**Capstone Travel Booking — Project Documentation**

**Overview**

A travel booking application built with Angular (frontend) and Spring Boot microservices (backend). The system supports flight search via API Gateway, multi-passenger booking, and a simulated payment workflow using Kafka.

**Prerequisites**

- Java 17+

- Node.js 18+ and npm

- Maven 3.9+

- Kafka & ZooKeeper (localhost:9092)

**Architecture**

- API Gateway (Spring Cloud Gateway MVC) — port 8080

- Eureka Service Discovery — port 8761

- Config Server — port 8888

- Flight Data MS — port 8081

- Booking MS — port 8082

- Payment MS — port 8083

- Angular frontend — dev on 4200 (connects to http://localhost:8080)

**Services & Ports**

- Gateway: 8080

- Eureka: 8761

- Config Server: 8888

- flight-data-ms: 8081

- booking-ms: 8082

- payment-ms: 8083

- Angular app: 4200

**Gateway Routes (Dev)**

- http://localhost:8080/flight-data-ms/... → http://localhost:8081/...

- http://localhost:8080/booking/... → http://localhost:8082/...

- http://localhost:8080/payment-ms/... → http://localhost:8083/...

**Service Interactions**

- Frontend → Gateway → flight-data-ms / booking-ms / payment-ms

- booking-ms ↔ Kafka (topics T1/T2) ↔ payment-ms

**Swagger & Eureka**

**- Swagger (direct):**

- Flight Data MS: <http://localhost:8081/swagger-ui/index.html>

- Booking MS: <http://localhost:8082/swagger-ui/index.html>

- Payment MS: <http://localhost:8083/swagger-ui/index.html>

**- Swagger (via Gateway):**

- Flight Data MS:<http://localhost:8080/flight-data-ms/swagger-ui/index.html>

- Booking MS: <http://localhost:8080/booking/swagger-ui/index.html>

- Payment MS: <http://localhost:8080/payment-ms/swagger-ui/index.html>

- Eureka dashboard: <http://localhost:8761>

**Backend APIs**

**Flight Data MS (8081)**

- GET /flights/search?origin=City&destination=City&date=YYYY-MM-DD → 200 Flight[ ] | 404

**Booking MS (8082)**

- POST /booking/create?flightId=ID — body: Passenger[ ] → 201 Booking

- POST /booking/payments?bookingId=ID → 202

- GET /booking/bookings/{id} → 200 Booking

- GET /booking/bookings/{id}/status → 200 { status }

- GET /booking/search?origin&destination&date → 200 FlightDto[ ]

**Payment MS (8083)**

- Kafka consumer/producer for payment simulation

**Data Models**

Passenger

{

"fullName": "John Doe",

"dateOfBirth": "1990-05-10",

"gender": "Male",

"email": "john@example.com",

"phoneNumber": "9876543210",

"address": "...",

"passportId": "P1234567",

"specialRequests": "Wheelchair"

}

Booking

{

"id": 1,

"flightId": 1001,

"status": "initiated|pending|successful|failed",

"passengers": [ Passenger, ... ]

}

**Frontend**

**Key Routes**

- / — Home (search)

- /flight-results — Results list

- /booking — Multi-passenger booking

- /payment — Payment

- /confirmation — Booking summary and ticket

**Environment & API Paths**

- Base: http://localhost:8080 (gateway)

- Endpoints: /booking/search, /booking/create, /booking/bookings/{id}, /booking/bookings/{id}/status, /booking/payments

**Workflow**

- Search: UI → API Gateway → booking-ms → flight-data-ms → booking-ms → UI

- Booking: UI → API Gateway → booking-ms → UI (booking created)

- Payment initiation: UI → API Gateway → booking-ms → Kafka T1

- Payment processing: payment-ms consumes T1 → produces T2 → booking-ms updates status

- Status check: UI → API Gateway → booking-ms (returns status)

Run Locally

**Backend**

- eureka-server (8761)

- api-gateway (8080)

- flight-data-ms (8081)

- booking-ms (8082)

- payment-ms (8083)

