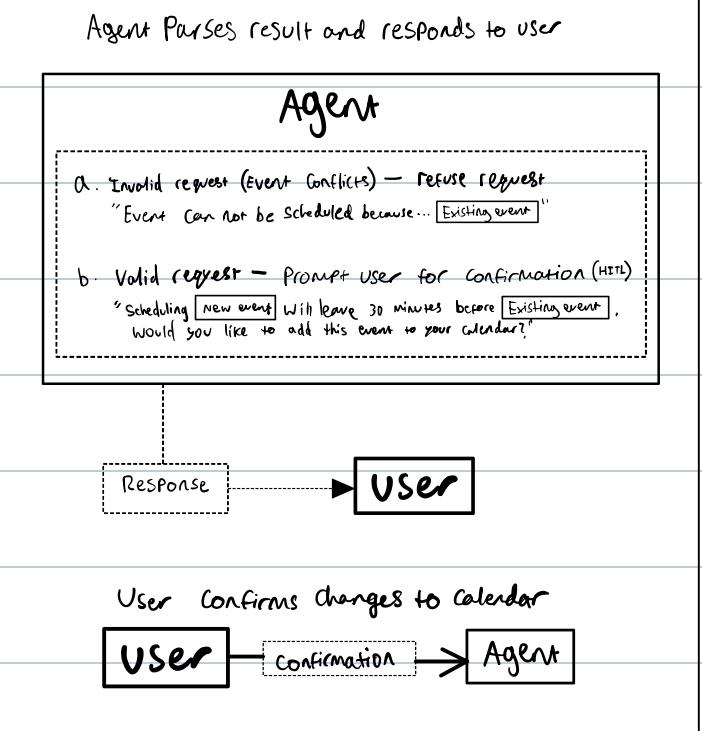


Example 2- Scheduling a New Event With an Explicit Time-slot

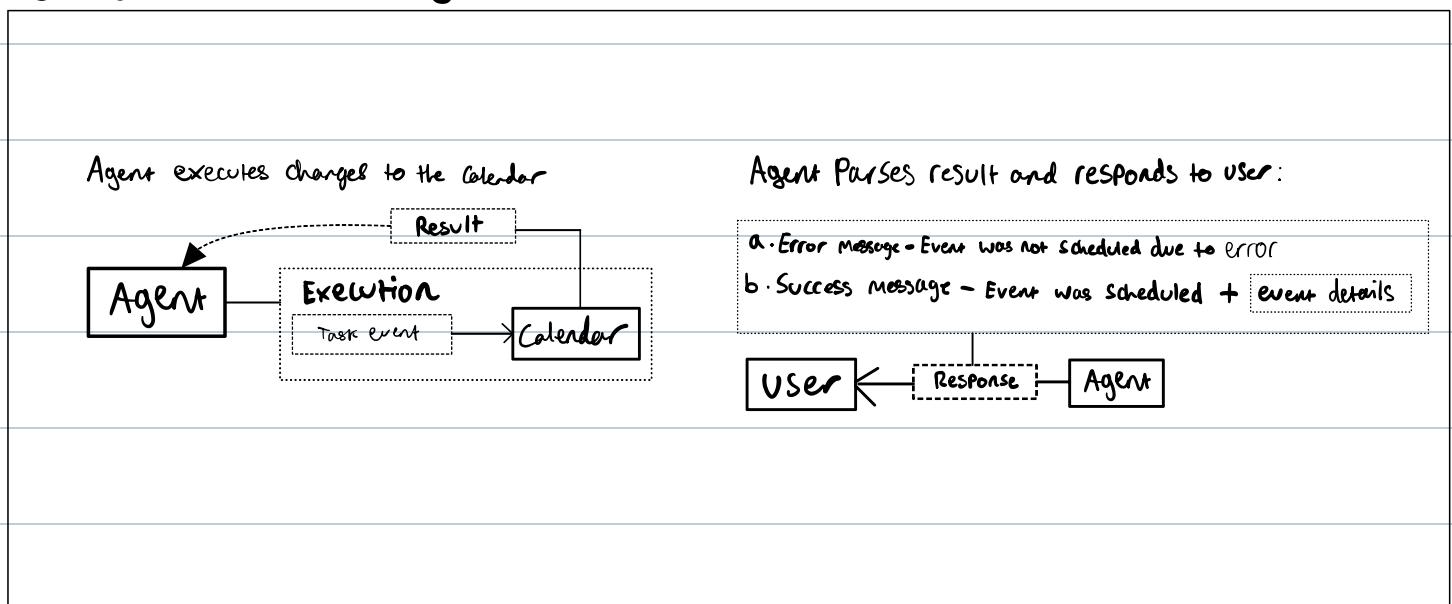
Stage 1 - Validating New event request



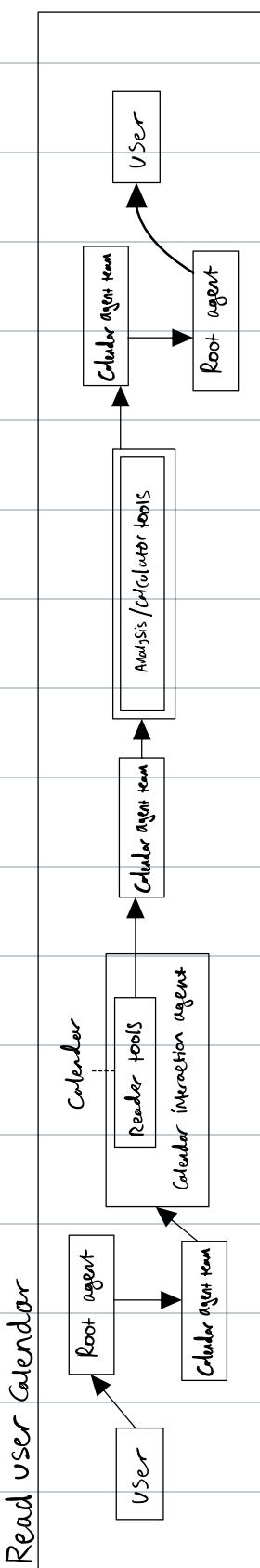
Stage 2 - User confirmation (HITL)



Stage 3 - Scheduling New event



Example 3- Calculating Sum Duration of Multiple Events



1. User prompts Model "How many hours of class do I have this week?"

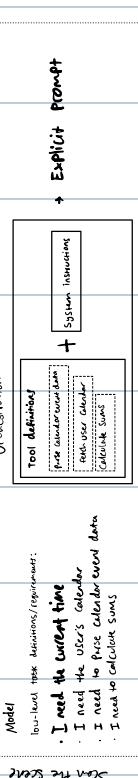
2. Model reasons to determine task and what it needs to complete task

The user wants to know the total hours of "class" scheduled in the calendar this week.
I need to:
1. Get this week's schedule from the user's calendar
2. Analyze relevant scheduled events and calculate the sum of event duration
(3. Return this information clearly to the user)

3. Orchestration layer uses tool definitions / relevant tools to assemble explicit prompt

4. Model uses tools to complete request
5. Model re-evaluates after each tool call

4. Model uses tools to complete request
 5. Model re-evaluates after each tool call
- Model determines which tools to execute next until all tasks are complete / have acquired # Now I have the relevant events data. I need to calculate the sum of hours of events x



