Customer and Plan Management System

Description:

Develop a Spring Boot REST API for a Telecom Customer and Plan Management System. This system allows telecom companies to manage customers, their subscriptions, and the available plans. The API should support basic CRUD (Create, Read, Update, Delete) operations and include proper validation, error handling, and documentation.

Requirements:

1. Entities:

- **Customer:**
  - id (Long, auto-generated)
  - name (String)
  - email (String)
  - phoneNumber (String)
  - address (String)
  - planId (Long, Foreign Key to Plan)
- **Plan:**
  - id (Long, auto-generated)
  - name (String)
  - description (String)
  - price (Double)
  - dataLimit (Integer, in GB)
  - callMinutes (Integer)
  - smsLimit (Integer)
- **Subscription:**
  - id (Long, auto-generated)
  - customerId (Long, Foreign Key to Customer)
  - planId (Long, Foreign Key to Plan)
  - startDate (Date)
  - endDate (Date)
  - status (String, e.g., ACTIVE, INACTIVE)

2. Endpoints:

- **Customer Endpoints:**
  - POST /customers - Create a new customer.
  - GET /customers - Retrieve a list of all customers.
  - GET /customers/{id} - Retrieve details of a specific customer by ID.
  - PUT /customers/{id} - Update details of a specific customer by ID.
  - DELETE /customers/{id} - Delete a specific customer by ID.

- **Plan Endpoints:**
  - POST /plans - Create a new plan.
  - GET /plans - Retrieve a list of all plans.
  - GET /plans/{id} - Retrieve details of a specific plan by ID.
  - PUT /plans/{id} - Update details of a specific plan by ID.
  - DELETE /plans/{id} - Delete a specific plan by ID.
- **Subscription Endpoints:**
  - POST /subscriptions - Create a new subscription.
  - GET /subscriptions - Retrieve a list of all subscriptions.
  - GET /subscriptions/{id} - Retrieve details of a specific subscription by ID.
  - PUT /subscriptions/{id} - Update details of a specific subscription by ID.
  - DELETE /subscriptions/{id} - Delete a specific subscription by ID.

3. Features:

- **Validation:** Ensure all inputs are validated (e.g., no null names, valid email formats, dates).
- **Error Handling:** Return appropriate HTTP status codes and error messages.
- **Database:** Use an in-memory database (e.g., H2/Oracle/MySQL) for development purposes.
- **Documentation:** Use Swagger/OpenAPI to generate API documentation.

4. Documentation:

- **README.md:**
  - Project overview.
  - How to set up and run the project.
  - Explanation of the endpoints and their usage.
  - Example requests and responses.
- **API Documentation:**
  - Use Swagger/OpenAPI annotations to generate detailed API documentation.

Instructions:

1. **Set Up Spring Boot Project:**
   - Create a new Spring Boot project using Spring Initializr or any IDE.
   - Add dependencies for Spring Web, Spring Data JPA, H2 Database, and Spring Boot Starter Validation.
2. **Develop the Application:**
   - Create the entities, repositories, services, and controllers.
   - Implement the required endpoints and business logic.
   - Ensure proper validation and error handling.
3. **Configure the Database:**
   - Set up an in-memory H2 database for development and testing purposes.

4. **Document the API:**
    - Use Swagger/OpenAPI for generating API documentation.
    - Ensure all endpoints, models, and possible responses are well-documented.
5. **Testing:**
    - Write unit and integration tests for the service and controller layers.