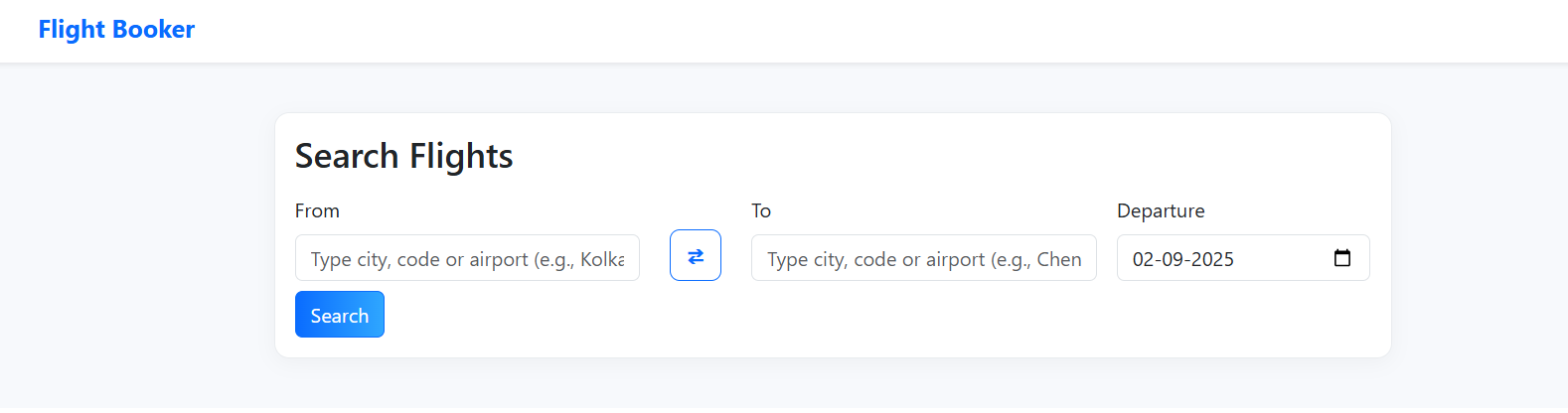
- Kafka & ZooKeeper (localhost:9092)

**Frontend**

- From frontend/travel: npm start

1) Home — Search

- Screen1:



- Purpose: Entry point to search available flights.

- Inputs/Controls: Origin, Destination, Journey Date (local date format), Search action.

- Validation: Non-empty origin and destination; valid date (not in the past).

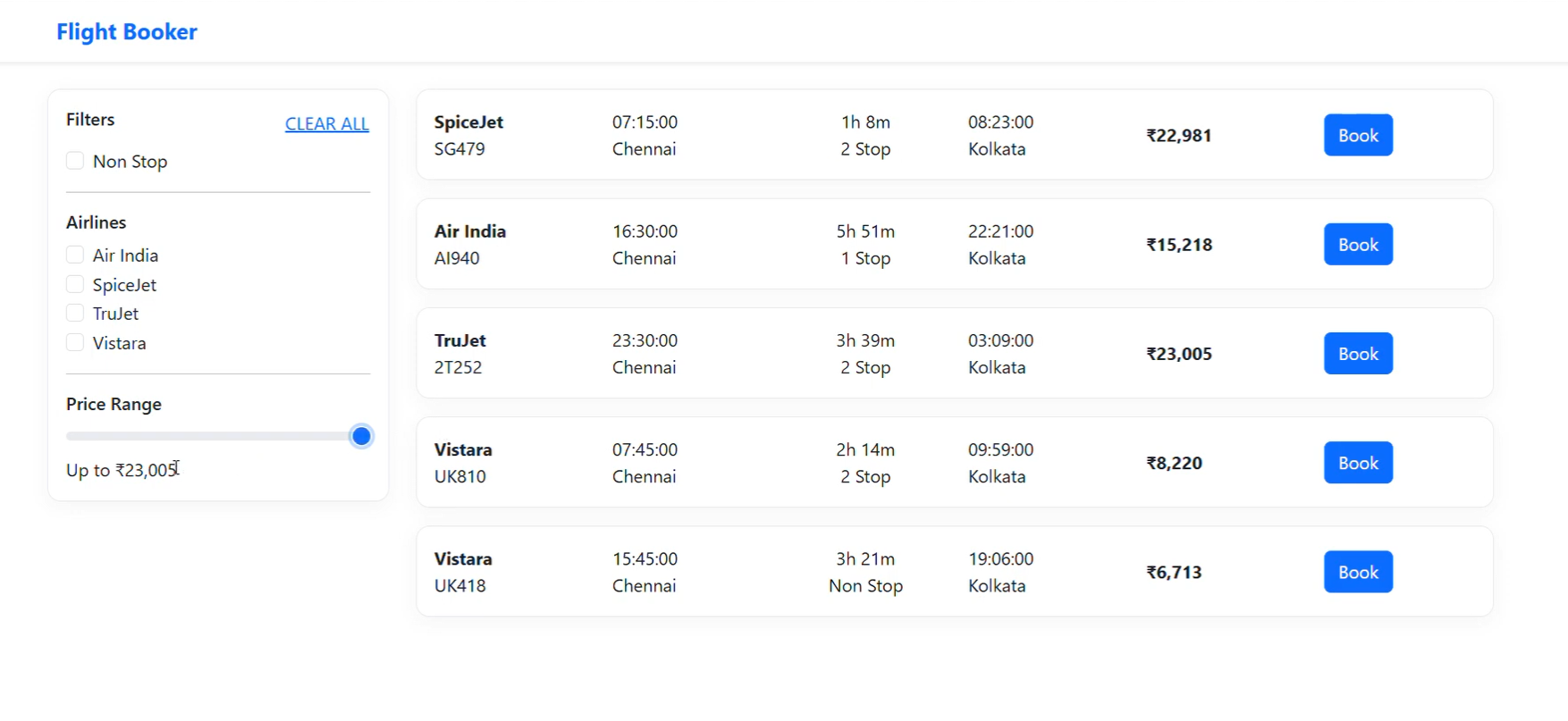
- Data: City/typeahead options; date is captured in local format.

