Air Cargo Booking & Tracking System
Project Documentation

Prepared by: Aryan

Contents

INTRODUCTION

Project Overview

The Air Cargo Booking & Tracking System is a comprehensive Django-based web application designed to manage the entire lifecycle of air cargo shipments. It provides a seamless interface for creating bookings, finding optimal flight routes, tracking cargo in real-time, and managing the complete shipment workflow.

Key Features

- Booking Management: Create, view, and manage cargo bookings with customer details.
- Route Planning: Intelligent flight route calculation with direct and transit options.
- Real-time Tracking: Comprehensive tracking with detailed timeline and status updates.
- Flight Integration: Seamless integration with flight schedules and capacity management.
- Status Management: Complete lifecycle tracking (Booked → Departed → Arrived → Delivered).
- Responsive UI: Modern, mobile-friendly interface built with Bootstrap 5.

Technology Stack

Backend

- Django 5.2.5: High-level Python web framework.
- Django REST Framework: Powerful API framework.
- SQLite: Lightweight database (easily configurable for PostgreSQL/MySQL).
- Python 3.8+: Programming language.

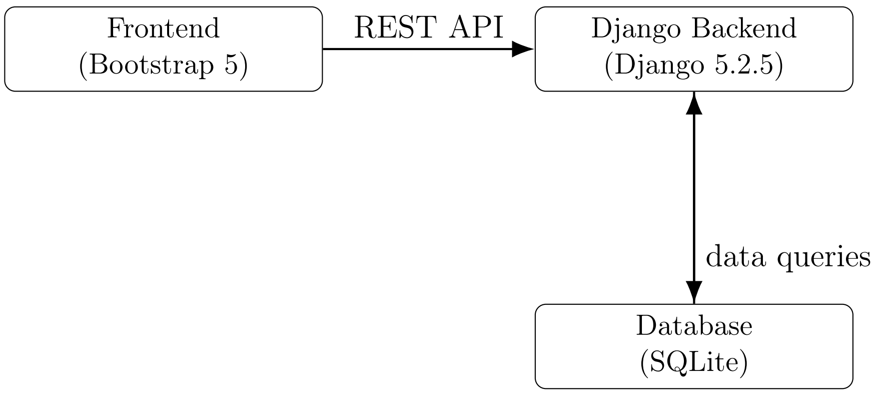
Frontend

- Bootstrap 5: Modern CSS framework.
- HTML5/CSS3: Semantic markup and styling.
- JavaScript (ES6+): Client-side functionality.
- Axios: Promise-based HTTP client.

HIGH-LEVEL DESIGN

System Architecture

The system follows a client-server architecture with a Django backend and a modern Bootstrap frontend. The backend exposes RESTful APIs that are consumed by the frontend.



Core Components

1. **Flight Management**: Handles flight information, schedules, and cargo capacity.
2. **Booking Management**: Manages cargo bookings, customer information, and tracking.
3. **Route Planning**: Calculates direct and transit routes between airports.
4. **Tracking System**: Provides real-time tracking of cargo shipments.
5. **Admin Interface**: Django admin panel for system management.

Data Flow

1. Users access the web interface to create bookings or track shipments.
2. Frontend makes API calls to the Django backend.
3. Backend processes requests and interacts with the database.
4. Responses are returned to the frontend for display.

LOW-LEVEL DESIGN

Flight Management Module

Models

The `Flight` model represents airline flights with the following attributes:

- Flight number (unique)
- Airline name

- Departure and arrival datetime
- Origin and destination airport codes
- Aircraft type
- Max cargo weight and available cargo weight

Key Functions

- `reserve_cargo_weight()`: Reserves cargo weight for a booking.
- `release_cargo_weight()`: Releases reserved cargo weight.
- `is_available_for_booking()`: Checks if flight has available capacity.
- `update_availability()`: Updates available cargo after reservation/release.

Booking Management Module

Models

The `Booking` model represents cargo bookings with:

- Reference ID (human-friendly unique identifier)
- Origin and destination airport codes
- Number of pieces and weight
- Status (Booked, Departed, Arrived, Delivered, Cancelled)
- Customer information (name, email, phone)
- Flight relationships (many-to-many)
- Additional details (description, special instructions)
- Timestamps and current location tracking

The `BookingEvent` model tracks the history of booking events:

- Booking reference
- Event type (Booked, Departed, Arrived, etc.)
- Location
- Flight reference
- Description and timestamp

Key Functions

- `generate_ref_id()`: Creates a unique human-friendly reference ID.
- `depart()`: Marks booking as departed and logs event.
- `arrive()`: Marks booking as arrived and logs event.
- `deliver()`: Marks booking as delivered and logs event.
- `cancel()`: Cancels booking if allowed and releases reserved cargo.
- `can_be_cancelled()`: Checks if booking can be cancelled according to business rules.

