Project Plan - Office and Personal Assistant

Project Description

Develop an intelligent multi-agent personal assistant that seamlessly integrates work and personal life management through automated task estimation, calendar optimization, and lifestyle curation, eliminating the communication overhead between specialized agents while maintaining sophisticated functionality.

Use cases

Highest priority Use Cases:

- Basic task planning: "I need to finish the following task: Example task1."
  The agent will give the user an estimation of how long the task will take. And it will suggest calendar schedule where the user will work on that task. The user can either reject the suggestion, saying why they don't like it or they can accept it.
- Integration with a service through an mcp where the tasks are set. It takes the tasks from the server and it will give the estimation for each of them.

Necessary Use Cases

- If the user wants to watch a movie, continue reading a book, watch a youtube video or a podcast, or go to the gym, the agent can evaluate time that will be spent on a certain activity and schedule that in the calendar as well.
- Monitors user entered sites (yt channels, substack posts etc..) and when the user asks about them, it lists new content that might be interesting.
- The agent takes images or audio as input as well.

Good Use Cases to have

- The user can stop work on a certain task, letting the agent know that they are making an unplanned stop. The agent will log the task in a unfinished tasks database, as well as time remaining on that task.
- The user can continue to work on half finished tasks. The calendar agent shall find new schedules for it.
- Ability to access imdb through and mcp that will give the user
- The agent can recommend activities (if the user says I want to watch a movie, what do I watch?)
- Spawns fresh sub-agents when sub-agent context is getting close to max.
- Ability to ask questions about past tasks, past completion times, and past estimations.
- Text to Speech implementation for output

