FastAPI PokéCenter Project – Explanation & Features

This PokéCenter project is a full-featured backend API built using FastAPI. It simulates a real-world ecosystem for managing Pokémon data, trainers, teams, and media. All routes are covered by comprehensive unit tests using pytest and FastAPI's TestClient. Security has been tightly integrated using JWT authentication (access + refresh tokens), role-based access control (RBAC), and secure file validation.

The project has been successfully deployed to multiple platforms:
- □ Render (cloud app hosting)
- □ AWS EC2 (Ubuntu) with NGINX reverse proxy
- □ HTTPS enabled via Let's Encrypt SSL with auto-renewal cron jobs

This ensures that the website is not only functional and modular but also production-ready, secure, and scalable. The architecture includes modular routers, reusable utility functions, custom error handling, logging, and a Swagger UI for interactive API testing.

Key Features Included in the Project:

- □ Modular architecture using routers (Pokémon, Trainer, Uploads, Auth, etc.)
- □ JWT Authentication with access and refresh tokens
- □ Role-Based Access Control (RBAC) – 'admin' and 'trainer' roles
- □ Token rotation and expiry for refresh tokens
- □ Full CRUD operations for Pokémon and trainer teams
- □ Upload endpoints for CSV files, images, and text with validation
- □ Download endpoints for Pokédex CSVs, battle logs, and images

- ⍰ Preview endpoints for text and image uploads
- ⍰ Custom rate limiting using slowapi
- ⍰ Secure MIME and file-type validation to prevent malicious uploads
- ⍰ Unit tests for every router using pytest and FastAPI TestClient
- ⍰ Deployment to Render and AWS EC2 (Ubuntu) using NGINX reverse proxy
- ⍰ HTTPS security enabled via Let's Encrypt SSL (auto-renewed)
- ⍰ Swagger and ReDoc docs with custom descriptions and tags
- ⍰ Detailed logging via `custom_logger.py` and middleware integration
- ⍰ Static file hosting (images, downloads) and environment variable management
- ⍰ Project ready for future extension into Dart UI frontend, Power BI dashboards, and ML integrations

All configurations, secrets, and environment variables are managed securely using .env and dependency injection.

To build this, I used ChatGPT (Nova) as a learning partner. Every decision, config, and line of code was implemented, debugged, and tested by me. Nova helped me understand each FastAPI concept deeply, but this project is my own creation.

Why Pokémon?

I chose a Pokémon theme because it’s been a huge part of my life since childhood. I grew up watching the anime, playing the games, and collecting cards — so it’s more than just nostalgia. Pokémon has always symbolized curiosity, growth, and persistence to me.

But beyond the emotional connection, Pokémon is also structurally perfect for a full-fledged API project. It has clearly defined entities like Pokémon species, types, evolutions, trainers, teams, and regions — all of which map beautifully to API resources, endpoints, data models, and database relationships.

That made it the ideal candidate to explore complex backend systems while staying passionate and creatively engaged. And if I ever decide to shift from a fun dataset like Pokémon to a real-world business use case (like e-commerce or health records), the architecture will remain the same — only the data would change. That’s the power of good modular design.

Interview Q&A Short & Confident Answers:

Q: What is this project about?

A: Its a full backend API built with FastAPI, handling Pokemon, trainers, and teams with modular routing and secure architecture.

Q: Why FastAPI?
A: Its modern, async-ready, type-safe, and perfect for high-performance APIs great for learning and real deployment.

Q: What features does your API include?

A: JWT auth, RBAC, image uploads, background tasks, custom error logging, validation, and live deployment.

Q: How did you deploy it?
A: First to Render, then AWS EC2 with NGINX and HTTPS for production
readiness.

Q: Did you build it alone?
A: Yes. Every component was built, debugged, and integrated by me using Nova (AI) as a guide,not a coder.

Q: Can this scale?
A: Absolutely. Its modular, follows best practices, and can be containerized or extended further.

Q: Why not a traditional app theme?
A: Pokemon kept it fun, but I made sure the logic maps cleanly to real-world backend systems.

Q: How did you handle security?
A: JWT tokens, rotating refresh tokens, role-based access, custom 401/403 error handling, and secure routing.

Q: What challenges did you face?
A: Handling token rotation and refresh logic, NGINX proxy config, and modularizing over 10 routers.

Q: Is this production-ready?
A: Yes. Its been deployed to cloud (Render + AWS EC2), with HTTPS, .env secrets, CORS, and Docker setup
Q: How would you extend this?
A: Add a PostgreSQL database, CI/CD pipeline, frontend dashboard, and ML recommendation model.

Q: What makes your API different from other FastAPI projects?
A: Most FastAPI projects stop at CRUD. Mine goes further — RBAC, refresh token rotation, image & file uploads, background tasks, logging, and production deployment. Plus, it’s built for evolution — I’ll soon integrate a Dart UI, machine learning models and interactive dashboards.