## ****Project: Production-Grade Task Management API****

We will build a **cloud-ready FastAPI backend** with:

* **JWT-based authentication** with OAuth2 password flow
* **Role-based access control (RBAC)** — Admin / Regular User
* **Task CRUD API** with PostgreSQL persistence
* **External API integration** (task metadata enrichment)
* **Background job processing** (email notifications, async heavy tasks)
* **Caching & rate limiting** with Redis
* **Robust error handling & validations**
* **Deployment** to AWS ECS (Dockerized) with CI/CD
* **Scalability, fault tolerance, and observability features**

This API will be production-grade and prepared for **high concurrency, large user base, and cloud scaling**. **Key Features:**

1. **Validation** — every endpoint has request/response Pydantic validation.
2. **Authentication & Authorization** — JWT + RBAC middleware.
3. **Rate limiting** — Redis-backed limit to prevent abuse.
4. **Caching** — frequently accessed tasks cached in Redis.
5. **Resilience** — retries for failed DB or API calls.
6. **Background jobs** — Celery worker for sending emails, async enrichment.
7. **Error handling** — custom exception handlers for consistent API errors.
8. **Scalability** — ECS Fargate, load balancer, auto-scaling.
9. **Database Migrations** — Alembic for schema version control.
10. **Logging & Monitoring** — structured logging, OpenTelemetry-ready.

# Quick summary (what we'll deliver in the project)

* Full FastAPI app with async SQLAlchemy + PostgreSQL + Alembic (data modeling included).
* JWT-based auth + **RBAC** (roles & permissions), used in route protection.
* Input validation endpoints + advanced Pydantic usage.
* Redis caching + Redis-based **rate limiting** and job queue patterns.
* Background tasks & async external API integration (httpx).
* Observability: structured logs, health checks, metrics (Prometheus).
* Resilience: retries, circuit breaker pattern guidance, idempotency.
* Scalability & concurrency: design for autoscaling on ECS / Kubernetes or Lambda (with concurrency limits).
* Load balancing & deployment options: AWS ECS/Fargate with ALB or EKS / Kubernetes, or Lambda + API Gateway.
* CI/CD + tests + docs for interviews.

## ****Day-by-Day Execution Plan (7 days, 5-6 hrs/day)****

**Day 1:** Project setup, Docker + docker-compose with Postgres & Redis, FastAPI base app, config management.  
**Day 2:** DB models for User & Task, Alembic migrations, DB session handling.  
**Day 3:** Auth endpoints (register/login), password hashing, JWT tokens, role-based access.  
**Day 4:** CRUD for tasks, Pydantic schemas, validations, pagination, caching with Redis.  
**Day 5:** External API integration for task metadata, Celery background jobs, SendGrid emails.  
**Day 6:** Rate limiting, error handling, retries, idempotency, logging setup.  
**Day 7:** CI/CD with GitHub Actions, ECS Fargate deployment, load testing, final polishing.

If we build **this exact project**, you’ll:

* Learn async FastAPI end-to-end
* Understand how to structure large production apps
* Be able to talk confidently about scalability, resilience, and best practices in interviews

## ****Tech Stack****

| **Layer** | **Technology** |
| --- | --- |
| **Framework** | FastAPI (async-first) |
| **Database** | PostgreSQL (with asyncpg driver) |
| **ORM** | SQLAlchemy 2.0 async |
| **Migrations** | Alembic |
| **Authentication** | OAuth2 + JWT (python-jose[cryptography]) |
| **Password hashing** | Passlib (bcrypt) |
| **External APIs** | Public task-metadata API + SendGrid (email) |
| **Async HTTP Client** | Httpx |
| **Caching & Rate Limiting** | Redis + fastapi-limiter |
| **Task Queue** | Celery (with Redis broker) |
| **Retries / Idempotency** | tenacity |
| **Configuration** | Pydantic Settings (v2) |
| **Testing** | pytest, pytest-asyncio, httpx |
| **Logging** | loguru + JSON structured logs |
| **Deployment** | Docker, AWS ECS Fargate, RDS for Postgres, Elasticache for Redis |
| **CI/CD** | GitHub Actions |

**Interview Question Examples:**

* “How do you handle high load in FastAPI?”
* “How do you scale an API horizontally?”
* “How do you reduce DB load in a high-traffic API?”