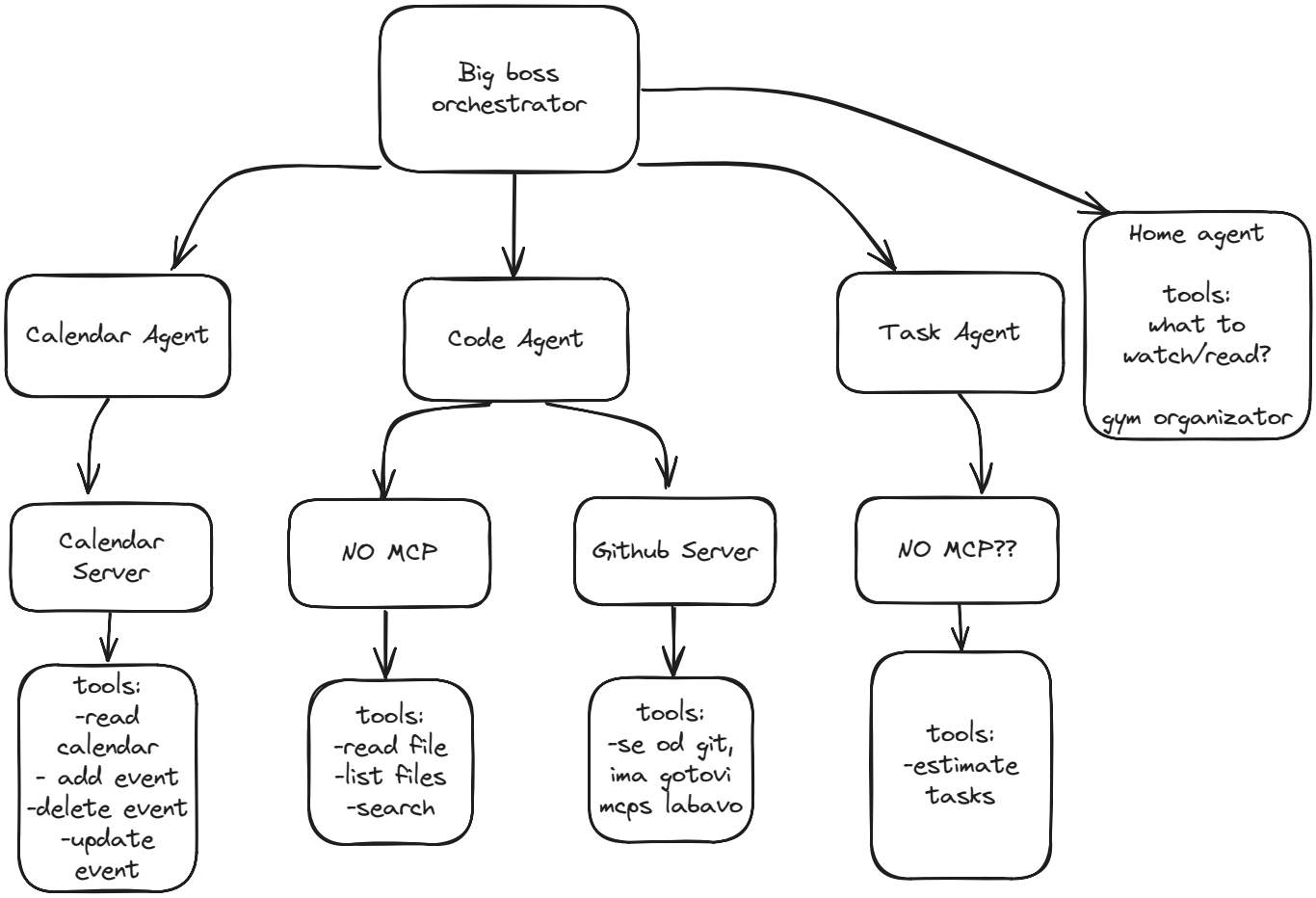
If there is time

- The agent tracks sleep. It asks the user how much he slept last night, and it logs that. If there are multiple days with little sleeping, it should warn the user to rest.
- When the token limit is exceeded it automatically summarizes conversation history
- Stores past conversations in a vector database that it can reference.

Project Architecture

Example architecture 1

This is my first iteration of the architecture, though I think I should move away from this

