Project Report

Parimal Mehta

Here is the writeup for the Conversational Booking Agent Project.

Outcomes

What I accomplished.

- I built a three-party booking system involving a human user, an AI Assistant, and a Browser Agent.
- I built a Chat UI for the human user and AI assistant to talk via APIs, and I wrote a browser automation script for the AI assistant to call to book a meeting on Calendly.

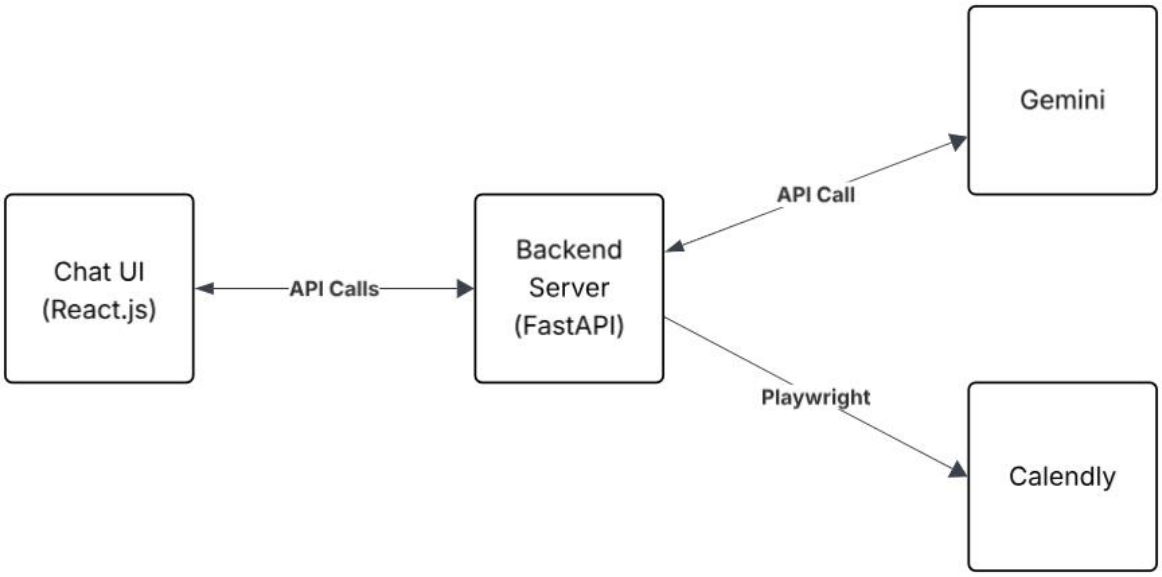
What I learned from the assessment.

- How to use the Gemini API and Playwright.
- the Calendly booking workflow.
- How Google Maps reservations, and Housecall Pro works.
- About other browser automation tools like Puppeteer, Stagehand, Browserbase, etc.
- About other AI assistants like DeepSeek, Manus, Claude, etc.

I revised:

- Building frontends with React and TypeScript.
- Building APIs with FastAPI and Python.

Architecture



System Design

Tech Stack

- TypeScript
- React
- Python
- FastAPI
- Playwright
- Gemini
- Node/npm
- Git

Why use React/Python/FastAPI/TypeScript?

- I am familiar with these technologies.
- Frontdesk uses these technologies.

Why use Playwright?

- The configuration process is more streamlined than Selenium.
- It supports multiple browsers.
- I was using Python for the APIs and Playwright has better support for Python than Puppeteer does.
- Frontdesk uses Stagehand. Since Stagehand uses Playwright, I decided to use Playwright. I did not use Stagehand because AI can be finicky, and I did not want to spend time picking it up for this project. Since I have experience with Selenium, I could quickly pick up another automation tool like Playwright.
- It is well documented.
- Why not browserless.io? – I did not want to rely on an external service. It is easier to test locally.
- It's recommended by Browserbase.

Why use Gemini?

