Travel Planner Backend API Documentation

> ***Version:*** *1.0.0*
> ***Base URL:*** `http://localhost:3001`
> ***Last Updated:*** *January 2025*

Table of Contents

---

1. Overview

The Travel Planner Backend is a RESTful API built with Express.js and Oracle Database. It provides endpoints for:

- Customer registration and authentication
- Travel package catalog browsing
- Hotel, transport, and food service management
- Booking creation and management (using stored procedures)
- Payment processing (using stored procedures)
- Admin reporting and analytics (using views)

Tech Stack

Component	Technology
Runtime	Node.js
Framework	Express.js
Database	Oracle Database (via `oracledb`)
Authentication	JWT (jsonwebtoken)
Password Hashing	bcrypt

UUID Generation	uuid (v4)

---

2. Architecture

Project Structure

```
backend/
├── app.js                    # Application entry point
├── config/
│   └── db.js                 # Oracle database connection pool
├── controllers/
│   ├── authController.js     # Authentication logic
│   ├── bookingController.js  # Booking operations
│   ├── catalogController.js  # Catalog management
│   ├── customerController.js # Customer operations
│   ├── foodController.js     # Food service operations
│   ├── hotelController.js    # Hotel operations
│   ├── paymentController.js  # Payment processing
│   ├── reportController.js   # Admin reports
│   └── transportController.js# Transport operations
├── middlewares/
│   ├── auth.js               # JWT verification middleware
│   └── helpers.js            # Utility functions (UUID, date formatting)
├── models/
│   ├── adminModel.js         # Admin database operations
│   ├── bookingModel.js       # Booking DB ops (uses stored procedures)
│   ├── catalogModel.js       # Catalog DB ops (uses views)
│   ├── customerModel.js      # Customer DB ops (uses views)
│   ├── foodModel.js          # Food database operations
│   ├── hotelModel.js         # Hotel DB ops (uses views)
│   ├── paymentModel.js       # Payment DB ops (uses stored procedures)
│   ├── reportModel.js        # Report DB ops (uses views)
│   └── transportModel.js     # Transport database operations
├── routes/
│   ├── authRoutes.js
│   ├── bookingRoutes.js
│   ├── catalogRoutes.js
│   ├── customerRoutes.js
│   ├── foodRoutes.js
│   ├── hotelRoutes.js
│   ├── paymentRoutes.js
│   ├── reportRoutes.js
│   └── transportRoutes.js
└── postman/
    └── TravelPlanner.postman_collection.json
```

Request Flow

```
Client Request → Express Router → Auth Middleware → Controller → Model → Database
                                                                    ↓
```

```
Client Response ← Controller ← Model ← (Stored Procedure/View/Table)
```

---

3. Authentication

JWT Token Structure

```
{
  "id": 1,
  "username": "johndoe",
  "role": "customer",  // or "admin"
  "iat": 1234567890,
  "exp": 1234654290
}
```

Authentication Middleware

Middleware	Description
`verifyToken`	Validates JWT token from `Authorization: Bearer <token>` header
`isCustomer`	Ensures user has `role: "customer"`
`isAdmin`	Ensures user has `role: "admin"`

Authorization Rules

Resource	Public	Customer	Admin
View catalogs/hotels/transport/food	✅	✅	✅
Register/Login	✅	-	-
Create bookings	❌	✅ (own)	✅
View bookings	❌	✅ (own)	✅ (all)
Cancel bookings	❌	✅ (own)	✅ (all)
Process payments	❌	✅ (own)	✅
View reports	❌	❌	✅
CRUD hotels/transport/food/catalogs	❌	❌	✅

---

4. API Endpoints

4.1 Auth

Register Customer

```
POST /api/auth/customer/register
```

Request Body:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "+923001234567",
  "username": "johndoe",
  "password": "password123"
}
```

Response (201):

```
{
  "message": "Customer registered successfully",
  "customerId": 1
}
```

Customer Login

```
POST /api/auth/customer/login
```

Request Body:

```
{
  "username": "johndoe",
  "password": "password123"
}
```

Response (200):

```
{
  "message": "Login successful",
  "token": "eyJhbGciOiJIUzI1NiIs...",
  "customer": {
    "customerId": 1,
    "name": "John Doe",
    "email": "john@example.com",
    "username": "johndoe"
  }
}
```

Admin Login

```
POST /api/auth/admin/login
```

Request Body:

```
{
  "username": "admin",
  "password": "admin123"
}
```

Response (200):