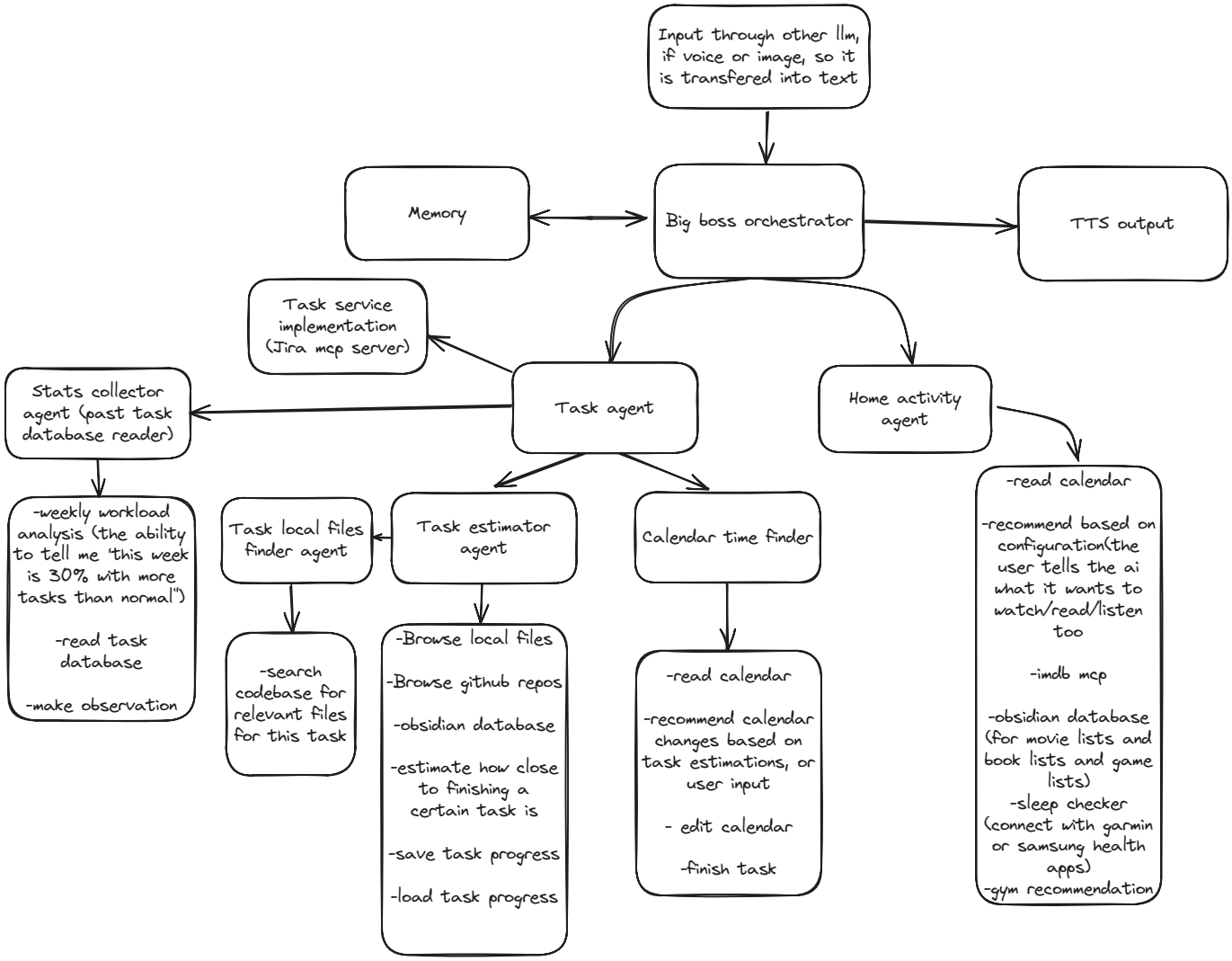


Example Architecture 2

The problem with the previous architecture is that for one task, a calendar agent, a code agent, and a task agent need to be consulted. So the outputs from the calendar agent need to go to the task agent, and so does the code agent output.

This creates a conversational overhead between agents. I am scared that it will make the app more prone to mistakes, and it will make it slower. (though I am not sure)

So the idea with this agent is to reduce that inner agent communication.



The orchestrator either calls the home agent, or the task agent. The task agent is also a multi agent.

Big Boss Orchestrator

Primary Function: Route user requests to appropriate agents with minimal overhead
Key Capabilities:
- Intent classification (work vs personal)
- Context management and summarization
- Response synthesis from agent outputs
- Token limit management and agent spawning

The Task Agent

Task Agent (Multi-Agent System)

Primary Function: Complete work-related task planning and scheduling
Architecture: Hierarchical sub-agents with internal coordination
Key Workflow: User Request/Jira mcp call output → Task Analysis → Estimation → Calendar Recommendation → Response

Task service implementation

A toolset(mcp server) that calls a task software where the tasks are put for the user.

Task estimator agent

The task estimator agent, is used for estimating how long a certain task needs to be finished. It estimates this by calling the **task local files finder agent**, searching the code base, either locally or through github, finds the relevant files, and return them to the main task estimator. The main task estimator looks at those files and sees what i have implemented vs what I need to implement. For now the plan is to just return a number by just looking at what i have in my code, but this could be enhanced by making the estimator write dummy code, and see how much it added, and estimate like that. (But I am not even sure this would be better)

Then the task and estimation is stored in a db. It is also returned to the main task agent. I have added functionality for when the user stops working on a certain task, for that task progress to be saved in a db. And for the new updated estimation (old_estimation-hours_worked) to be saved.

When the user says I want to continue my unfinished task, for that task to be loaded from a db, and to be sent to the task agent, to send to the calendar. (or I can make the ai to check for the unfinished db, and automatically warn the user, and suggest times with the calendar agent)

Finish task is when the user to says, that he finished his task, and if he says in a specific time, that time is added as real_time_finished to the db.