Route Planning Module

API Endpoint

The route search API finds direct flights and 1-transit routes:

- Input: origin, destination, departure date.
- Finds direct flights with available cargo capacity.
- Calculates 1-transit routes with reasonable connection times.
- Returns both direct flights and transit routes with estimated total time and available capacity.

Key Algorithm Notes

- Avoids cycles in transit routing.
- Uses connection-time thresholds to filter infeasible transfers.
- Prioritizes fewer transits and shorter total travel time.

Tracking System Module

Event Tracking

All booking events are logged in the `BookingEvent` model:

- Automatic event creation during status transitions (departed, arrived, delivered).
- Manual event creation for custom events (delays, inspections).
- Chronological timeline display in UI and audit logs for each booking.

DATABASE SCHEMA

Database Schema Details

Flights Table

- `id` (INTEGER PRIMARY KEY): Unique identifier for the flight.
- `flight_number` (VARCHAR(20) UNIQUE): Unique flight number.
- `airline_name` (VARCHAR(100)): Name of the airline operating the flight.
- `departure_datetime` (DATETIME): Scheduled departure date and time.
- `arrival_datetime` (DATETIME): Scheduled arrival date and time.
- `origin` (VARCHAR(10)): IATA code of the departure airport.
- `destination` (VARCHAR(10)): IATA code of the arrival airport.
- `aircraft_type` (VARCHAR(50)): Type of aircraft used for the flight.
- `max_cargo_weight` (INTEGER): Maximum cargo weight capacity in kilograms.
- `available_cargo_weight` (INTEGER): Currently available cargo weight capacity in kilograms.
- `created_at` (DATETIME): Timestamp when the flight record was created.
- `updated_at` (DATETIME): Timestamp when the flight record was last updated.

Bookings Table

- `id` (INTEGER PRIMARY KEY): Unique identifier for the booking.
- `ref_id` (VARCHAR(20) UNIQUE): Human-friendly unique reference ID for the booking.
- `origin` (VARCHAR(10)): IATA code of the origin airport.
- `destination` (VARCHAR(10)): IATA code of the destination airport.
- `pieces` (INTEGER): Number of pieces in the shipment.
- `weight_kg` (INTEGER): Total weight of the shipment in kilograms.
- `status` (VARCHAR(20)): Current status of the booking (e.g., Booked, Departed, Arrived, Delivered, Cancelled).
- `customer_name` (VARCHAR(100)): Name of the customer associated with the booking.
- `customer_email` (VARCHAR(254)): Email address of the customer.
- `customer_phone` (VARCHAR(20)): Phone number of the customer.

- `description` (TEXT): Description of the cargo being shipped.
- `special_instructions` (TEXT): Any special instructions for handling the shipment.
- `created_at` (DATETIME): Timestamp when the booking was created.
- `updated_at` (DATETIME): Timestamp when the booking was last updated.
- `current_location` (VARCHAR(10)): IATA code of the airport where the shipment is currently located.

Booking Events Table

- `id` (INTEGER PRIMARY KEY): Unique identifier for the event.
- `booking_id` (INTEGER): Foreign key referencing the associated booking in the `bookings` table.
- `event_type` (VARCHAR(20)): Type of event (e.g., Booked, Departed, Arrived, Delivered, Cancelled, Delayed, Inspection).
- `location` (VARCHAR(10)): IATA code of the airport where the event occurred.
- `flight_id` (INTEGER): Foreign key referencing the associated flight in the `flights` table (if applicable).
- `description` (TEXT): Description of the event.
- `timestamp` (DATETIME): Date and time when the event occurred.
- `created_by` (VARCHAR(100)): User or system that created the event record.
- `metadata` (JSON): Additional structured data related to the event.

Junction Table (Bookings-Flights)

- `id` (INTEGER PRIMARY KEY): Unique identifier for the relationship.
- `booking_id` (INTEGER): Foreign key referencing a booking in the `bookings` table.
- `flight_id` (INTEGER): Foreign key referencing a flight in the `flights` table.