```
{
  "message": "Admin login successful",
  "token": "eyJhbGciOiJIUzI1NiIs...",
  "admin": {
    "adminId": 1,
    "name": "Admin User",
    "email": "admin@travelplanner.com",
    "username": "admin"
  }
}
```

4.2 Customers

Get All Customers (Admin Only)

```
GET /api/customers
Authorization: Bearer <admin_token>
```

Response (200):

```
[
  {
    "customerId": 1,
    "name": "John Doe",
    "email": "john@example.com",
    "phone": "+923001234567",
    "username": "johndoe"
  }
]
```

Get Customer Profile

```
GET /api/customers/:id
Authorization: Bearer <token>
```

Response (200):

```
{
  "customerId": 1,
  "name": "John Doe",
  "email": "john@example.com",
```

```
  "phone": "+923001234567",
  "username": "johndoe"
}
```

Get Customer Summary (Uses View)

```
GET /api/customers/:id/summary
Authorization: Bearer <token>
```

Uses: `VW_CUSTOMER_BOOKING_SUMMARY`

Response (200):

```
{
  "customerId": 1,
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "+923001234567",
  "username": "johndoe",
  "totalBookings": 5,
  "confirmedBookings": 3,
  "cancelledBookings": 1,
  "totalSpent": 125000,
  "customerTier": "Gold"
}
```

Customer Tier Thresholds:

- Platinum: ≥ 200,000
- Gold: ≥ 100,000
- Silver: ≥ 50,000
- Bronze: < 50,000

Update Customer Profile

```
PUT /api/customers/:id
Authorization: Bearer <token>
```

Request Body:

```
{
  "name": "John Updated",
  "email": "john.new@example.com",
  "phone": "+923009876543",
  "password": "newpassword123"  // optional
}
```

Delete Customer (Admin Only)

```
DELETE /api/customers/:id
Authorization: Bearer <admin_token>
```

4.3 Catalogs

Get All Catalogs (Uses View)

```
GET /api/catalogs
```

Uses: `VW_CATALOG_FULL_DETAILS`

Response (200):


```
[
  {
    "catalogId": 1,
    "packageName": "Karachi Explorer",
    "destination": "Karachi",
    "description": "3-day tour of Karachi",
    "noOfDays": 3,
    "budget": 50000,
    "departure": "2025-12-01",
    "arrival": "2025-12-03",
    "totalHotels": 1,
    "totalHotelCost": 8000,
    "totalTransports": 1,
    "totalTransportCost": 5000,
    "totalFoodPlans": 2,
    "totalFoodCost": 3000,
    "calculatedTotalCost": 16000
  }
]
```

Get Catalog By ID (Uses View)

```
GET /api/catalogs/:id
```

Response (200):


```
{
  "catalogId": 1,
  "packageName": "Karachi Explorer",
  "destination": "Karachi",
  "description": "3-day tour of Karachi",
  "noOfDays": 3,
  "budget": 50000,
  "departure": "2025-12-01",
  "arrival": "2025-12-03",
  "totalHotels": 1,
  "totalHotelCost": 8000,
```

```
  "totalTransports": 1,
  "totalTransportCost": 5000,
  "totalFoodPlans": 2,
  "totalFoodCost": 3000,
  "calculatedTotalCost": 16000,
  "hotels": [
    {
      "hotelId": 1,
      "hotelName": "Pearl Continental",
      "hotelAddress": "Club Road, Karachi",
      "rent": 8000,
      "roomsIncluded": 1
    }
  ],
  "transport": [
    {
      "transportId": 1,
      "type": "Bus",
      "fare": 2500,
      "seatsIncluded": 2
    }
  ],
  "food": [
    {
      "foodId": 1,
      "meals": "Breakfast + Dinner",
      "price": 1500
    }
  ]
}
```

Create Catalog (Admin Only)

```
POST /api/catalogs
Authorization: Bearer <admin_token>
```

Request Body:


```
{
  "packageName": "Lahore Heritage",
  "destination": "Lahore",
  "description": "Historical tour of Lahore",
  "noOfDays": 4,
  "budget": 60000,
  "departure": "2025-12-10",
  "arrival": "2025-12-14"
}
```

Add Hotel to Catalog (Admin Only)

```
POST /api/catalogs/:id/hotels
Authorization: Bearer <admin_token>
```

Request Body:

```
{
  "hotelId": 2,
  "roomsIncluded": 1
}
```

Add Transport to Catalog (Admin Only)