THINK

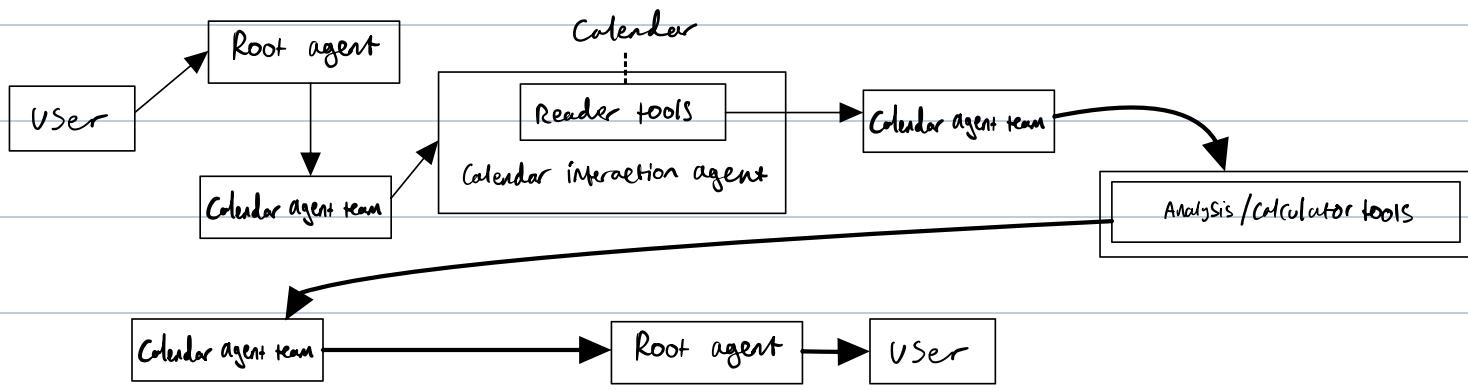
return

return

return

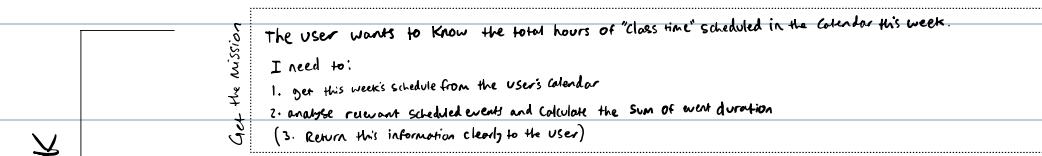
Example 3- Calculating Sum Duration of Multiple Events

Read user Calendar

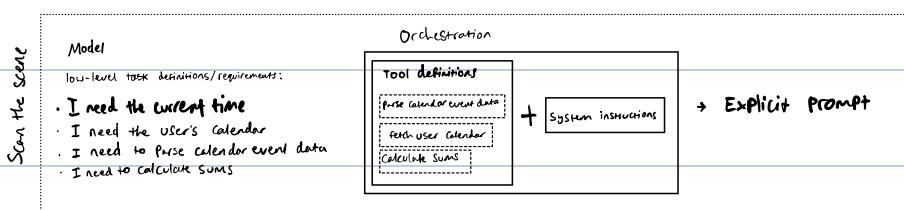


1. User prompts Model "How many hours of class do I have this week?"

2. Model reasons to determine task and what it needs to complete task

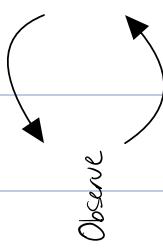


3. Orchestration Layer uses tool definitions / relevant data to assemble explicit prompt



4. Model uses tools to complete request

Explicit prompt → Model → Tool → Model



5. Model re-evaluates after each tool call

Model determines which step to execute next, until all tasks are complete / data acquired
Now I have the relevant events data. I need to calculate the sum of hours of event x

Return

6. Parse result and return to user

"you have a total of 20 hours of class this week... details..."

Software Stack

Programming Languages

- Python (3.14.0)

Development Environments

- Visual Studio Code (version?)
- Google ADK web

APIs

- Google Maps Weather API
- Google Maps Geocode API
- Google Gemini API
- Google Calendar API

Python Packages

- google-adk 1.19.0
- google-api-python-client 2.187.0
- google-auth 2.41.1
- google-auth-oauthlib 1.2.3
- google-auth-httplib2 0.2.1
- google-cloud-aiplatform 1.128.0
- google-genai 1.52.0

- requests 2.32.5
- python-dotenv 1.2.1
- dateparser 1.2.2
- isodate 0.7.2
- fastapi 0.118.3
- uvicorn 0.38.0

Development Process

Planning and Documentation

For this project to be a success, I allocated time toward planning my approach and documenting my progress. This was especially important as building my application would require the use of technologies with which I had little to no experience.

Planning involved summarizing a brief of the project requirements, and outlining the project architecture. During development, I kept track of the progression of the project. Where applicable, this involved journaling each day's work, and maintaining a dev-log on a dedicated 'docs' branch.

