NO	REQUEST TYPE	API ENDPOINT	REQUEST HEADERS	REQUEST BODY	RESPONSE CODE	RESPONSE BODY	Comments

1	POST	/api/v1.0/flight/airline/inventory/add	Content-Type: application/json	json {
				"airlineName": "Indigo",
				"flightNumber": "AE101",
				"fromPlace": "DELHI",
				"toPlace": "MUMBAI",
				"departureTime": "2025-11-25T09:00:00",
				"arrivalTime": "2025-11-25T11:00:00",
				"totalSeats": 180,
				"price": 4500,
				"specialFare": 3500,
				"fareCategory": "STUDENT"
				}
					200	json {
						"id": 1,
						"airlineName": "Indigo",
						"flightNumber": "AE101",
						"fromPlace": "DELHI",
						"toPlace": "MUMBAI",
						"departureTime": "2025-11-25T09:00:00",
						"arrivalTime": "2025-11-25T11:00:00",
						"totalSeats": 180,
						"availableSeats": 180,
						"price": 4500,
						"specialFare": 3500,
						"fareCategory": "STUDENT"
						}
							Adds a new flight schedule (inventory) into the system with airline, flight number, route, timings, seats and fares.
							
							Generates a database id automatically using @GeneratedValue when inventory is saved.
							
							Initializes availableSeats to totalSeats when a new inventory record is created.
							
							Persists full flight metadata: airline name, flight number, fromPlace, toPlace, departureTime, arrivalTime.
							
							Stores pricing info: price (base fare) and specialFare (category-specific reduced fare).
							
							Stores fareCategory on inventory (STUDENT/SENIOR/REGULAR/ARMY/CORPORATE) so booking logic can apply discounts.
							
							Admin-only endpoint (POST /api/v1.0/flight/airline/inventory/add) accepts DTO and returns the saved entity.
							
							Uses DTO -> Entity mapping in service layer to prevent exposing JPA entities directly to the API.
							
							Validates input and rejects invalid or malicious inventory data before persisting.

					400	{ "airlineName": "Airline name is required" }	
					500	{ "message": "Something went wrong" }	

			Content-Type: application/json	1) json {
				"fromPlace": "DELHI",
				"toPlace": "MUMBAI",
				"travelDate": "2025-11-25",
				"travelTime": "",
				"tripType":"ROUND_TRIP",
				"returnDate": "2025-11-28"
				}
					200	1) IF return flight is not there
						
						json [
						{ "airlineName": "Indigo",
						"flightNumber": "AE101",
						"departureTime": "2025-11-25T09:00:00",
						"arrivalTime": "2025-11-25T11:00:00",
						"oneWayPrice": 4500,
						"roundTripPrice": 9000,
						"returnFlightNumber": null,
						"returnDepartureTime": null,
						"returnArrivalTime": null,
						"message": "Only onward flight available. Return flight not found."
						}
						]
							Supports searching only the onward journey based on fromPlace, toPlace, and travelDate.
							
							Performs separate searches for onward and return flights and combines them in a single response.
							
							If no return flight is found, onward flight is still returned with a message indicating return unavailability.
							
							If travelTime is empty, system performs a full-day search (00:00–23:59).
							
							If travelTime is provided, a ±30-minute window is used to find nearby flights.
							
							fromPlace and toPlace are matched in lowercase to avoid case sensitivity issues.
							
							Leading or trailing spaces in city names are trimmed

2	POST	/api/v1.0/flight/search		2) json{
				"fromPlace": "DELHI",
				"toPlace": "MUMBAI",
				"travelDate": "2025-11-25",
				"travelTime": "",
				"tripType": "ROUND_TRIP",
				"returnDate": "2025-11-28"
				}
				
				3) json{
				"fromPlace": "DELHI",
				"toPlace": "MUMBAI",
				"travelDate": "2025-11-25",
				"travelTime": "",
				"tripType": "ONE_WAY"
				}
						2) IF return flight is there
						
						[
						{
						"airlineName": "Indigo",
						"flightNumber": "AE101",
						"departureTime": "2025-11-25T09:00:00",
						"arrivalTime": "2025-11-25T11:00:00",
						"oneWayPrice": 4500.0,
						"roundTripPrice": 9000.0,
						"message": null,
						"returnDepartureTime": "2025-11-28T09:00:00",
						"returnArrivalTime": "2025-11-28T11:00:00",
						"returnFlightNumber": "AE102"
						}
						]
						
						3) ONE Way flight
						[
						{
						"airlineName": "Indigo",
						"flightNumber": "AE101",
						"departureTime": "2025-11-25T09:00:00",
						"arrivalTime": "2025-11-25T11:00:00",
						"oneWayPrice": 4500.0,
						"roundTripPrice": 9000.0,
						"message": "One-way trip available.",
						"returnDepartureTime": null,
						"returnArrivalTime": null,
						"returnFlightNumber": null
						}
						]
							before searching.
							