```
POST /api/catalogs/:id/transport
Authorization: Bearer <admin_token>
```

Request Body:

```
{
  "transportId": 1,
  "seatsIncluded": 2
}
```

Add Food to Catalog (Admin Only)

```
POST /api/catalogs/:id/food
Authorization: Bearer <admin_token>
```

Request Body:

```
{
  "foodId": 1
}
```

4.4 Hotels

Get All Hotels

```
GET /api/hotels
```

Response (200):

```
[
  {
    "hotelId": 1,
    "hotelName": "Pearl Continental",
    "hotelAddress": "Club Road, Karachi",
    "availableRooms": 50,
    "rent": 8000
```

```
  }
]
```

Get Hotel Availability (Uses View)

```
GET /api/hotels/availability
```

Uses: `VW_HOTEL_AVAILABILITY`

Response (200):

```
[
  {
    "hotelId": 1,
    "hotelName": "Pearl Continental",
    "hotelAddress": "Club Road, Karachi",
    "totalRooms": 50,
    "pricePerNight": 8000,
    "currentlyBookedRooms": 5,
    "availableRoomsNow": 45
  }
]
```

Get Hotel By ID

```
GET /api/hotels/:id
```

Create Hotel (Admin Only)

```
POST /api/hotels
Authorization: Bearer <admin_token>
```

Request Body:

```
{
  "hotelName": "Marriott",
  "hotelAddress": "F-7, Islamabad",
  "availableRooms": 100,
  "rent": 15000
}
```

4.5 Transport

Get All Transport

```
GET /api/transport
```

Response (200):

```
[
  {
    "transportId": 1,
    "type": "Bus",
    "availableSeats": 40,
    "fare": 2500
  }
]
```

Get Transport By ID

```
GET /api/transport/:id
```

Create Transport (Admin Only)

```
POST /api/transport
Authorization: Bearer <admin_token>
```

Request Body:

```
{
  "type": "Flight",
  "availableSeats": 150,
  "fare": 25000
}
```

4.6 Food

Get All Food Plans

```
GET /api/food
```

Response (200):

```
[
  {
    "foodId": 1,
    "meals": "Breakfast + Dinner",
    "price": 1500
  }
]
```

Get Food By ID

```
GET /api/food/:id
```

Create Food Plan (Admin Only)

```
POST /api/food
Authorization: Bearer <admin_token>
```

Request Body:

```
{
  "meals": "All-inclusive (3 meals)",
  "price": 3000
}
```

4.7 Bookings

Get All Bookings (Admin Only, Uses View)

```
GET /api/bookings
Authorization: Bearer <admin_token>
```

Uses: `VW_BOOKING_DETAILS`

Response (200):

```
[
  {
    "bookingId": "abc123...",
    "customerId": 1,
    "customerName": "John Doe",
    "customerEmail": "john@example.com",
    "catalogId": 1,
    "packageName": "Karachi Explorer",
    "destination": "Karachi",
    "isCustom": false,
    "bookingType": "Package Booking",
    "bookingDescription": "Karachi trip",
    "bookingDate": "2025-01-15T10:30:00Z",
    "status": "confirmed",
    "totalHotelCost": 16000,
    "totalTransportCost": 5000,
    "totalFoodCost": 3000,
    "grandTotal": 24000,
    "paymentStatus": "completed",
    "paidAmount": 24000
  }
]
```

Get Customer Bookings (Uses View)

```
GET /api/bookings/customer/:customerId
Authorization: Bearer <token>
```

Get Booking By ID (Uses View)

```
GET /api/bookings/:id
Authorization: Bearer <token>
```

Response includes: Full booking details with hotels, transport, food arrays and payment info.

Get Booking Cost Breakdown (Uses Stored Procedure)

```
GET /api/bookings/:id/total
Authorization: Bearer <token>
```

Uses: `SP_CALCULATE_BOOKING_TOTAL`

Response (200):

```
{
  "hotelCost": 16000,
  "transportCost": 5000,
  "foodCost": 3000,
  "totalCost": 24000
}
```

Create Booking from Catalog (Uses Stored Procedure)

```
POST /api/bookings/catalog
Authorization: Bearer <customer_token>
```

Uses: `SP_CREATE_BOOKING_FROM_CATALOG`

Request Body:

```
{
  "customerId": 1,
  "catalogId": 1,
  "bookingDescription": "Karachi trip",
  "checkIn": "2025-12-01",
  "checkOut": "2025-12-03",
  "travelDate": "2025-12-01"
}
```

Response (201):