- Actions: Search navigates to Flight Results with selected criteria.

- Navigation: Home → Flight Results.

2) Flight Results

- Screen2:



- Purpose: Display available flights for the selected criteria.

- Inputs/Controls: Optional sorting/filtering (e.g., price, duration, stops), Book action per flight.

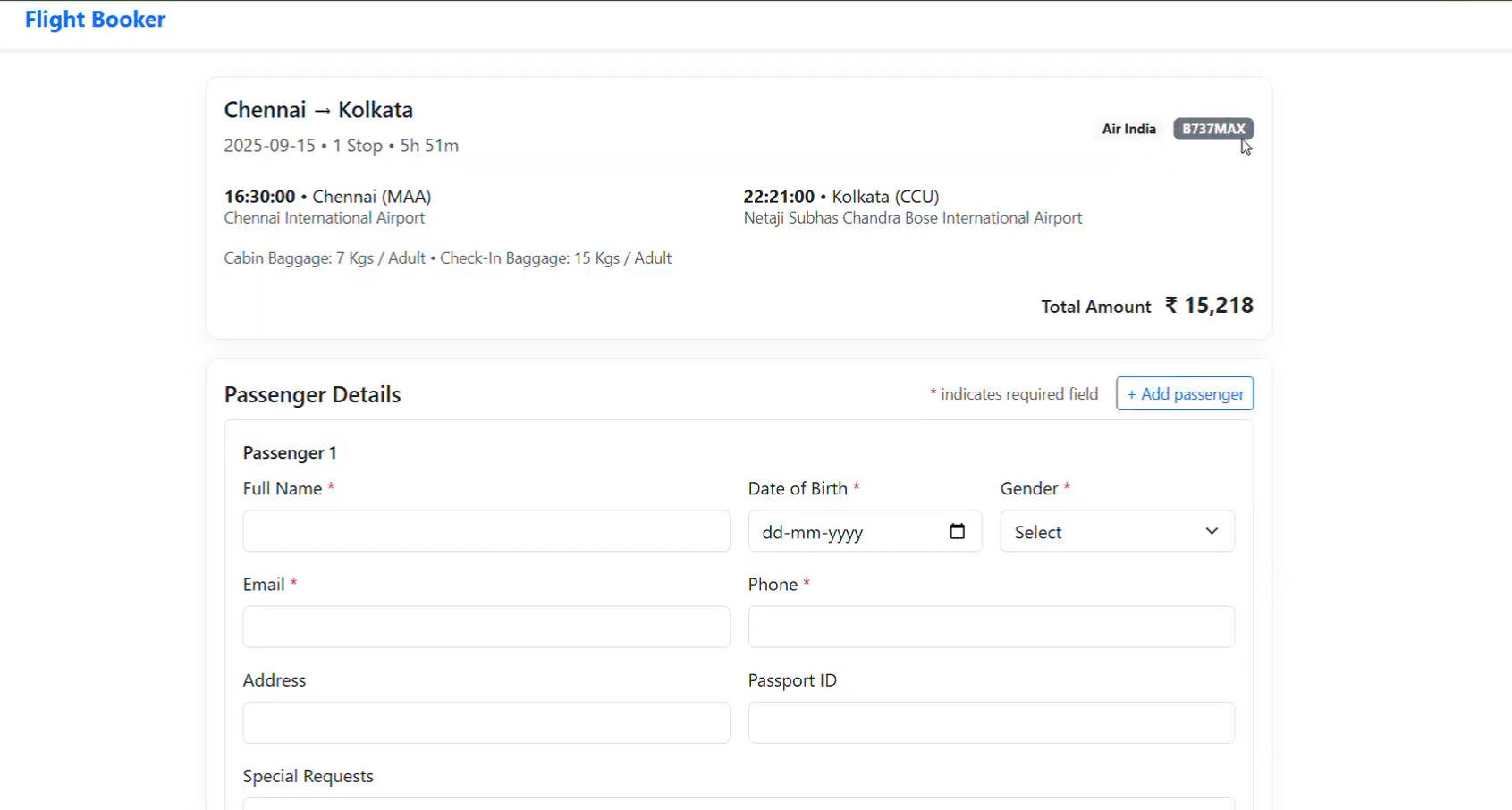
- Data: Loaded via GET /booking/search through API Gateway.

