MAC - the medical appointment coordinator (A chatbot prototype)

Karen Gaffney 1.20.26

Why use a chatbot for medical appointment scheduling?

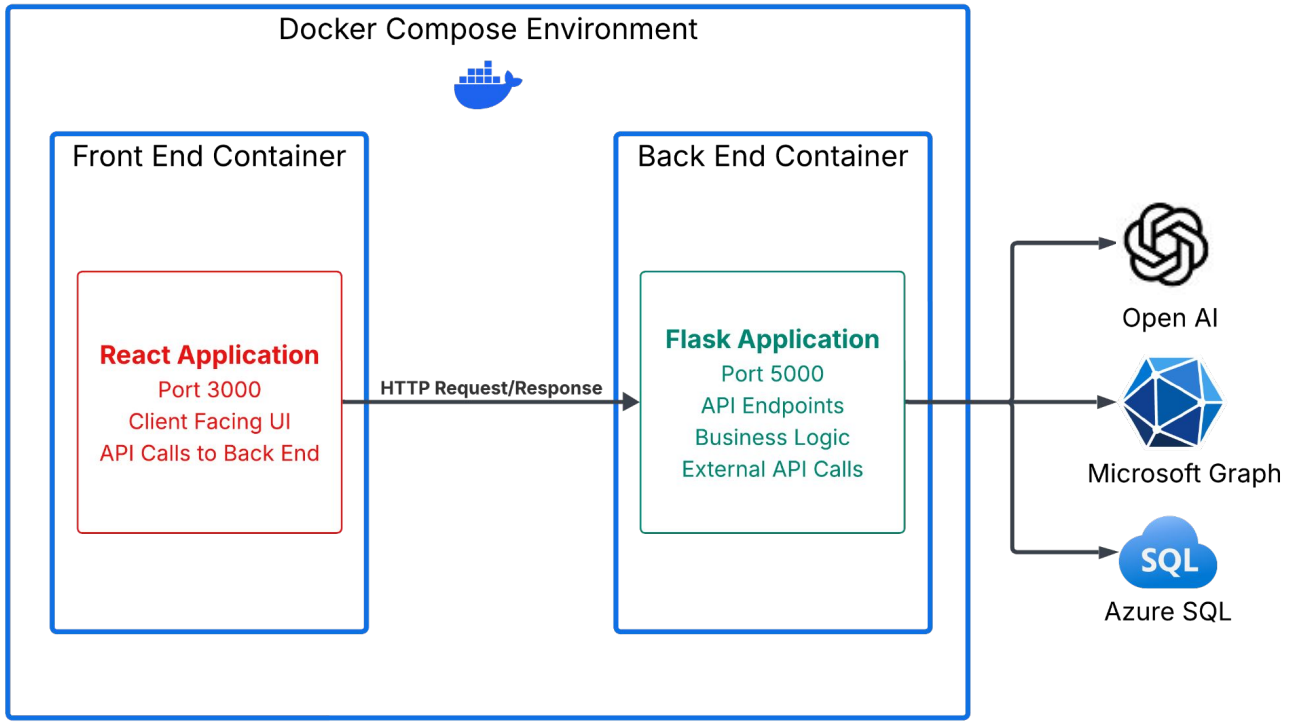
- **Single interface** to schedule appointments, no need to navigate a menu
- **Relative intent** Chatbots understand “next tuesday”
- **Data and Insights** within the chat “when was the last time Mary came in for an exam?”
- **Accessibility** allows voice to text integration can offer hands free scheduling for busy doctors

Hi I’m Mac! (Medical Appointment Coordinator)

A Chatbot for **Doctors** to schedule appointments with their Patients

- Schedule appointments
  - Capture intent, gather required information, and then ask for confirmation before scheduling the appointment
  - Patient email is retrieved from SQL database by name
  - Redirect to add_patient if patient is not in database
- Adds new patients
  - Capture intent, gather required information, and then ask for confirmation before adding a patient to the patient database

Tech Stack

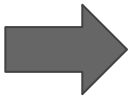

Docker Compose Environment
Front End Container
React Application
Port 3000
Client Facing UI
API Calls to Back End
HTTP Request/Response
Back End Container
Flask Application
Port 5000
API Endpoints
Business Logic
External API Calls
Open AI
Microsoft Graph
SQL
Azure SQL

