Backend Code Generation Prompt for Claude

Project Overview (Backend Focus):

Develop the backend for a next-generation flight training management system that surpasses current solutions (e.g., Hinfact, SimOrg) by offering intelligent scheduling, document processing, compliance tracking, real-time analytics, adaptive assessments, and advanced AI integrations.

The backend is built as a set of microservices using Modern C++ (C++17/20) with the Drogon framework and Python for AI/ML tasks. It must support high performance, low latency (including real-time simulator telemetry at 1000Hz), robust security (AES-256, TLS 1.3, zero-trust), and scalable API endpoints. The backend repository structure must be organized clearly to fit within a GitHub repository structure.

Repository Structure:

```
/advanced-pilot-training-platform
  /backend
    /core              # Shared utilities (configuration, logging, error handling)
    /document          # Document processing pipeline and AI-based content extraction
    /syllabus          # Syllabus generation engine and training structure creation
    /assessment        # Competency-based assessment, grading, and biometric integrations
    /user-management   # Authentication, digital logbooks, and role-based dashboards
    /scheduler         # AI-driven scheduling and resource optimization module
    /analytics         # Real-time performance analytics and predictive insights
    /compliance        # Regulatory compliance engine, audit trails, and document verification
    /collaboration     # Backend support for virtual workspaces and messaging integration
    /visualization     # Data services for 3D/AR knowledge maps and simulation visualizers
    /integration       # Connectors for simulators, biometric devices, enterprise systems, and calendars
    /security          # Zero-trust security, blockchain audit trails, and ethical AI governance
```

Code Generation Instructions:

1. Core Framework Components

- Generate a ConfigurationManager class to load settings from environment, files, and database with type-safe access and change notifications.
- Include robust logging (structured and contextual) and thread-safe error handling using modern C++ practices.

2. Document Processing Pipeline

- Create an abstract DocumentProcessor interface with concrete implementations for handling PDF, DOCX, XLSX, HTML, and PPTX.
- Integrate OCR (Tesseract), ML-based structure recognition, and entity extraction.
- Implement asynchronous processing with progress tracking and error reporting.
- Ensure regulatory mapping and audit logging.

3. Syllabus Generation Engine

- Develop a SyllabusGenerator class that extracts learning objectives, competency areas, and training requirements.

- Support template-based syllabus creation with customization, version control, and audit trails.
- Map regulatory standards (FAA, EASA, ICAO) to syllabus components.

4. Real-Time Data Processing

- Create a SimulatorDataProcessor handling high-frequency telemetry (1000Hz) using lock-free queues and multithreading.
- Integrate SIMD optimizations for simulator data.
- Provide real-time and historical data access along with anomaly detection.

5. API Gateway and RESTful Endpoints

- Build a comprehensive API gateway using Drogon framework with JWT-based authentication, input validation, and rate limiting.
- Modular controllers for each backend module (document, syllabus, assessment, scheduler, etc.).
- Auto-generate OpenAPI/Swagger documentation.
- Ensure detailed error handling, logging, and monitoring.

6. Database Access Layer

- Implement a DatabaseManager with connection pooling for PostgreSQL (TimescaleDB for time-series data).
- Include prepared statement caching, transaction management, and migration support.
- Support structured queries and time-series data operations.

7. AI & ML Modules (Python)

- Develop document understanding pipelines:
    - Document classification, text summarization, named entity recognition, and relationship extraction using transformer models.
- Create performance prediction models (TensorFlow/PyTorch) to forecast trainee outcomes and suggest adaptive interventions.
- Build no-code automation workflows (e.g., auto-generation of tasks from document insights), auto-research assistant modules with web scraping, citation tracking, and plagiarism checks.

8. Security & Compliance

- Integrate zero-trust security: AES-256 encryption, TLS 1.3 for data in transit, MFA, and blockchain-backed audit trails.
- Implement ethical AI governance with bias detection, transparency reporting, and user-controlled data ownership.
- Ensure comprehensive audit logging for regulatory compliance.

9. Testing and Performance

- Include unit tests (Google Test) for each module.
- Write integration tests covering end-to-end workflows (document ingestion through syllabus generation).
- Benchmark real-time components and optimize for latency (target <5ms response for high-frequency data APIs).

10. Documentation and CI/CD

- Provide inline documentation and developer guides for each module.
- Generate API documentation using OpenAPI standards.
- Set up CI/CD pipelines (GitHub Actions) for automated testing, linting, and deployment to Vercel (if containerized microservices are deployed there).

GitHub Repository Structure Clarification

Each file should be explicitly marked where it belongs in the GitHub structure, ensuring:

- API-related files reside in /backend/api.
- Core utilities go into /backend/core.
- AI/ML models and scripts should be within /backend/ai.
- Security modules in /backend/security.
- Documentation should be structured in /docs and include API specifications and user guides.

Final Note

The backend code must be modular, thoroughly tested, and follow best practices in error handling, performance optimization, and security. The generated code should be structured explicitly for GitHub repository organization to allow seamless collaboration and version control. This backend will integrate seamlessly with the frontend services.