API ENDPOINTS

Flight Endpoints

- `GET /api/flights/`: List all flights.
- `POST /api/flights/`: Create a new flight.
- `GET /api/flights/{id}/`: Get flight details.
- `PUT /api/flights/{id}/`: Update flight details.

- `DELETE /api/flights/{id}/`: Delete a flight.
- `POST /api/flights/routes/`: Find routes between airports.
- `GET /api/flights/search/`: Search flights with filters.

Booking Endpoints

- `GET /api/bookings/`: List all bookings.
- `POST /api/bookings/`: Create a new booking.
- `GET /api/bookings/{ref_id}/`: Get booking details.
- `PUT /api/bookings/{ref_id}/`: Update booking details.
- `DELETE /api/bookings/{ref_id}/`: Delete a booking.
- `GET /api/bookings/history/{ref_id}/`: Get booking history.
- `GET /api/bookings/search/{ref_id}/`: Search booking by reference ID.
- `GET /api/bookings/events/{ref_id}/`: Get booking events.
- `POST /api/bookings/depart/{ref_id}/`: Mark booking as departed.
- `POST /api/bookings/arrive/{ref_id}/`: Mark booking as arrived.
- `POST /api/bookings/deliver/{ref_id}/`: Mark booking as delivered.
- `POST /api/bookings/cancel/{ref_id}/`: Cancel a booking.

USER INTERFACE

Main Pages

- Home Dashboard: Overview of recent bookings and system status.
- Create Booking: Form for creating new cargo bookings.
- Search Booking: Interface for tracking existing bookings.
- Booking Details: Detailed view of booking information and timeline.

UI Components

- Responsive Bootstrap 5 design.
- Interactive forms with validation.
- Real-time status updates.
- Timeline visualization for tracking history.
- Flight information display.
- Customer information panel.

Admin Interface

- Flight management.
- Booking management.
- User management (Django admin).
- System reports and analytics.

SECURITY CONSIDERATIONS

Implemented Security Features

- CSRF protection for forms.
- SQL injection prevention through Django ORM.
- XSS protection in templates.
- Input validation on all forms and APIs.
- Secure admin interface with authentication.

Authentication and Authorization

- Django admin authentication for system management.
- API endpoints can be secured with authentication middleware (e.g., Token/JWT/OAuth2).
- Booking actions are protected through API endpoints that validate status transitions.

Data Protection

- Sensitive information (customer details) stored securely in database.
- Passwords handled by Django's secure authentication system.
- Logging of all booking events for audit trail.

PERFORMANCE OPTIMIZATIONS

Database Optimizations

- Proper indexing on frequently queried fields (`ref_id`, `status`, `origin`, `destination`, timestamps).
- Efficient query design using Django ORM.
- Transaction management for data consistency.

API Optimizations

- Pagination for list endpoints.
- Caching strategies implemented with distributed locks for concurrent operations.
- Efficient serialization of data.

Frontend Optimizations

- Minimized HTTP requests.
- Responsive design with efficient CSS.
- Asynchronous JavaScript for non-blocking operations.

TESTING STRATEGY

Unit Testing

- Model validation tests.
- API endpoint tests.
- Business logic tests for status transitions.

Integration Testing

- End-to-end workflow testing (booking creation to delivery).
- API integration with frontend.
- Database interaction tests.

Manual Testing

- UI functionality verification.
- User experience validation.
- Cross-browser compatibility testing.

Test Data

The system includes sample data generation scripts to create test flights and bookings with various statuses.

DEPLOYMENT GUIDELINES

Development Setup

1. Clone the repository.
2. Set up virtual environment.
3. Install dependencies from `requirements.txt`.
4. Configure database settings.
5. Run migrations.
6. Load sample data.
7. Start development server.

Production Deployment

1. Use PostgreSQL/MySQL instead of SQLite.
2. Configure proper CORS settings.
3. Set up SSL certificates.
4. Use a production WSGI server (Gunicorn/uWSGI).
5. Configure static file serving (e.g., using WhiteNoise or a CDN).
6. Set up monitoring and logging.
7. Implement proper backup strategies.

Environment Configuration

- Set `DEBUG=False` for production.
- Use environment variables for sensitive settings.
- Configure `ALLOWED_HOSTS` appropriately.
- Set up proper logging configuration.

Scaling Considerations

- Database optimization for large datasets.
- Caching strategies for frequently accessed data.
- Load balancing for high-traffic scenarios.
- Asynchronous task processing for heavy operations (e.g., Celery).