LLM workflow Trial and Error 1

1. Prompt AI with Doctor's message and system prompt to **return structured json** containing appointment details
    a. Doesn't see previous messages, all information must be in one prompt

System Prompt

User Message

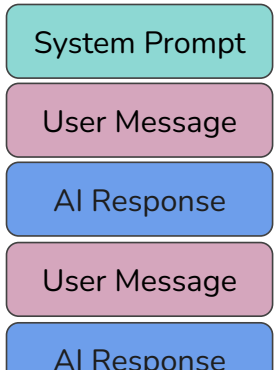


```
{
  "assistant_message": string,
  "draft_event": {
    "attendee_name": string | null,
    "start_time_local": string | null,
    "duration_minutes": number | null,
    "title": string | null,
    "notes": string | null
  }
}
```

LLM workflow Trial and Error 2

Prompt AI with a history of Doctor messages, AI response, and system prompt to **return structured json** containing appointment details. Doesn't ask for already provided info

- If all required info is present, overwrite ai response with predefined confirmation message, execute event when confirm_intent = True
- Breaks json structure after 3~4 messages (style drift)



```
{
  "assistant_message": string,
  "draft_event": {
    "attendee_name":
              ...
    "notes": string | null
  },
  "confirm_intent": "yes" | "no" | "unknown"
}
```