							Converts travelDate into a time range to fetch only flights on the specified day.
							
							Returns roundTripPrice as oneWayPrice × 2 (placeholder for future pricing logic).
							
							Adds messages like “One-way trip available” or “Only onward flight available; return flight not found”.

					400	Validation failed	
					404	No flights found	
					500	Internal Server Error	

3	POST	/api/v1.0/flight/booking/{flightid}	content-type : application/json	{
				"email": "amankumar@gmail",
				"numberOfSeats": 2,
				"passengers": [
				{
				"name": "aman",
				"gender": "MALE",
				"age": 31,
				"meal": "VEG",
				"fareCategory": "STUDENT",
				"seatNumber": 14
				},
				{
				"name": "amen",
				"gender": "MALE",
				"age": 31,
				"meal": "VEG",
				"fareCategory": "ARMY",
				"seatNumber": 15
				}
				]
				}
					200	{
						"pnr": "5RWB4Z",
						"email": "amankumar@gmail",
						"numberOfSeats": 2,
						"totalPrice": 8000.0,
						"bookingTime": "2025-11-17T11:52:31.010319200Z",
						"message": "Booking successful",
						"passengers": [
						{
						"name": "aman",
						"seatNumber": 14,
						"gender": "MALE",
						"age": 31,
						"meal": "VEG",
						"fareCategory": "STUDENT",
						"fareApplied": 3500.0,
						"fareMessage": "Special fare applied for category: STUDENT"
						},
						{
						"name": "amen",
						"seatNumber": 15,
						"gender": "MALE",
						"age": 31,
						"meal": "VEG",
						"fareCategory": "ARMY",
						"fareApplied": 4500.0,
						"fareMessage": "No special fare available. Extra benefits applied."
						}
						]
						}
							Allows a logged-in user to book seats on a selected flight using its flightId.
							
							Generates a unique 6-character alphanumeric PNR for every successful booking.
							
							Automatically links each passenger to the booking and the flight.
							
							Deducts seat count from flight inventory based on number of seats booked.
							
							Stores booking confirmation details including total price, passenger details, and booking timestamp.
							
							Ensures atomic booking using @Transactional to guarantee safe seat updates.
							
							Applies special fare automatically when passenger's fare category matches the flight's configured special category.
							
							Falls back to standard base fare when special fare is not applicable.
							
							Generates personalized messages such as "Special fare applied" or "No special fare available, extra benefits awarded."
							
							Calculates per-passenger fare individually and stores the applied fare in the database.
							
							Computes final total price using the actual per-passenger applied fares (special + normal mix).
							
							Validates that number of seats requested matches the number of passengers provided.
							
							Ensures at least 1 seat and at most available seats are requested.
							
							Checks that no seat number is duplicated in the request (each passenger must be unique).
							
							Prevents booking of already-taken seats using a database query for booked seats.
							
							Ensures the flight exists for the provided flightId, otherwise throws "Flight not found."
							
							Validates all passenger details (name, age, gender, meal preference, seat number).
							
							Requires each passenger to provide a valid fare category (STUDENT/SENIOR/ARMY/etc.).

					500	{
						"status": 500,
						"error": "Internal Server Error",
						"message": "Something went wrong while processing booking.",
						"path": "/api/v1.0/flight/booking/1"
						}	

					400	{
						"status": 400,
						"error": "BAD_REQUEST",
						"message": "Duplicate seat numbers in request",
						"path": "/api/v1.0/flight/booking/1"
						}	

4	GET	/api/v1.0/flight/ticket/{pnr}	content-type : application/json		200	{
						"pnr": "5RWB4Z",
						"email": "amankumar@gmail",
						"numberOfSeats": 2,
						"totalPrice": 8000.0,
						"bookingTime": "2025-11-17T11:52:31.010319Z",
						"message": "Booking retrieved",
						"passengers": [
						{
						"name": "aman",
						"seatNumber": 14,
						"gender": "MALE",
						"age": 31,
						"meal": "VEG",
						"fareCategory": "STUDENT",
						"fareApplied": 3500.0,
						"fareMessage": "Special fare applied for category: STUDENT"
						},
						{
						"name": "amen",
						"seatNumber": 15,
						"gender": "MALE",
						"age": 31,
						"meal": "VEG",
						"fareCategory": "ARMY",
						"fareApplied": 4500.0,
						"fareMessage": "No special fare available. Extra benefits applied."
						}
						]
						}
							Fetches complete ticket details using a unique PNR number.
							
							Returns booking information including flight details, passenger list, and booking metadata.
							
							Provides passenger-wise fare details (special or regular).
							
							Displays current booking status (BOOKED / CANCELLED).
							
							Validates that PNR is a non-empty 6-character alphanumeric string.
							
							Ensures booking exists for the provided PNR; otherwise returns 404 “Ticket not found”.
							
							Ensures cancelled tickets still return history but clearly show "status": "CANCELLED".
							
							Blocks invalid PNR formats (symbols, incorrect length, blank values).
							