Setup Instructions

(See README.md)

Pre-requisites

Before you begin, ensure you meet one of the following criteria:

- a. The developer has registered your Gmail account as a test user and provided a credentials file credentials.json**
- b. The developer has provided you a service account test-credentials.json**

Also ensure that you have the following installed on your system:

- Python 3.14.0**

You will also need an API key for each of the following:

- 1. Google Gemini API**
- 2. Google Maps Weather API**
- 3. Google Calendar API**

Setting up Python environment

5. Navigate to root directory:

```
$ cd google-capstone-project
```

6. Create and activate Python virtual environment:

```
$ python -m venv virtual_env
```

```
$ source ./virtual_env/bin/activate
```

7. Install Python dependencies:

```
$ pip install -r requirements.txt
```

8. Configure environment variables

```
$ cd google-capstone-project
```

```
$ touch .env
```

Open .env and add the following lines:

GOOGLE_GENAI_USE_VERTEXAI=FALSE

ALLOW_GOOGLE_CALENDAR_WRITE_ACCESS=TRUE

GOOGLE_API_KEY= <your Google Gemini API key>

GOOGLE_MAPS_API_KEY= <your Google Maps API key>

Authorizing Google Calendar API

a. Testing with a registered Google account:

1. Copy credentials.json to google-capstone-project/setup/

2. Add the following line to .env:

USE_SERVICE_ACCOUNT=FALSE

3. Run `configure_google_calendar_api.py` and open your web browser to grant permissions to the application:

```
$ cd setup
```

```
$ python configure_google_calendar_api.py
```

b. Testing with a service account:

1. Copy `test-credentials.json` to `google-capstone-project/setup/`
2. Add the following line to `.env`:

USE_SERVICE_ACCOUNT=TRUE

Running the application

```
$ cd google-capstone-project/agents
```

```
$ adk web
```

1. Prompting the Root Agent

A screenshot of a messaging interface. The user (blue bubble) says: "I want to schedule 4 meetings tomorrow." The root agent (pink bubble) responds: "Okay, I can help you with that. To schedule these meetings, I need some more information. Could you please provide the times you'd like to schedule them, the duration of each meeting, and the names or descriptions of the meetings so I can add them to your calendar?" The user (blue bubble) says: "I'll be free from 9am. Each meeting should last 30 minutes and I will need 10 minutes break at least between meetings." The root agent (pink bubble) responds: "Okay, I will transfer you to the calendar agent to schedule these meetings."

2. Calendar Agent team receives the request

A screenshot of a messaging interface. The user (blue bubble) says: "I can help you schedule these meetings. Let's confirm the details: You're free from 9am tomorrow, you want to schedule 4 meetings, each lasting 30 minutes, and you need at least a 10-minute break between meetings. Is this correct?" The calendar agent (orange bubble) responds: "yes. The meetings are one each with Steve, Jeff, Dan and Tyler." The user (blue bubble) says: "Okay, I will schedule those meetings for you. First, I need to calculate the exact times for each meeting based on your availability, meeting duration, and break time. Then, I will schedule them in your calendar."