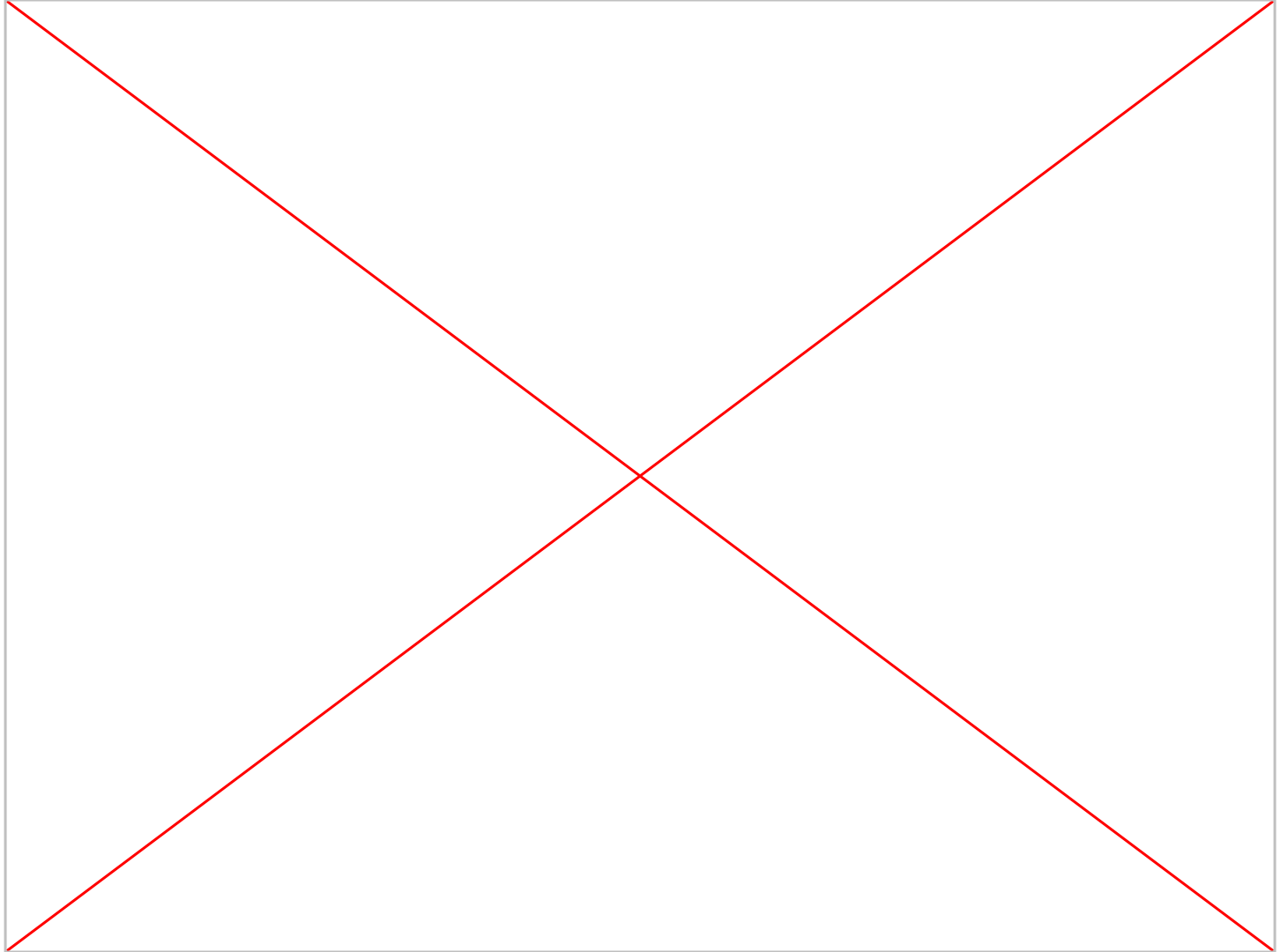
Final LLM-assisted workflow

1. **State based updating**. Front end generates a **unique session ID** and sends it with every Doctor's message. Session ID retrieves an existing state or initializes a new state.
2. AI receives the **current state** + **latest Doctor's message**
   a. **State** tracks
      i. Current mode [schedule/add]
      ii. Awaiting user confirmation
      iii. Draft event
   b. **AI returns**
      i. Assistant message
      ii. Updates to draft event
      iii. Changes in intent from current state
      iv. Confirmation intent

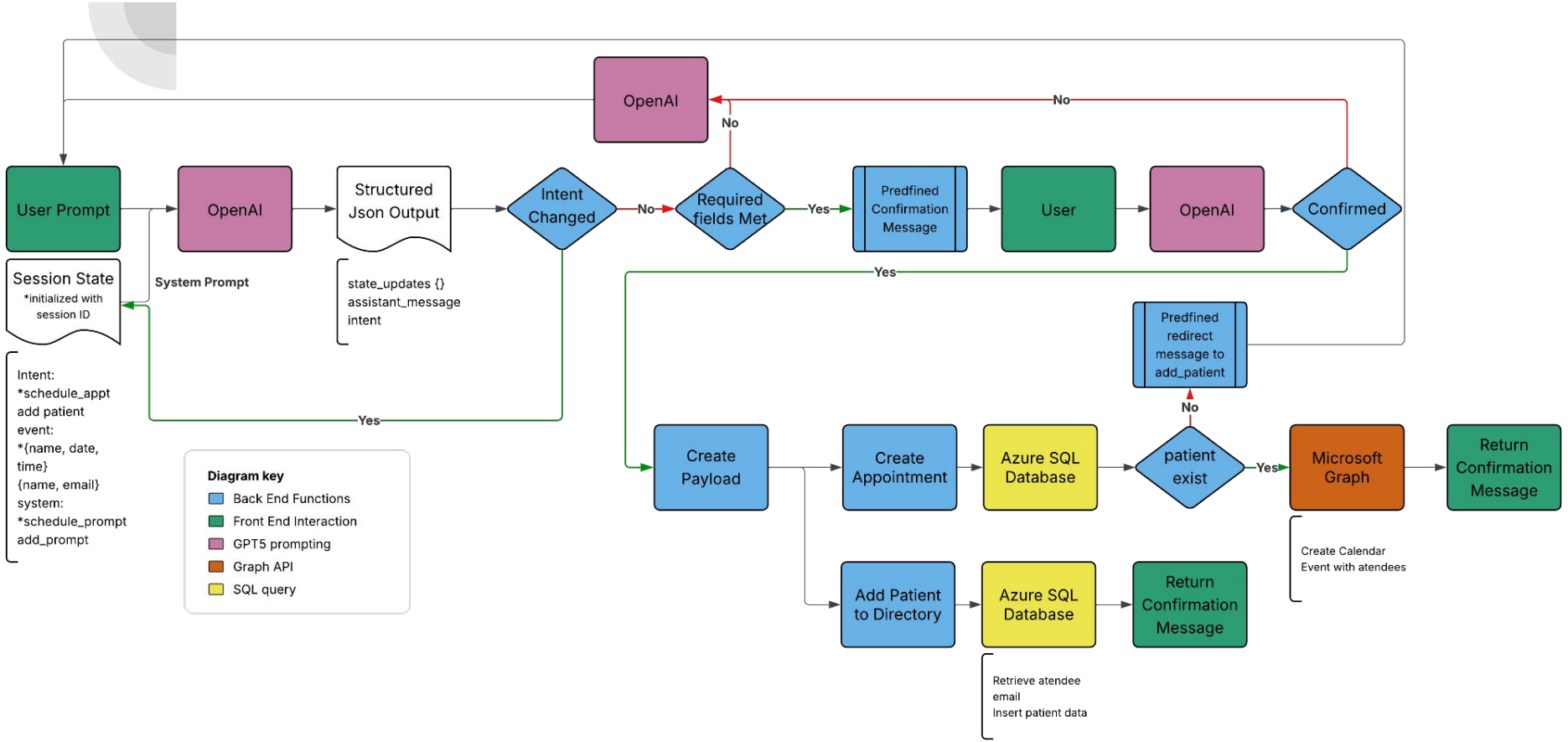
```
{"draft_event": {
"attendee_name": None,
...
"notes": None},
"awaiting_confirmation": False,
"mode": "schedule",
"pending_patient": {
 "name": None
...
 "dob": None
 }}
```

```
{
  "assistant_message": string,
  "updates": {
    "attendee_name":
              ...
    "intent": null | schedule | add_patient
  },
  "confirm_intent": "yes" | "no" | "unknown"
}
```