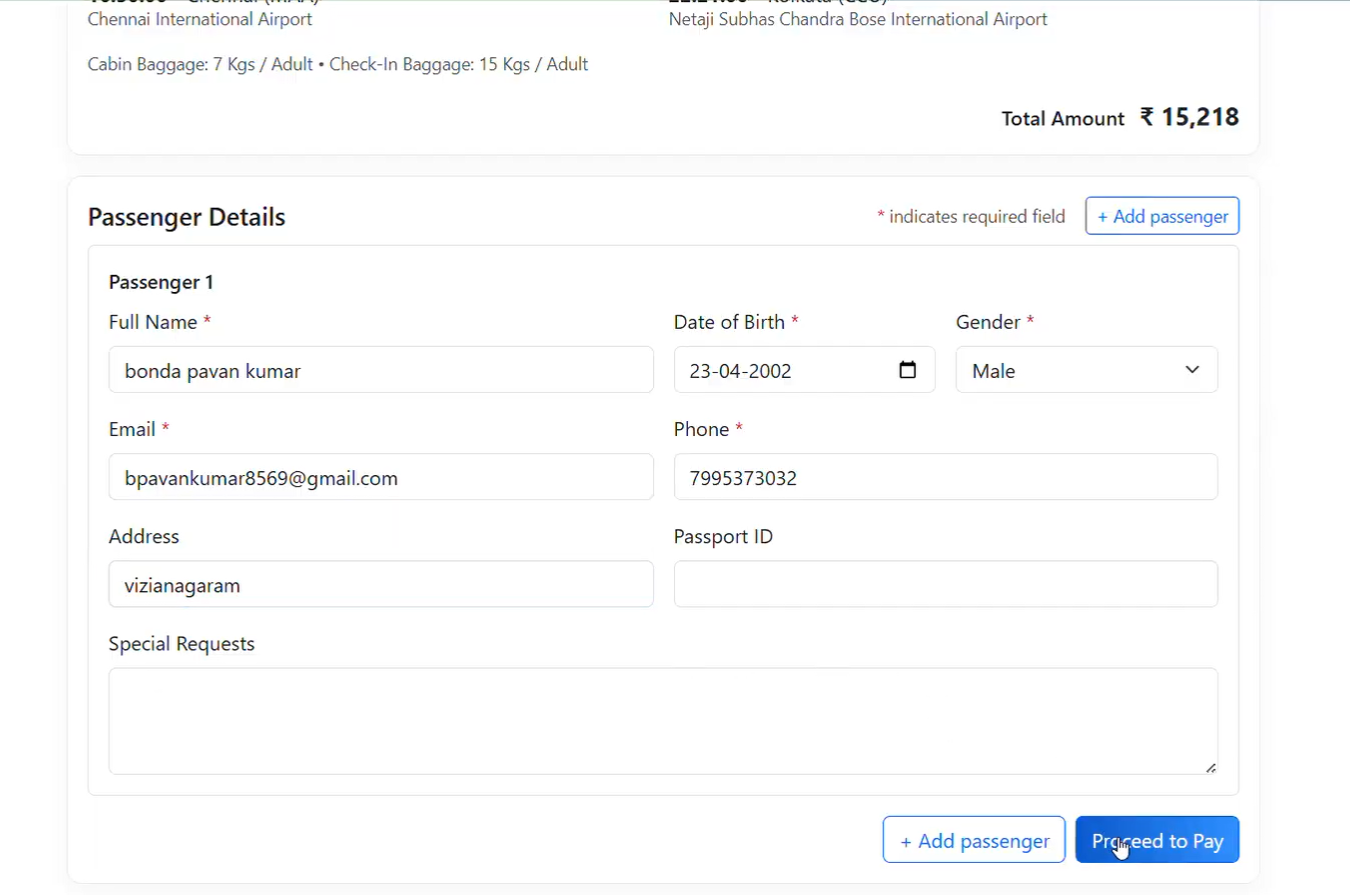
- Actions: Book selects a flight and proceeds to Booking.