```
{
  "message": "Booking created successfully",
  "bookingId": "abc123def456...",
```

```
  "totalCost": 24000
}
```

Validations (in stored procedure):

- Customer must exist
- Catalog must exist
- Check-out must be after check-in
- Check-in cannot be in the past

Create Custom Booking

```
POST /api/bookings/custom
Authorization: Bearer <customer_token>
```

Request Body:

```
{
  "customerId": 1,
  "bookingDescription": "Custom Lahore trip",
  "hotels": [
    {
      "hotelId": 2,
      "roomsBooked": 1,
      "checkIn": "2025-12-15",
      "checkOut": "2025-12-17"
    }
  ],
  "transport": [
    {
      "transportId": 1,
      "seatsBooked": 2,
      "travelDate": "2025-12-15"
    }
  ],
  "food": [
    {
      "foodId": 1,
      "quantity": 2
    }
  ]
}
```

Cancel Booking (Uses Stored Procedure)

```
POST /api/bookings/:id/cancel
Authorization: Bearer <token>
```

Uses: `SP_CANCEL_BOOKING`

Request Body:

```
{
  "reason": "Change of plans"
}
```

Response (200):

```
{
  "message": "Booking cancelled successfully",
  "refundIssued": true
}
```

Behavior:

- If payment was completed → marks payment as `refunded`, `refundIssued: true`
- If payment was pending → marks payment as `failed`, `refundIssued: false`
- Updates booking status to `cancelled`

Update Booking Status (Admin Only)

```
PATCH /api/bookings/:id/status
Authorization: Bearer <admin_token>
```

Request Body:

```
{
  "status": "confirmed"
}
```

Valid statuses: `pending`, `confirmed`, `cancelled`

Delete Booking (Admin Only)

```
DELETE /api/bookings/:id
Authorization: Bearer <admin_token>
```

4.8 Payments

Get All Payments (Admin Only)

```
GET /api/payments
Authorization: Bearer <admin_token>
```

Response (200):

```
[
  {
    "paymentId": "pay123...",
```

```
    "bookingId": "book123...",
    "amount": 24000,
    "paymentDate": "2025-01-15T12:00:00Z",
    "method": "credit_card",
    "status": "completed",
    "transactionId": "TXN123456",
    "customerName": "John Doe"
  }
]
```

Get Payment by Booking ID

```
GET /api/payments/booking/:bookingId
Authorization: Bearer <token>
```

Process Payment (Uses Stored Procedure)

```
POST /api/payments/process
Authorization: Bearer <token>
```

Uses: `SP_PROCESS_PAYMENT`

Request Body:

```
{
  "bookingId": "abc123...",
  "method": "credit_card",
  "transactionId": "TXN123456"
}
```

Valid methods: `credit_card` , `debit_card` , `paypal` , `bank_transfer`

Response (201):

```
{
  "message": "Payment processed successfully",
  "paymentId": "pay123...",
  "amount": 24000,
  "status": "pending"
}
```

Validations (in stored procedure):

- Booking must exist
- Booking cannot be cancelled
- Payment must not already exist for booking
- Booking must have items to pay for

Complete Payment (Uses Stored Procedure)

```
POST /api/payments/:id/complete
Authorization: Bearer <token>
```

Uses: `SP_COMPLETE_PAYMENT`

Request Body:

```
{
  "transactionId": "TXN123456_COMPLETE"
}
```

Response (200):

```
{
  "message": "Payment completed successfully. Booking confirmed.",
  "status": "completed"
}
```

Behavior:

- Marks payment as `completed`
- Updates booking status to `confirmed`
- Sets payment date to current timestamp

Update Payment Status (Admin Only)

```
PATCH /api/payments/:id/status
Authorization: Bearer <admin_token>
```

Request Body:

```
{
  "status": "refunded",
  "transactionId": "REFUND123"
}
```

Valid statuses: `pending`, `completed`, `failed`, `refunded`

4.9 Reports (Admin Only)

Get Dashboard Stats

```
GET /api/reports/dashboard
Authorization: Bearer <admin_token>
```

Response (200):

```
{
  "totalCustomers": 50,
```

```
  "totalBookings": 120,
  "confirmedBookings": 85,
  "pendingBookings": 25,
  "cancelledBookings": 10,
  "totalRevenue": 2500000,
  "totalCatalogs": 8
}
```

Get Revenue Report (Uses View)

