Bank Management REST API with Spring Boot Data JPA Rest

Overview:

This project is a RESTful API built with Spring Boot and Spring Data JPA, designed for managing bank accounts and customers. It provides endpoints for performing CRUD operations on both accounts and customers, utilizing Spring Data repositories for seamless database interactions.

Key Components:

1. **Controllers:**
   - **AccountController** (/api/accounts): Manages operations related to bank accounts.
     - **Endpoints:**
       - GET /api/accounts: Retrieve all accounts.
       - GET /api/accounts/{id}: Retrieve an account by its ID.
       - POST /api/accounts: Create a new account.
       - PUT /api/accounts/{id}: Update an existing account.
       - DELETE /api/accounts/{id}: Delete an account.
   - **CustomerController** (/api/customers): Handles operations related to customers.
     - **Endpoints:**
       - GET /api/customers: Retrieve all customers.
       - GET /api/customers/{id}: Retrieve a customer by its ID.
       - POST /api/customers: Create a new customer.
       - PUT /api/customers/{id}: Update an existing customer.
       - DELETE /api/customers/{id}: Delete a customer.
2. **Entities:**
   - **Account**: Represents a bank account with fields id, accountNumber, balance, and a relationship with Customer.
   - **Customer**: Represents a customer with fields id, name, address, and associated accounts.
3. **Repositories:**
   - **AccountRepository**: Interface for account-related database operations.
   - **CustomerRepository**: Interface for customer-related database operations.
4. **Services:**
   - **AccountService**: Contains business logic for account management, including error handling.
     - **Services Provided:**
       - getAllAccounts(): Retrieve all accounts.
       - getAccountById(Long accountId): Retrieve an account by its ID.
       - createAccount(Account account): Create a new account.
       - updateAccount(Long accountId, Account accountDetails): Update an existing account.
       - deleteAccount(Long accountId): Delete an account.
       - transferFunds(Long fromAccountId, Long toAccountId, double amount): Transfer funds between accounts.

        - getAccountsByCustomer(Long customerId): Retrieve all accounts belonging to a specific customer.
    - **CustomerService**: Encapsulates business logic for customer management, including error handling.
        - **Services Provided:**
            - getAllCustomers(): Retrieve all customers.
            - getCustomerById(Long customerId): Retrieve a customer by its ID.
            - createCustomer(Customer customer): Create a new customer.
            - updateCustomer(Long customerId, Customer customerDetails): Update an existing customer.
            - deleteCustomer(Long customerId): Delete a customer.
            - getCustomerByEmail(String email): Retrieve a customer by their email address.
            - getCustomerAccountsSummary(Long customerId): Provide a summary of all accounts held by a customer.
5. **Exception Handling:**
    - **ResourceNotFoundException**: Custom exception mapped to HTTP status 404 to handle cases where requested resources are not found.

Features:

- **Cross-Origin Resource Sharing (CORS)**: Enabled for all origins (@CrossOrigin("*")), allowing client-side applications from any domain to access the API.
- **CRUD Operations**: Comprehensive endpoints for creating, reading, updating, and deleting accounts and customers.
- **Error Handling**: Graceful handling of scenarios where resources are not found, with appropriate HTTP status codes.
- **Fund Transfer**: Secure method to transfer funds between different accounts.
- **Customer Account Management**: Efficient methods to manage and retrieve customer-specific account information.

Technology Stack:

- Java
- Spring Boot
- Spring Data JPA
- Hibernate
- Lombok (for reducing boilerplate code)

Purpose:

The API is designed to facilitate basic CRUD operations for managing bank accounts and customer details, along with additional services like fund transfers and customer-specific account summaries. This makes it suitable for integration with frontend applications in a banking system, providing a robust backend for financial transactions and customer management.