**Project Features That Cover It:**

* Async programming for concurrency.
* Redis caching for expensive queries.
* Rate limiting to control abusive requests.
* Load balancing with ECS/EKS.

**Reliability & Fault Tolerance**

**Interview Question Examples:**

* “What happens if a request fails mid-operation?”
* “How do you ensure the same request doesn’t run twice?”
* “How do you retry failed requests?”

**Project Features That Cover It:**

* DB transactions & rollback.
* Idempotency keys for POST requests.
* Retry mechanism with exponential backoff.
* Circuit breaker pattern.

**External API Integration**

**Interview Question Examples:**

* “How do you call an external API in FastAPI?”
* “How do you handle timeouts?”
* “What if the API you depend on is down?”

**Project Features That Cover It:**

* Async HTTP calls using httpx.
* Timeout and retry handling.
* Graceful degradation if external API is down.

**9. Deployment & DevOps**

**Interview Question Examples:**

* “How do you deploy a FastAPI app?”
* “How do you containerize it?”
* “What’s the difference between ECS and Lambda?”

**Project Features That Cover It:**

* Dockerfile for containerization.
* AWS ECS/Lambda deployment config.
* CI/CD with GitHub Actions.

**System Design & Architecture**

**Interview Question Examples:**

* “How would you design this app to handle 1M users?”
* “How do you make the app resilient?”
* “How would you scale horizontally?”

**Project Features That Cover It:**

* Modular service-oriented design.
* Caching, rate limiting, async processing.
* Auto-scaling, load balancing, fault tolerance patterns.

# The core technical pattern (high level)

Client → API Gateway / Load Balancer → Multiple FastAPI instances (ASGI, uvicorn/gunicorn) → PostgreSQL (RDS) + Redis (cache, rate-limit) → Optional worker pool (Celery/RQ/Redis Queue) for heavy jobs → Observability services (CloudWatch / Prometheus + Grafana)

# How to handle concurrency, failover & recovery (practical patterns)

* Use connection pooling and tune max\_connections in DB and pool size in async engine.
* Implement graceful shutdown: stop accepting new requests, finish in-flight requests, close DB and redis connections.
* Use health checks and ALB/ECS health probes to remove unhealthy instances.
* Use retries + backoff + queueing for transient failures.
* Backup & restore: schedule RDS backups, test restore procedure. Keep schema and seed data versioned.
* Why async in FastAPI? Show example of blocking vs non-blocking calls.
* How JWTs work (exp, iat, refresh token flow).
* How Alembic migrations work and why they’re critical.
* Indexing strategies, when to add a GIN index (JSONB), how to optimize queries.
* How you implement rate-limiting and caching, with code sketch.
* How you would scale DB connections for thousands of concurrent requests.
* How to make operations idempotent and how to recover from partial failures.
* How to test background workers & CI/CD pipeline.
* How to instrument for latency and error metrics.

**NOTE**

 S**QLAlchemy** is a Python ORM (Object-Relational Mapper) that lets you work with databases using Python classes instead of writing raw SQL. It supports many databases **including PostgreSQL, MySQL, SQLite, Oracle, and others**.

 You just need the right database driver/connector installed (for PostgreSQL, it’s usually psycopg2 or asyncpg for async).

**SQLAlchemy does NOT work with MongoDB**, because:

* SQLAlchemy is designed as an **ORM for relational databases** (SQL-based) like PostgreSQL, MySQL, SQLite.
* MongoDB is a **NoSQL document database**, which uses a totally different data model and query language.