- I was initially planning to use OpenAI until I found out it was paid.
- Out of the free options I checked, the Gemini API seems to be very well documented and had a robust free tier with a lot of features and a large request limit.

Separation of Logic

All frontend logic is within App.tsx.

A .env file in the backend folder is used to save the Calendly booking url and the Gemini API key.

Backend logic is divided into multiple files:

- main.py – Has all logic related to exposing REST APIs for the frontend to call.
- model.py – Has all logic related to communicating with the Gemini API. (dialog management)
- booking.py – Has all logic related to browser automation using Playwright to book a meeting on Calendly.

Bonus Objectives

I did not work on any of the bonus objectives.

How I would handle Bonus 1: Housecall Pro Integration

- The frontend UI does not have to change.
- The backend FastAPI server with endpoints does not have to change.
- Update model.py to define a function declaration for the Housecall Pro booking function.
- Add this function to the model configuration.
- Update model system instructions to include the ability to book on Housecall Pro and what information the AI must collect to book.
- Create a function in booking.py to do the Housecall Pro booking.
- Update get_ai_response in model.py to call the Housecall Pro booking function when the AI requests it.

How I would handle Bonus 2: OpenTable Integration

- The frontend UI does not have to change.
- The backend FastAPI server with endpoints does not have to change.
- Update model.py to define a function declaration for the OpenTable booking function and a separate function declaration for the Google Maps booking function.
- Add these functions to the model configuration.
- Update model system instructions to include the ability to book on OpenTable/Google Maps and what information the AI must collect to book.
- Create functions in booking.py to do the bookings.
- Update get_ai_response in model.py to call the booking functions when the AI requests it.

Note: OpenTable requires Email/Phone verification. Will have to investigate how to handle this. A possible approach - ask the user to provide OTP for verification. So, there will be two function calls by the AI to complete the booking (One to input the booking details, the second to verify the email/phone using the OTP and complete the reservation).

https://www.opentable.com/restaurant-solutions/api-partners/ - As an alternative to browser automation, opentable has a booking API.

https://developers.google.com/actions-center - As an alternative to browser automation, Google has a restaurant reservations API.

Challenges Faced

- UnicodeEncodeError when starting the FastAPI backend server.
- Cross-Origin Request Blocked error when communicating between backend and frontend via APIs.
- How to fetch API data and populate the frontend before displaying it.
- How to use an API key without hardcoding it or adding it to Git.
- Selecting elements uniquely and error handling with Playwright.

Future Work

1. Make the UI more presentable.
    a. Make chats from AI and the user in the frontend have different colors.
2. Clean up code and logs.
    a. Rename files, variables, and functions to be more understandable.
    b. Use a logger.
    c. Write a better return message for the root REST endpoint.
3. Testing
    a. Test out all failure flows and add error handling for: (to fail gracefully)
        i. If Gemini API is down.
        ii. If browser automation fails. (Tested)
        iii. If Calendly booking url is down.
        iv. If data passed to booking function is invalid.
        v. If booking day has no slots.
        vi. If booking time is not available, but other time slots are available on that day. (Tested)
        vii. If calendly url redirects to another url.
        viii. Should the AI assistant consider timezones?
        ix. The user should not be able to reset the AI context(send new system instructions) or get the AI to do tasks not relevant to booking a meeting. i.e The AI should not act like a cat if the user tells it to.
    b. In booking.py, throw a custom informative error message for various failure scenarios.
    c. Write automated tests.
4. Investigate and make browser automation more robust. Right now, it does not work consistently even given the same inputs.

    a. Additionally, have the AI agent try calling browser automation multiple times in case its an automation issue rather than an input data issue before responding with a failure to a user.
    b. Update browser automation to have smaller timeouts.
    c. Investigate StageHand and see if it can make browser automation simpler than directly with Playwright. This may help abstract the booking logic rather than having separate booking functions for Google Maps, Housecall Pro, OpenTable, and Calendly.
5. Rather than returning messages from the booking function directly to the user, return them to the AI, and send the final AI response back to the user.