```
GET /api/reports/revenue
Authorization: Bearer <admin_token>
```

Uses: `VW_REVENUE_REPORT`

Response (200):

```
[
  {
    "paymentMonth": "2025-01",
    "totalTransactions": 45,
    "totalBookings": 40,
    "uniqueCustomers": 35,
    "completedRevenue": 450000,
    "pendingRevenue": 50000,
    "refundedAmount": 25000,
    "packageRevenue": 400000,
    "customBookingRevenue": 50000
  }
]
```

---

5. Data Models

Customer

```
{
  customerId: Number,      // Auto-generated (sequence)
  name: String,
  email: String,           // Unique, lowercase (trigger enforced)
  phone: String,
  username: String,        // Unique
  password: String         // Hashed with bcrypt
}
```

Booking

```
{
  bookingId: String,       // UUID (SYS_GUID)
```

```
  customerId: Number,
  catalogId: Number,        // null for custom bookings
  isCustom: Boolean,        // 0 = catalog, 1 = custom
  bookingDescription: String,
  bookingDate: Date,        // Auto-set to SYSDATE
  status: String            // 'pending', 'confirmed', 'cancelled'
}
```

Payment

```
{
  paymentId: String,        // UUID (SYS_GUID)
  bookingId: String,
  amount: Number,
  paymentDate: Date,
  method: String,           // 'credit_card', 'debit_card', 'paypal', 'bank_transfer'
  status: String,           // 'pending', 'completed', 'failed', 'refunded'
  transactionId: String     // External transaction reference
}
```

---

6. Stored Procedures Integration

The backend uses 5 stored procedures for critical operations:

Procedure	Used By	Purpose
`SP_CREATE_BOOKING_FROM_CATALOG`	`bookingModel.createBookingFromCatalog()`	Atomically creates booking with all items from catalog
`SP_CALCULATE_BOOKING_TOTAL`	`bookingModel.calculateBookingTotal()`	Calculates cost breakdown for a booking
`SP_PROCESS_PAYMENT`	`paymentModel.processPayment()`	Creates pending payment with validation
`SP_COMPLETE_PAYMENT`	`paymentModel.completePayment()`	Completes payment and confirms booking
`SP_CANCEL_BOOKING`	`bookingModel.cancelBooking()`	Cancels booking with refund handling

Example: Calling Stored Procedure

```
const result = await connection.execute(
  `BEGIN
     SP_CREATE_BOOKING_FROM_CATALOG(
       p_customer_id   => :customerId,
       p_catalog_id    => :catalogId,
       p_description   => :description,
       p_check_in      => TO_DATE(:checkIn, 'YYYY-MM-DD'),
```

```
      p_check_out     => TO_DATE(:checkOut, 'YYYY-MM-DD'),
      p_travel_date   => TO_DATE(:travelDate, 'YYYY-MM-DD'),
      p_booking_id    => :bookingId,
      p_total_cost    => :totalCost
    );
  END;`,
  {
    customerId: 1,
    catalogId: 1,
    description: 'My trip',
    checkIn: '2025-12-01',
    checkOut: '2025-12-03',
    travelDate: '2025-12-01',
    bookingId: { dir: oracledb.BIND_OUT, type: oracledb.STRING, maxSize: 36 },
    totalCost: { dir: oracledb.BIND_OUT, type: oracledb.NUMBER }
  }
);
```

---

7. Views Integration

The backend uses 5 views for optimized data retrieval:

View	Used By	Purpose
`VW_CATALOG_FULL_DETAILS`	`catalogModel.getAllCatalogs()`, `getCatalogById()`	Catalog with calculated costs
`VW_BOOKING_DETAILS`	`bookingModel.getAllBookings()`, `getBookingsByCustomerId()`, `getBookingById()`	Comprehensive booking info
`VW_CUSTOMER_BOOKING_SUMMARY`	`customerModel.getCustomerSummary()`	Customer stats and tier
`VW_REVENUE_REPORT`	`reportModel.getRevenueReport()`	Monthly revenue analytics
`VW_HOTEL_AVAILABILITY`	`hotelModel.getHotelAvailability()`	Real-time hotel availability

---

8. Error Handling

HTTP Status Codes

Code	Meaning
200	Success
201	Created
400	Bad Request (validation error)
401	Unauthorized (invalid token)

403	Forbidden (access denied)
404	Not Found
500	Internal Server Error

Oracle Error Code Mapping

The controllers map stored procedure errors to HTTP responses:

Oracle Error	HTTP Status	Message
ORA-20001	404	Customer not found
ORA-20002	404	Catalog package not found
ORA-20003	400	Check-out date must be after check-in date
ORA-20004	400	Check-in date cannot be in the past
ORA-20005	404	Booking not found
ORA-20006	400	Cannot process payment for cancelled booking
ORA-20007	400	Payment already exists for this booking
ORA-20008	400	Invalid payment method
ORA-20009	400	Booking has no items to pay for
ORA-20010	404	Payment not found
ORA-20011	400	Payment is not pending
ORA-20012	400	Booking is already cancelled

Error Response Format