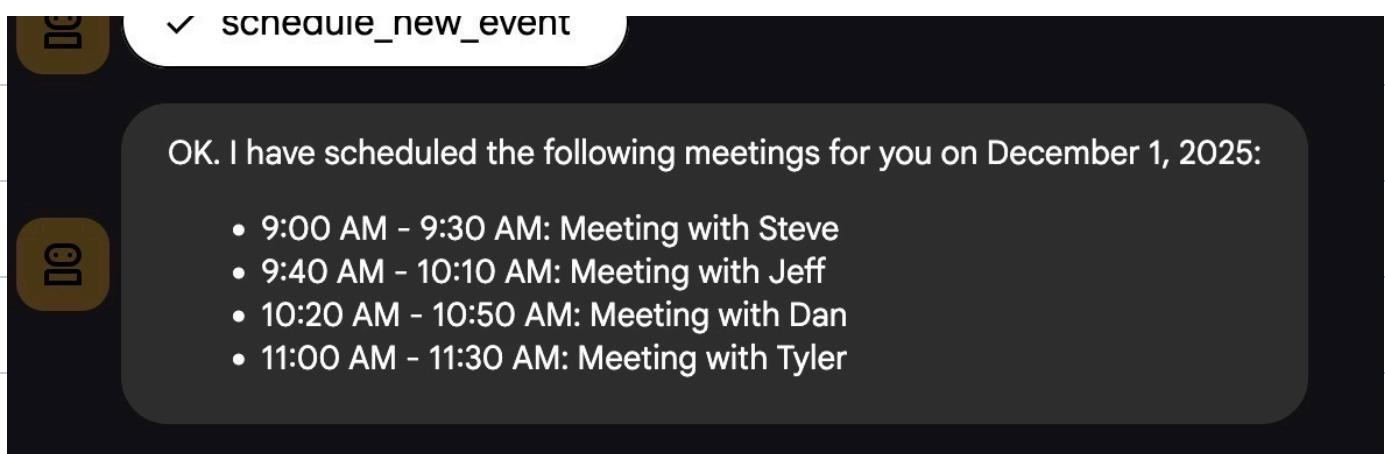
3. Math and Time utility Agent calculates event timestamps

A screenshot of a messaging interface. The user (blue bubble) says: "Okay, the meetings will be scheduled on Monday, December 1, 2025. The first meeting starts at 9:00 AM, each meeting is 30 minutes long, and there's a 10-minute break between meetings. Let me calculate the schedule:" The math/time utility agent (green bubble) lists the meeting times: "Meeting 1: 9:00 AM - 9:30 AM Break: 9:30 AM - 9:40 AM Meeting 2: 9:40 AM - 10:10 AM Break: 10:10 AM - 10:20 AM Meeting 3: 10:20 AM - 10:50 AM Break: 10:50 AM - 11:00 AM Meeting 4: 11:00 AM - 11:30 AM". The user (blue bubble) says: "I will now transfer this information to the calendar agent."

4. Calendar Agent team updates its prompt

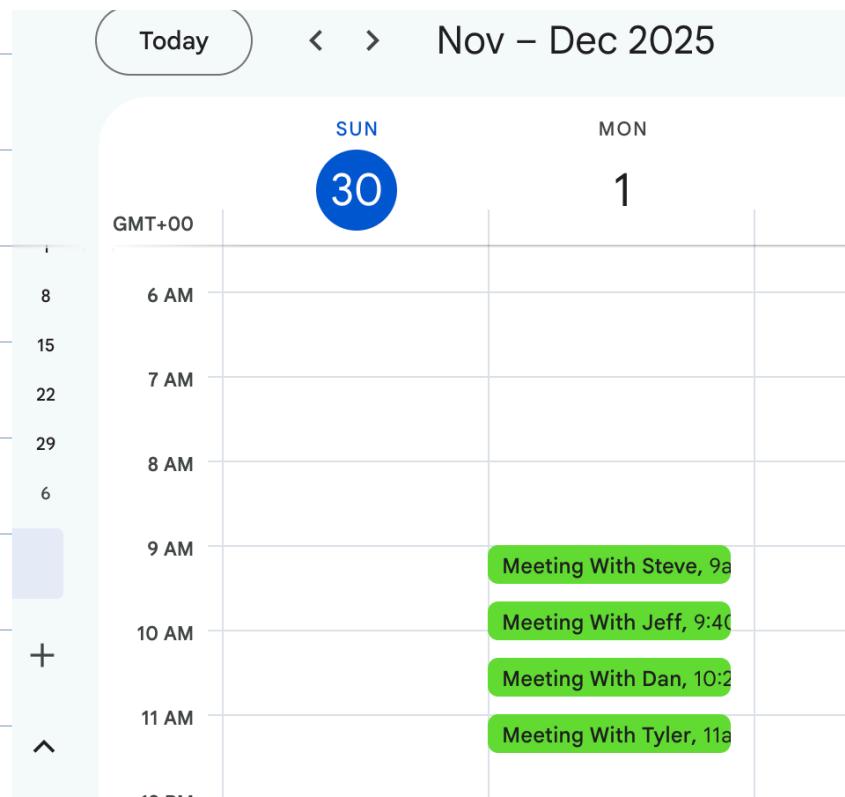
A screenshot of a messaging interface. The user (blue bubble) says: "Okay, the meeting schedule is as follows on December 1, 2025:" The calendar agent (orange bubble) lists the meeting times: • 9:00 AM - 9:30 AM: Meeting with Steve • 9:40 AM - 10:10 AM: Meeting with Jeff • 10:20 AM - 10:50 AM: Meeting with Dan • 11:00 AM - 11:30 AM: Meeting with Tyler. The user (blue bubble) says: "I will now schedule these meetings."

