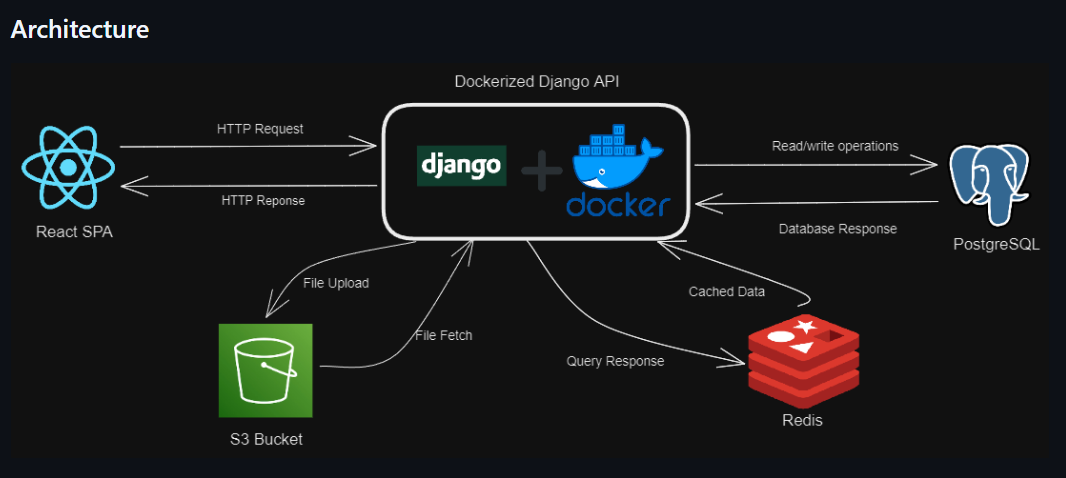
**JETSETGO**

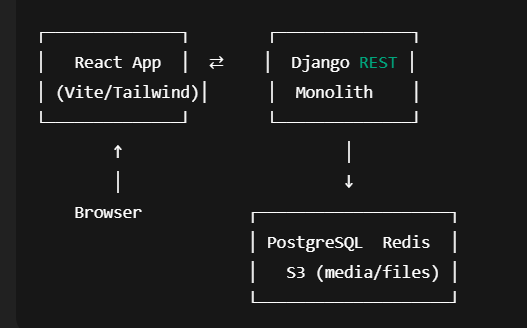
*Backend*

*Django nd python architecture*



## **1. Tech Stack Overview**

Architecture (Monolith + Client)



**Backend (server/)**

* **Framework:** Django 4.2 + Django REST Framework
* **Auth & Security:** Django’s auth + DRF token auth, CORS middleware, rate-limiting, JWT for some flows
* **Database:** PostgreSQL (via django.db.backends.postgresql)
* **Caching & Queues:** Redis (via django-redis)
* **Storage:** S3-compatible object storage (via django-storages + boto3)
* **Static & Media:** Whitenoise for static files, media served from S3
* **Other libs:** django-cors-headers, django-ratelimit, django-storages, whitenoise, troposphere (CloudFormation helper), certifi, requests, xml.etree for XML parsing

**Frontend (client/)**

* **Framework:** React + TypeScript + Vite
* **Styling:** Tailwind CSS
* **Build & Deploy:** Dockerfile for client, ESLint config, deployed via Vercel

**Infrastructure & CI**

* **Containerization:** Docker & Docker Compose (Redis, Postgres)
* **Server:** Gunicorn config for WSGI, ASGI support via asgi.py
* **CI:** GitHub Actions workflow (.github/workflows/integrate.yml)
* **Hosting:** Vercel for frontend, likely AWS (CloudFormation) for backend

## **2. Core Architecture**

1. **Monolithic Django app** under server/api/ exposing REST endpoints for:  
   * User registration, login/logout, token auth
   * Flight search (FlightsView), city autocomplete, location lookups
   * Posts, comments, likes, popular destinations
   * Restaurant suggestions (external API calls + XML parsing)
2. **URL routing**
   * Root: api/ → api.urls
   * Key paths:  
     + api/auth/login, api/auth/logout
     + api/flights/<departure>/<arrival>/<date>
     + api/comments, api/comments/<post>, api/likes, etc.
3. **Data models** (api/models.py)  
   * UserPost, Comments, PostLikes for social features
   * PopularDestination and Item lookup tables
   * ForeignKey relationships to Django User
4. **Middleware & Helpers**
   * Custom middleware in api/middleware.py (e.g., timing, logging)
   * Storage backends in jet\_set\_go/storage\_backends.py for S3