- Navigation: Flight Results → Booking.

3) Booking (Multi-passenger)

- Screen3:





- Purpose: Capture traveller details for one booking.

- Inputs/Controls: Per-passenger fields — full name, date of birth, gender, email, phone; optional: address, passport ID, special requests. Add/Remove passenger controls (max 6).

- Validation: Per-passenger validation for date of birth (valid date), email, and phone; required fields enforced.

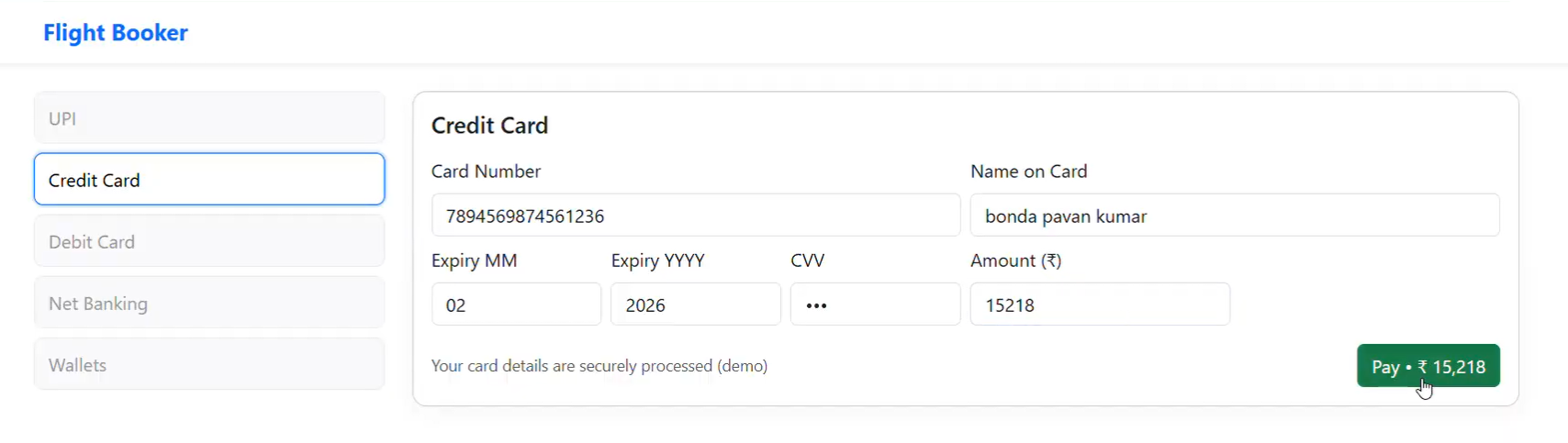
- Data: Passenger[] is prepared for submission; DOB captured in local format.

- Actions: Proceed creates the booking via POST /booking/create?flightId with body Passenger[].

- Navigation: Booking → Payment.

4) Payment

- Screen4:



- Purpose: Capture payment details and initiate payment processing.