5. Calendar interaction agent invokes the schedule_new_event tool and responds with success message



Exam Google Calendar has been updated and now displays the new events ple

Usag



Summary and Reflection

What I learned

During this project, in addition to building applications with Agents, I gained experience including:

- Google APIs and OAuth flows
- Python, pip and virtual environments

Challenges and solutions

I found the project to be challenging, because it involved technologies and tools with which I had little to no experience. Problems I encountered include:

- Agent tool implementation (solved with clear docstrings)
- Agents stuck in loops (solved with strict instructions and programmatic tool calling)
- Sub-agents communicating directly with the user unwantedly
- Issues with Python environment and pip packages

I was able to find solutions to these problems through trial-and-error, reading documentation and browsing forums. Through solving these problems I learned skills such as managing Python environments and configuring Google OAuth flow and service account validation.

What I Enjoyed

I enjoyed learning and working with new tools. Once I had configured my initial codebase build and agent workflow, I had fun interacting with my Google Calendar, scheduling events and testing out edge cases to see how the system would react.

What I Could Improve

For future development with agents, there are many changes I would make to the development process. These include but are not limited to:

- Following a robust Test-Driven-Development approach to building my agents and tools from the start
- Using language models to evaluate the effectiveness of each agent at completing its given tasks
- Giving more thought to the appropriateness of using agents for any given task- a programmatic, deterministic approach is sometimes more suitable
- Following stricter design principals regarding orchestration and agent/sub-agent authority

A Key point for future improvement:

Many of the issues I faced during development were difficult to diagnose, because I neglected to implement an appropriate tracing and debugging workflow from the beginning stages of development.

Closing

Personally, I feel that the deadline given to complete this project was too short for me to fully materialise my idea.

Initially, I found the entire concept of agentic A.I to be challenging to grasp. During the course, I struggled to keep up with the syllabus and had some difficulty in comprehending what was being communicated, so I took time after the course had concluded to follow up with revising the subject matter. This helped me to better understand the principals, however I am certain that fully harnessing this exciting new technology

will require much more time spent reading and studying documentation, and experimenting with agentic A.I in my own projects.

Despite the challenges, I am proud of what I was able to achieve. Going forward, I plan to continue building upon this project, and further develop these ideas with the end goal of creating a fully integrated web/mobile application for users with frameworks such as ReactJS and React Native.

As a first-time project, my use of language models only scratches the surface of what this technology can be used to achieve.

I am already planning my next project.

References

- **A.I and Machine Learning Podcasts (Spotify Playlist):**

[https://open.spotify.com/playlist/6nK4fFHXO4Kxgta7nlwK14?
si=_vQqClewQJ6-GPT8rYc0Gw&pi=t8EFtqKVS4Cjv](https://open.spotify.com/playlist/6nK4fFHXO4Kxgta7nlwK14?si=_vQqClewQJ6-GPT8rYc0Gw&pi=t8EFtqKVS4Cjv)

- **MLOps Community Event:**

[https://home.mlops.community/public/events/agentsinproduction2025-
mlops-prosus?](https://home.mlops.community/public/events/agentsinproduction2025-mlops-prosus?)

agenda_day=68adc754616baff77f687bbb&agenda_track=68adc754616baff77f687bd0&agenda_stage=68adc754616baff77f687bc1&agenda_filter_view=stage&agenda_view=list

- **Google A.I Agents Intensive Course:**

[https://rsvp.withgoogle.com/events/google-ai-agents-intensive_2025/](https://rsvp.withgoogle.com/events/google-ai-agents-intensive_2025/home)

home