Backend Case Study: Blog Platform (FastAPI, Python, PostgreSQL/SQLite)

Overview

Design and implement the backend for a modern blog platform
using FastAPI and PostgreSQL/SQLite. The backend must support real-time features (SSE, WebSockets), robust authentication, role-based workflows, and production readiness. The API will serve a React frontend, but your focus is on backend code, architecture, and deployment.

Core Backend Skills Demonstrated

- **Python 3.10+**
- **FastAPI** (async endpoints, dependency injection, background tasks)
- **ORM**: SQLAlchemy or Tortoise ORM
- **Database**: PostgreSQL (preferred) or SQLite (for local/dev)
- **Authentication**: JWT and/or session-based auth, secure password hashing (bcrypt/argon2)
- **Authorization**: Role-based access control (RBAC)
- **Real-Time APIs**: Server-Sent Events (SSE), WebSockets (FastAPI native)
- **Testing**: Pytest, HTTPX, FastAPI TestClient
- **API Documentation**: OpenAPI/Swagger (auto-generated), Markdown docs
- **Security**: HTTPS, CORS, CSRF, input validation, secrets management
- **Production Readiness**: Reverse proxy (Nginx/Caddy), SSL, environment config, logging
- **Code Quality**: Type hints, docstrings, comments, modular structure

---

Backend Architecture & Features

1. Authentication & Authorization

- **User Registration & Login**
    - Endpoints: /api/auth/register, /api/auth/login
    - Passwords hashed with bcrypt/argon2
    - JWT tokens (access/refresh) or secure session cookies
    - Role assignment: user, admin, L1 approver, etc.
- **Role-Based Access Control**
    - Decorators/dependencies to restrict endpoints (e.g., admin-only for approvals)
    - Role field in user model

2. Blog Article Management

- **CRUD Endpoints**
    - /api/blogs/ (GET, POST)
    - /api/blogs/{id} (GET, PUT, DELETE)
    - Fields: title, content (markdown/rich text), images, status (pending/approved/rejected), author, timestamps
- **Approval Workflow**
    - Users submit articles (status: pending)
    - Admins/L1 Approvers approve/reject via /api/blogs/{id}/approve or /api/blogs/{id}/reject
    - Only approved articles are public

3. Real-Time Features

- **Server-Sent Events (SSE)**
    - Endpoint: /api/notifications/sse
    - Admins receive real-time notifications when new articles are pending approval
- **WebSockets**
    - Endpoint: /api/blogs/{id}/ws
    - Real-time chat/comments under each blog post
    - Broadcast to all connected clients (no polling)

4. Feature Requests

- **Endpoints**
    - /api/feature-requests/ (GET, POST)
    - /api/feature-requests/{id} (PATCH for status update)
    - Fields: title, description, status (pending/accepted/declined), user, priority/rating

5. Session Management

- **Session Storage**
    - Store session data (e.g., in Redis or DB) for draft blog posts
    - Endpoint to retrieve/restore draft data

6. Security

- **HTTPS**: Enforced in production (SSL via Nginx/Caddy or Azure)
- **CORS**: Configured for frontend domain
- **Input Validation**: Pydantic models, length/type checks
- **Secrets Management**: .env files, never hardcoded
- **API Rate Limiting**: (Optional) via middleware or proxy

- **Firewalls/Private Networks**: Documented in deployment

7. Production Deployment

- **Open Source Stack**
    - Nginx/Caddy as reverse proxy for FastAPI (Uvicorn/Gunicorn)
    - SSL via Let's Encrypt
    - Custom domain via DNS provider
    - Real-time endpoints (SSE/WebSocket) proxied correctly
    - Secrets via environment variables or Docker secrets
- **Azure Stack (if available)**
    - Azure App Service/AKS for FastAPI
    - Azure SQL for DB
    - Azure Redis for sessions
    - Azure Key Vault for secrets
    - Azure DNS & managed SSL

8. Documentation & Testing

- **API Docs**: OpenAPI (auto), Markdown for custom notes
- **README**: Setup, deployment, real-time explanation
- **Unit Tests**: At least two (e.g., blog creation, SSE notification)
- **Code Comments**: Throughout

Example Endpoint List

Endpoint	Method	Auth	Description
/api/auth/register	POST	Public	Register new user
/api/auth/login	POST	Public	Login, returns JWT/session
/api/blogs/	GET	Public	List all approved blogs
/api/blogs/	POST	User	Submit new blog (pending)
/api/blogs/{id}	GET	Public	Get blog details
/api/blogs/{id}	PUT	Author	Edit own blog (if not approved)
/api/blogs/{id}	DELETE	Author	Delete own blog
/api/blogs/{id}/approve	POST	Admin	Approve blog
/api/blogs/{id}/reject	POST	Admin	Reject blog
/api/notifications/sse	GET	Admin	SSE stream for new pending blogs
/api/blogs/{id}/ws	WS	User	WebSocket for blog comments/chat
/api/feature-requests/	GET	User	List feature requests
/api/feature-requests/	POST	User	Submit feature request
/api/feature-requests/{id}	PATCH	Admin	Update feature request status
/api/session/draft	GET	User	Get saved draft for blog submission
/api/session/draft	POST	User	Save draft for blog submission

Sample Implementation Highlights

- **JWT Auth**: Using fastapi.security and pyjwt
- **Password Hashing**: passlib[bcrypt]
- **ORM**: SQLAlchemy models for User, Blog, FeatureRequest, Comment
- **SSE**: Async generator endpoint, yields events on new pending blogs
- **WebSocket**: FastAPI native, manages chat rooms per blog post
- **Testing**: Pytest with FastAPI TestClient, fixtures for DB setup/teardown

- **Deployment**: Dockerfile, Nginx config for SSL and WebSocket/SSE proxying

---

Production Readiness Checklist

- HTTPS enforced (SSL via Nginx/Caddy or Azure)
- Secure secrets management (env vars, not in code)
- CORS configured for frontend
- Real-time endpoints proxied correctly
- Role-based access enforced on all endpoints
- API docs auto-generated and up-to-date
- Unit tests for critical paths
- Logging and error handling in place

Documentation

- **README.md**: Setup, run, test, deploy (local & production)
- **API Docs**: /docs (Swagger UI), plus Markdown for workflows
- **Security Notes**: How data is protected, how secrets are managed

---

Example Unit Test (Pytest)

```
def test_create_blog(client, user_token):
  response = client.post(
    "/api/blogs/",
    headers={"Authorization": f"Bearer {user_token}"},
    json={"title": "Test", "content": "Hello"}
  )
  assert response.status_code == 201
  data = response.json()
  assert data["status"] == "pending"
```

Conclusion

This backend case study demonstrates your ability to:

- Architect and implement a secure, real-time, production-ready API with FastAPI
- Handle authentication, authorization, and role-based workflows
- Deliver real-time features (SSE, WebSockets) efficiently
- Write clean, tested, and well-documented code
- Prepare for cloud or open-source deployment with best practices

---

Deliverables:

- Public GitHub repo (or zip) with code, README, API docs, and tests
- Note on deployment choice (Azure or open source stack)