							Returns 400 for invalid/malformed PNR.
							
							Returns 404 if booking does not exist.
							
							Returns structured JSON error response through GlobalErrorHandler

					404	{
						"status": 404,
						"error": "NOT_FOUND",
						"message": "Ticket not found for PNR: XYZ123",
						"path": "/api/v1.0/flight/ticket/XYZ123"
						}	

5	GET	/api/v1.0/flight/booking/history/{emailId}	content-type : application/json		200	[
						{
						"pnr": "5RWB4Z",
						"email": "amankumar@gmail",
						"numberOfSeats": 2,
						"totalPrice": 8000.0,
						"bookingTime": "2025-11-17T11:52:31.010319Z",
						"message": "Booking history item",
						"passengers": [
						{
						"name": "aman",
						"seatNumber": 14,
						"gender": "MALE",
						"age": 31,
						"meal": "VEG",
						"fareCategory": "STUDENT",
						"fareApplied": 3500.0,
						"fareMessage": "Special fare applied for category: STUDENT"
						},
						{
						"name": "amen",
						"seatNumber": 15,
						"gender": "MALE",
						"age": 31,
						"meal": "VEG",
						"fareCategory": "ARMY",
						"fareApplied": 4500.0,
						"fareMessage": "No special fare available. Extra benefits applied."
						}
						]
						},
							Fetches all bookings ever made using a specific email ID.
							
							Returns past, current, and cancelled bookings for complete user history.
							
							Shows booking date/time, PNR, total price, and status for each booking.
							
							Supports multiple bookings on different flights and different dates.
							
							Ensures email format is valid according to RFC-compliant regex.
							
							Rejects blank or malformed email values.
							
							Returns an empty list if user has no bookings instead of throwing an error.
							
							Prevents exposing invalid/static resources by using proper path variable escaping.
							
							400 returned for invalid email format.
							
							404 only when email is syntactically correct but not found (optional behavior).
							
							Clean JSON error via GlobalErrorHandler for validation issues.

					400	{
						"status": 400,
						"error": "BAD_REQUEST",
						"message": "Invalid email format",
						"path": "/api/v1.0/flight/booking/history/kunal@gmail"
						}	

					404	{
						"status": 404,
						"error": "NOT_FOUND",
						"message": "No bookings found for email: user@example.com",
						"path": "/api/v1.0/flight/booking/history/user@example.com"
						}	

6	DELETE	/api/v1.0/flight/booking/cancel/{pnr}	content-type : application/json Authorization : Bearer eyJhbGciOiJIUzI		200	Ticket with PNR K3Q22C has been cancelled successfully.	Cancels an existing flight booking using PNR number. Updates booking status to CANCELLED without deleting booking history. Restores previously booked seats back into the flight's available seat count. Ensures passengers remain stored as historical records. Validates PNR format (6 uppercase letters/numbers). Ensures booking exists and is currently in "BOOKED" state (cannot cancel twice). Restricts cancellation if journey date is within 24 hours of departure. Prevents cancellation of already cancelled or expired bookings. 400 returned for invalid PNR format. 404 returned when booking does not exist. 409 Conflict returned when trying to cancel an already cancelled ticket. 400 returned if cancellation attempted within 24 hours of flight departure. Clean JSON error response via GlobalErrorHandler.

					400	{
						"status": 400,
						"error": "BAD_REQUEST",
						"message": "Ticket cannot be cancelled within 24 hours of flight departure.",
						"path": "/api/v1.0/flight/booking/cancel/A34KDX"
						}	

					404	{
						"timestamp": "2025-11-17T14:28:11.869283500Z",
						"status": 404,
						"error": "Not Found",
						"message": "PNR 'K3Q22C' not found",
						"path": "/api/v1.0/flight/booking/cancel/K3Q22C"
						}	

					409	{
						"status": 409,
						"error": "CONFLICT",
						"message": "Ticket with PNR A34KDX is already cancelled.",
						"path": "/api/v1.0/flight/booking/cancel/A34KDX"
						}	

	state	properties	nouns	Is-A	
	behavior	methods	verbs	Author is a human.	GET+/agent
Agent (class)				Author has written 3 books.	POST+/agent
enum	role	"ADMIN" ,"AGENT", "VIEW"			
collection	stores		Has-A (composition)	one to many	
	status			agent -> stores	
int	defaultStoreID				
diableAgent()					
Agent(table)			**store**		
role			id		
			Agent_FK		
id					

		Agent
DefaultStore	role	id
1	ADMIN	1

Store		
id	name	Agent_FK
1	BTM	1
2	Arekere	1

Backend design			Many to many		
AgentController		AgentService		client_service_join_table	
	search()		service_ID	service_FK	client_FK
	update()		1	1	10
	create();	createAgent()		1	10
		book			
			MRP	discount	final price

Fee	client_ID
10	10
12	

Emp			
id		EMP_FK	ROLE
2	ram	1	
1	Ramesh		Manager