```
{
  "message": "Human-readable error message",
  "error": "Technical error details (development mode only)"
}
```

---

9. Environment Variables

Create a `.env` file in the backend directory:

```
# Server
PORT=3001
NODE_ENV=development

# Database
DB_USER=travel_planner
```

```
DB_PASSWORD=travel123
DB_CONNECTION_STRING=localhost:1521/FREEPDB1

# JWT
JWT_SECRET=your-super-secret-jwt-key
JWT_EXPIRES_IN=24h
```

Appendix: API Endpoint Summary

Method	Endpoint	Auth	Description
POST	`/api/auth/customer/register`	-	Register customer
POST	`/api/auth/customer/login`	-	Customer login
POST	`/api/auth/admin/login`	-	Admin login
GET	`/api/customers`	Admin	List all customers
GET	`/api/customers/:id`	Token	Get customer profile
GET	`/api/customers/:id/summary`	Token	Get customer stats (view)
PUT	`/api/customers/:id`	Token	Update customer
DELETE	`/api/customers/:id`	Admin	Delete customer
GET	`/api/catalogs`	-	List catalogs (view)
GET	`/api/catalogs/:id`	-	Get catalog details (view)
POST	`/api/catalogs`	Admin	Create catalog
PUT	`/api/catalogs/:id`	Admin	Update catalog
DELETE	`/api/catalogs/:id`	Admin	Delete catalog
POST	`/api/catalogs/:id/hotels`	Admin	Add hotel to catalog
POST	`/api/catalogs/:id/transport`	Admin	Add transport to catalog
POST	`/api/catalogs/:id/food`	Admin	Add food to catalog
GET	`/api/hotels`	-	List hotels
GET	`/api/hotels/availability`	-	Hotel availability (view)
GET	`/api/hotels/:id`	-	Get hotel
POST	`/api/hotels`	Admin	Create hotel
PUT	`/api/hotels/:id`	Admin	Update hotel
DELETE	`/api/hotels/:id`	Admin	Delete hotel
GET	`/api/transport`	-	List transport

GET	`/api/transport/:id`	-	Get transport
POST	`/api/transport`	Admin	Create transport
PUT	`/api/transport/:id`	Admin	Update transport
DELETE	`/api/transport/:id`	Admin	Delete transport
GET	`/api/food`	-	List food plans
GET	`/api/food/:id`	-	Get food plan
POST	`/api/food`	Admin	Create food plan
PUT	`/api/food/:id`	Admin	Update food plan
DELETE	`/api/food/:id`	Admin	Delete food plan
GET	`/api/bookings`	Admin	List all bookings (view)
GET	`/api/bookings/customer/:id`	Token	Customer's bookings (view)
GET	`/api/bookings/:id`	Token	Get booking details (view)
GET	`/api/bookings/:id/total`	Token	Booking cost breakdown (SP)
POST	`/api/bookings/catalog`	Customer	Create from catalog (SP)
POST	`/api/bookings/custom`	Customer	Create custom booking
POST	`/api/bookings/:id/cancel`	Token	Cancel booking (SP)
PATCH	`/api/bookings/:id/status`	Admin	Update status
DELETE	`/api/bookings/:id`	Admin	Delete booking
GET	`/api/payments`	Admin	List all payments
GET	`/api/payments/booking/:id`	Token	Get payment for booking
POST	`/api/payments/process`	Token	Process payment (SP)
POST	`/api/payments/:id/complete`	Token	Complete payment (SP)
PATCH	`/api/payments/:id/status`	Admin	Update payment status
GET	`/api/reports/dashboard`	Admin	Dashboard stats
GET	`/api/reports/revenue`	Admin	Revenue report (view)

Legend:

- **SP** = Uses Stored Procedure
- **View** = Uses Database View
- **Token** = Requires any valid JWT
- **Customer** = Requires customer role
- **Admin** = Requires admin role