| **Module** | **Django App Code** | **Core Models & Fields** |
| --- | --- | --- |
| **Authentication** | api/views.py & DRF tokens | Uses Django User + DRF’s TokenAuthentication + session cookies |
| **Items Catalog** | ItemCreateView/ItemDetails | Item (UUID item\_id, itemName) |
| **Social (Posts)** | UserPostView/UserPostDetailsView | UserPost (UUID post\_id, content, image URL) |
| **Comments & Likes** | PostCommentsView, LikesView, etc. | Comments (UUID, FK→User, FK→UserPost, content) PostLikes (UUID, unique constraint on User+Post) |
| **Popular Destinations** | PopularDestinationView | PopularDestination (UUID, name, meta-info) |
| **Flight Search** | FlightsView, FlightResponse | No local model—on GET it: calls external XML API → parses → returns JSON |
| **City Autocomplete** | CityAutocomplete | No model—pushes GET /cities/autocomplete/{city} to RapidAPI JSON |
| **Location & Dining** | GetLocationId, GetRestaurants | No models—lookup location IDs & restaurant data via external APIs |

## **4. API Endpoints & Request Flows**

### **Authentication**

* **POST** /api/auth/signup → create user (username, email, password)
* **POST** /api/auth/login → authenticate() + login() → issues session cookie
* **GET** /api/auth/verify → checks session, returns logged-in username
* **POST** /api/auth/logout → logout() → clears session

### **Flight & Location Lookups**

* **GET** /api/flights/{departure}/{arrival}/{date}  
  + Django view fetches from RapidAPI XML endpoint → xml.etree parses → JSON response
* **GET** /api/cities/autocomplete/{city}
* **GET** /api/location/{keyword} → returns a TripAdvisor location ID
* **GET** /api/restaurants/{locationId} → caches results for 15 min via @cache\_page(900)

### **Core CRUD Endpoints**

* **Items**
  + GET/POST /api/items
  + GET/PUT/DELETE /api/items/{item\_id}
* **Popular Destinations**
  + GET/POST /api/popular\_destinations
  + GET/PUT/DELETE /api/popular\_destinations/{id}
* **User Posts**
  + GET/POST /api/posts
  + GET/PUT/DELETE /api/posts/{post\_id}
* **Comments & Likes**
  + GET/POST /api/comments
  + DELETE /api/comments/{comment\_id}
  + GET/POST /api/likes
  + DELETE /api/likes/{like\_id}

More detailing on core parts

### **1. Flight Search**

**What it does:**

* Lets a user find flights between two cities on a chosen date.

**How it works (step-by-step):**

1. **User Request**: You enter departure city, arrival city, and travel date on the website.
2. **Server Call**: The website sends this information to JetSetGo’s central computer (the "server").
3. **External Lookup**: The server asks a flight information service (a third-party) for available flights.
4. **Data Conversion**: That service sends back data in a special format (XML), which the server converts into simple lists.
5. **Show Results**: The server sends back the first few flight options, and the website displays them for you.

**Key Points:**

* JetSetGo does not store flight schedules itself; it fetches them live from a trusted flight data provider.
* It only shows the first few matches to keep information quick and easy to read.

### **2. User Login & Security**

This is how people create accounts, log in, and stay logged in when using JetSetGo.

#### **A. Signing Up (Creating an Account)**

1. **Fill Form**: You provide a username, email, and password on the signup page.
2. **Save Details**: The server saves your details securely and hashes (scrambles) your password so no one can read it.
3. **Confirmation**: You get a message saying "User registration successful."

#### **B. Logging In**

1. **Enter Credentials**: You enter username and password on the login page.
2. **Check Details**: The server checks if the details match an existing account.
3. **Session Cookie**: If correct, the server gives your browser a "session cookie" (a small piece of data) to remember you.
4. **Success Message**: You see "Login successful." on the screen.

#### **C. Staying Logged In (Session Verification)**

* When you navigate to another page, your browser sends the session cookie back to the server.
* The server checks the cookie:
  + **Valid**: You remain logged in.
  + **Invalid or Missing**: You are asked to log in again.

#### **D. Logging Out**

* When you click "Logout", the server removes your session cookie and sends you back to the home page. You are no longer logged in.

### **3. How the Website Connects to These Features**

* The **front-end** (what you see and click) is built with React (a tool for building websites).
* It communicates with the **back-end** (the server) over the internet:
  + **Flight Search**: Sends flight queries → Receives flight options
  + **Login/Verify**: Sends login info → Receives session cookie → Uses cookie for protected pages

| **Why this matters for ASR Aviation:**   * **Flight Search**: We can compare how fast and how many options JetSetGo shows vs our system. * **User Login**: We can see if session cookies are secure enough or if we need stronger methods (like tokens). |
| --- |