CA

CareAI

Conversational appointment scheduler — Documentation and developer guide

Version 1.0
Author: Aditya A.

Overview

CareAI is a chat-first appointment scheduling assistant built with Node.js and OpenAI. It supports booking, checking, rescheduling, cancelling appointments, basic urgency routing, and stores data in JSON files for a simple demo setup.

Key Features

- Multi-turn slot-filling booking flow (saves fields across messages).
- Check appointments by name + phone.
- Reschedule & cancel using appointment ID.
- Urgency detection (red-flag symptoms → escalate).
- Default doctor handling & doctor name normalization (shows "Dr.Xyz").
- Fallback AI extraction for free-text booking inputs.

Conversation Flow

The booking flow uses multi-turn slot filling. The server asks for missing required fields and stores partial slot data (called `prevData`) so later replies can reference prior inputs.

```
User: "book"
Bot: "Please provide your name, phone, specialty, preferred date (YYYY-MM-DD), and time."
User: "aditya andhalkar, 7887547146, general checkup, 2025-08-10, 3 pm"
Bot: "So you want to book with Dr.Aditya Andhalkar for aditya andhalkar on 2025-08-10 at 3 pm. Proceed?"
User: "yes"
Bot: "Booked — ID: abc12345"
```

Intents & Entities

Primary intents:

- **book_appointment** — Entities: patient_name, phone, email, specialty, doctor_name, date (YYYY-MM-DD), time_of_day.
- **check_availability** — Entities: date, time, doctor.
- **reschedule_appointment** — Use appointment ID or details to find & update booking.
- **cancel_appointment** — Use appointment ID to cancel.
- **ask_instructions** — Location, documents required, consultation fees.
- **confirm_details** — yes/no confirmations.

API / Integration

Main endpoint for chat integration:

Endpoint	Purpose
`POST /chat`	Accepts JSON: `{ message, context, prevData? }`. Responds with `{ reply, context, prevData? }`.
`GET /`	Health-check (returns running message).

Important: when doing multi-turn flows the client should preserve and send back `prevData` returned by the server each turn. If the frontend does not send `prevData`, the server will not have the earlier collected slots and will say it "lost" booking details.

Example request (booking slot flow)

```
POST /chat
Content-Type: application/json

{
  "message": "My name is Aditya Andhalkar, phone 7887547146, general checkup on 2025-08-10 at 3 pm",
  "context": "booking_slot_filling",
  "prevData": { "patient_name": "Aditya" }  // optional, server merges it
}
```

Data Model

Bookings and users are stored as JSON objects (demo persistence).

```
{
  "id": "abc12345",
  "name": "Aditya Andhalkar",
  "phone": "7887547146",
  "email": "adi31820040gmail.com",
  "specialty": "general checkup",
  "date": "2025-08-10",
  "time_of_day": "3 pm",
  "doctor_name": "Dr.Aditya Andhalkar"
}
```

Backend Design & Key Functions

The demo backend is a Node/Express app that contains:

- Intent detection (keyword-based) for quick context switching.
- Slot-filling function (uses OpenAI to extract fields when user input is free text).
- Doctor normalization logic: converts "dr aditya" or "doctor aditya" → "Dr.Aditya Andhalkar" (default).
- Persistence: `bookingData.json` and `usersData.json`.
- Simple urgency check that detects red-flag phrases and escalates.

Why saving partial data matters

During booking, the server returns `prevData` to the client after each partial extraction. The client must send that same `prevData` in the next POST so server can merge and remember previously collected fields. If the client does not send it back, the server has no memory and will show "lost booking details".

Two options to fix "lost data" problem

1. **Client-side prevData (current demo):** The front-end keeps `prevData` returned from the server, and sends it with each subsequent request. This is stateless on server and simple to support across multiple servers, but the frontend must be implemented correctly.
2. **Server-side sessions (recommended for production):** Create a server-side session or persist partial slot state keyed by a user id / session token. Each incoming message includes a session id that server uses to load & update session data. This avoids relying on the client to send prevData every time.

Troubleshooting: "Lost booking details"

Symptom: Bot asks to confirm then on "yes" replies "Sorry, I lost your booking details. Please provide info again."
Root cause: server expects the client to send the saved `prevData` back when user confirms. If the front-end didn't keep and resend prevData (or the API call didn't include it), the server has no slot data to finalize booking.
Fixes:

- Make sure the client stores the `prevData` object returned by the server during the booking flow and includes it in the confirm request. Example confirm request payload:

```
POST /chat
{
  "message": "yes",
  "context": "confirm_booking",
  "prevData": {
    "patient_name": "Aditya Andhalkar",
    "phone": "7887547146",
    "specialty": "general checkup",
    "date": "2025-08-10",
    "time_of_day": "3 pm",
    "doctor_name": "Dr.Aditya Andhalkar"
  }
}
```

- Or switch to server-side sessions: generate a session token, return it to the client at start of booking, and save partial data on server keyed by that token. On confirm, server reads session data and completes booking.

Install & Run (Quick)

```
# 1. install dependencies
npm install express cors body-parser openai

# 2. create .env with OPENAI_API_KEY
OPENAI_API_KEY=sk-...

# 3. run server
node server.js
```

Files used: `server.js`, `bookingData.json`, `usersData.json`.

Future Improvements

- Server-side sessions or DB (Mongo/Postgres) for robust state management.
- Integrate with Google Calendar / hospital calendar for real-time availability.
- OTP-based user verification and secure storage for PII (GDPR/HIPAA considerations).
- Automated reminders via SMS/WhatsApp/email.
- Voice agent (Vapi or Twilio) and multi-language support.

Appendix — Helpful snippets

1) How the client should preserve prevData (example)

```
/* After a server response during booking you receive: { reply, context, prevData }
Save prevData in your frontend (memory, UI state or local storage).
When user replies "yes" to confirm, call:

fetch('/chat', {
  method:'POST',
  headers:{'Content-Type':'application/json'},
  body: JSON.stringify({
    message: 'yes',
    context: 'confirm_booking',
    prevData: savedPrevData
  })
})
```

2) Quick server-side session pattern (concept)

```
// Generate a session token (UUID) and store partial slots in memory or DB keyed by token.
// Return token to client; client sends token on every request.
// Server merges new fields into session store and persists.
```