- Inputs/Controls: Cardholder name, card number, expiry, CVV (simulation); Pay action.

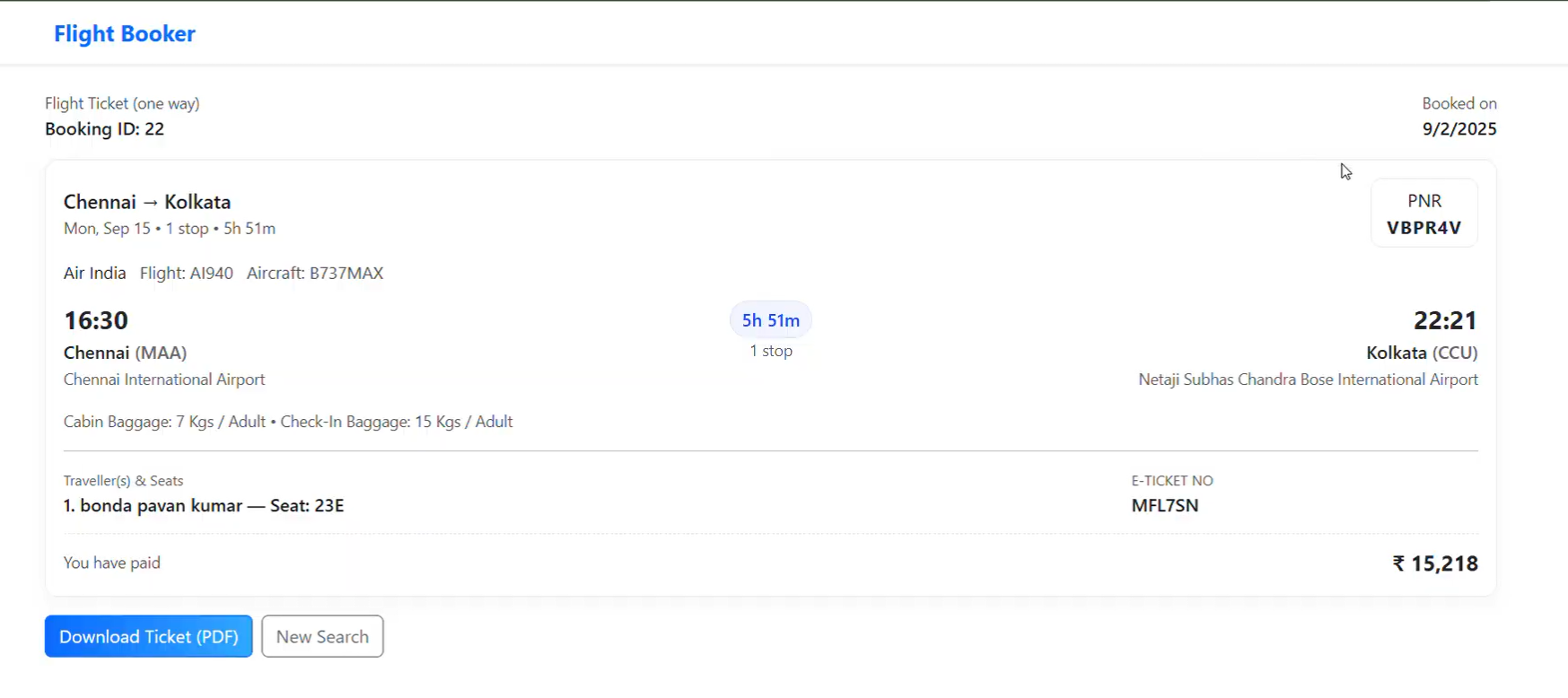
- Data: Booking ID from previous step; submission triggers payment initiation endpoint.

- Actions: Pay calls POST /booking/payments?bookingId; booking status is polled until a terminal state.

- Navigation: Payment → Confirmation.

5) Confirmation

- Screen5:



Pdf :

- Purpose: Present booking summary.

- Content: Booking ID, flight details, passenger list with seat assignment per passenger.

- Actions: Download Ticket generates a PDF of booking details.

- Navigation: Option to return to Home or close the flow.

Demo Video

- URL: [video link](https://drive.google.com/file/d/1CDsOtLX4laP2OL6Z3WnJcl1buj-pAdxU/view?usp=sharing)