Calendar time finder

Calendar time finder looks at my google calendar, and finds places where to put my tasks. It can do this by getting the estimations from the task estimator agent, the database, or through user input. It will ask the user if he is happy with the changes, if he is not, the agent will suggest different options. When the user is happy, he will need to confirm that he wants the changes to be added to the calendar.

Stats collector agent

Stats Collector Agent

- Analyzes historical task completion data
- Provides insights on estimation accuracy
- Identifies productivity patterns
- Generates performance recommendations

This agent can load db where there is a task, maybe code snipets that implements important parts of the task in question, estimated time and real time finished. The idea is to make specific statements and observations about past estimations.

Home activity agent

When the user asks for time to be allocated for something personal. If he wants to watch a movie, and doesn't know what movie to watch, search from the movie list on obsidian (this is tailored towards me, for a general user it might search the web.)

The user will put youtube channels, blogs, etc. (conversations about twitter acc through apis or reddit subreddits through apis were had but i don't think those are possible. Idk about linkedin.) as things they are interested in. And the ai might recommend them to watch a new episode of a podcast. Or to read a blog post. Or the user can ask if anything new has come out from those things.

The agent should also track sleep schedule. I can add implementation to this with either garmin connect app, or simply it should ask, if it is the first conversation of the day, hey, how many hours have you slept tonight?

It should still have access to some of the same tools as calendar time find

Databases present

There need to be 5 databases:
-Chat history
-Obsidian db.
-tasks db
-tasks paused db
-sleep tracker db

How to serve the system?

Because of time constraints it might be best to stick with cli. This raises questions about the multi modal parts of the app. Specifically the stt and vision parts.

For image processing there has to be a folder where there will be images, and I will have to call some python function to process them.

For audio input a library like pyaudio would do the trick. I can capture audio while a certain keybind is pressed for example. Then that audio will be sent to a stt that will process it to text.

If I have a day or two though, I would still love to make a simple chainlit app.

Dividing the work by weeks

Week 1

Discovery

This week is used as research for the project. The idea is to find a few things:

- If I decide to use strands (Probably) I need to check documentation for multi agent implementations.
- I need to find finished google calendar calls, check how I am going to create an agent that will just recommend additions to the calendar. I need to prototype that.
- Prototype how the agent should calculate estimated task time. (Should it create dummy code or just try and eye ball it)
- Decide implementations for each of the agents. Have a detailed plan of how to implement each one of them. Prototype the most questionable ones.
- Research how to run code on the server Prof Panche gave me access to.
- Prototype some of the 'home' agent additions.
- Find a way to get as much as possible from the context. The orchestrator should remove old context if nearing limit and change it with a summarization of the same. Find online implementations of that.
- Finding if it is worth thinking about spawning multiple instances of a subagent. For example, if I say "I am bored, what should I do tonight?" Multiple home agents to spawn

and to search for an answer to this.

Week 2

Beginning implementing Agent architecture

- Should begin with google calendar agent and task estimator, file finder agent. Need to have working communication between them. (task → estimate → schedule)
- Setting up the databases.
- By the end of week 2 It is good to have every part of the task agent working correctly. Play with prompt engineering to get best results.

Week 3

Implementing home agent

The home agent should not be as difficult. So this week will be split into:

Monday, Tuesday, Wednesday: Home activity agent

- **Content Monitoring**: YouTube, blog, podcast tracking systems
- **Personal Calendar**: Integration with Google Calendar for personal events
- **Sleep Tracking**: Either Garmin Connect integration or manual input system
- **Obsidian Integration**: Personal lists, preferences, and content curation
- **Recommendation Engine**: Smart suggestions based on available time and preferences

Thursday, Friday: Implementing the context summarization and db insertion.

- **Context Summarization**: Implement intelligent conversation compression
- **Memory Optimization**: Efficient storage and retrieval of conversation history
- **Agent Spawning**: Multiple instance management for complex queries
- **Session Management**: Handle long conversations without context loss

Week 4

Room for improvement/hosting/multimodal impl

Week 4 will be left for fixing any bugs, adding any missed error handling, cli impl, to have accounts, hosting, implementing tts output and stt and vision inputs.