Project Report

Parimal Mehta

Here is the writeup for the Conversational Booking Agent Project.

## Project Objective

To build a three-party booking system involving a Human user, an AI Assistant, and a Browser Agent. In this case, we are going to book a meeting on Calendly.

## Architecture

A diagram of a server

AI-generated content may be incorrect.

System Design

### Technology 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?

* It would be faster and easier to understand than Selenium.
* It supports multiple browsers. i.e a useful future skill to have.
* Frontdesk uses Stagehand. Since Stagehand uses Playwright, I decided to use it. I did not use Stagehand because AI can be finicky and I did not want to spend time learning it. Playwright would be more efficient/faster.
* It is more reliable and is well documented.
* Why not browserless.io – I did not want to rely on an external service. This is easier to test.

#### 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

### How would I 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 would I 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 separation 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.

## Future Work

1. Write automated tests.
2. Make the UI more presentable.
3. Clean up code and debug logs.
4. Rename files, variables, and functions to be more understandable.
5. Make chats from AI and the user in the frontend have different colors.
6. Test out all failure flows or add error handling for:
   1. If Gemini API is down.
   2. If browser automation fails.
   3. If Calendly booking url is down.
   4. If data passed to booking function is invalid.
   5. If booking day has no slots.
   6. If booking time is not available, but other time slots are available on that day.
   7. If calendly url redirects to another url.
   8. Should the AI assistant consider timezones?
   9. 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.
7. Write a better return message for the root REST endpoint.
8. Use a logger.
9. In booking.py, throw a custom error message for various failure scenarios.
10. Update browser automation to have smaller timeouts.
11. Investigate and make browser automation more robust. Right now, it does not work consistently even given the same inputs.
12. 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.
13. Investigate StageHand and see if it can make browser automation simpler than directly with Playwright.