Backend Assignment: Subscription Based Model

You are tasked with building a **microservice** that manages **user subscriptions** for a SaaS platform. The service should allow users to subscribe to various plans, manage their subscriptions, and retrieve their subscription details. It must follow modern microservices best practices.

Technical Requirements:

- **Language:** Java / Python / Go (your choice)
  Database: MongoDB / PostgreSQL / MySQL (use ORM or raw queries)
- **Authentication:** JWT
- **Design Principles:**
  - MVC or Clean Architecture
    Use environment variables for config
  - Proper error handling & input validation
  - RESTful API standards

Functional Requirements

1. **User Subscription Management**:
   - **Create a Subscription**: Users can subscribe to a plan by providing their user ID and plan details.
   - **Retrieve Subscription**: Retrieve the details of a user's current subscription.
   - **Update Subscription**: Allow users to upgrade or downgrade their subscription plan.
   - **Cancel Subscription**: Allow users to cancel their subscription.
2. **Plan Management**:
   - Define subscription plans with fields like `id`, `name`, `price`, `features`, and `duration`.
   - Allow retrieval of all available plans.
3. **Subscription Status**:
   - Handle subscription statuses (`ACTIVE`, `INACTIVE`, `CANCELLED`, `EXPIRED`).
   - Automatically expire subscriptions after their duration has passed.

Non-Functional Requirements

1. **Scalability**:
   - Design the microservice to handle a large number of subscription requests.
2. **Fault Tolerance**:
   - Implement retry mechanisms for operations like database writes or external API calls.
3. **Performance**:

    - Ensure the microservice can process requests with low latency.
4. **Security**:
    - Use JWT tokens or API keys for authentication.
    - Ensure sensitive data is encrypted in transit and at rest.

Implementation Details

Tech Stack:

- Backend Framework: Choose between Node.js, Go, or Java (Spring Boot).
- Database: PostgreSQL, MySQL, or MongoDB for data persistence.
- Message Queue: RabbitMQ, Kafka, or Redis (for bonus asynchronous updates).

Endpoints:

1. **POST /subscriptions**: Create a new subscription.
2. **GET /subscriptions/{userId}**: Retrieve a user's current subscription.
3. **PUT /subscriptions/{userId}**: Update a user's subscription.
4. **DELETE /subscriptions/{userId}**: Cancel a user's subscription.
5. **GET /plans**: Retrieve available subscription plans.

Assessment Criteria:

1. **Code Quality**: Modular, readable, and follows best practices.
2. **API Design**: Intuitive and adheres to RESTful conventions.
3. **Documentation**: Clear and comprehensive API docs and setup instructions.
4. **Bonus Features**: Implementation of advanced requirements.