Demo Video



System Design



Strengths of Design

- Avoids hallucinated actions - events must be awaiting confirmation and be confirmed
  - **Awaiting_confirmation** = set by back end logic when all required fields are filled
  - **Confirm_intent** = intent parsed by LLM on user response to confirm
- State object and updates workflow allow for clearer debugging
- Makes conversations resumable and able to switch between intents
- Scales cleanly to new intents and workflows
  - Add new intent options beyond add_patient and schedule

Testing and Error Handling

- Test routes for Microsoft Graph API functions
  - Manually entered payloads and events, verified execution in Microsoft Calendar
- Logging of state and LLM response for each unique session
  - Draft event
  - Awaiting confirmation status
  - LLM raw response
    - Update json
    - Confirm_intent
    - Assistant message
- Future testing for handling non json responses, connection timeouts to SQL server

Future Extensibility

- Calendar conflict handling
  - Give the chatbot a view of the Doctor’s calendar in a X day window to offer alternative times should there be a conflicting meeting
- Login for multiple Doctors (HIPPA privacy considerations)
  - Relational Database Schema would allow for Doctors to schedule appointments with their patients only, and retrieve information about their patients only
  - Login would use microsoft entra authentication to meet security standards
- Salting and hashing sensitive patient data - DOB, SSN etc
- Front End Features
  - Database view on past appointments (notes)
  - Calendar view on scheduled appointments
  - Database view on patient data with filtering based on Logged In Doctor

Prototype vs Production

- LLM
  - Using GPT 5 mini for speed and latency
  - Evaluate multiple models for different tasks. Add RAG augmented models for domain specific tasks. Semantic Kernel for schema stability
- Secrets
  - .env, loaded in to container at runtime
  - Migrate secrets to Azure Vault
- Identity/Tenant management
  - Single tenant support for Azure Graph
  - Multi-tenant architecture with client Entra ID
- State management
  - In memory session by front end random UUID. Past states retained indefinitely for debugging (till container is reloaded) <-memory leak
  - Move session state to Azure Cache. Implement TTL-based expiration
- Data Layer
  - Azure SQL + mmsql for query execution
  - Azure SQL + mmsql for transactional work. Databricks/Fabric for analytics, reporting, cross-tenant insights
- Deployment
  - Locally hosted with manual firewall IP access to SQL server
  - Deploy to Azure with CI/CD pipelines and comprehensive tests. No public IP firewall rules, Azure integrated networking

Collaboration with others

- User specifications and feedback
  - Doctor input on prioritization of features
  - Testing of real world performance
- Division of work
  - Front End UI
    - Pages for calendar view, patient data view
  - Back End
    - New intents + actions. System prompts, external API integration, build event payloads
  - Database
    - Relational Schemas for patient/doctor. New tables to store past patient notes
- IT Security and HIPPA standards
  - Microsoft Entra ID integration for more doctors
  - Sufficient encryption of patient data

Thank you! Any Questions?

MAC is a example prototype that shows properly structured AI workflows can translate **human intent** into **real-world actions** without sacrificing security or transparency