**Banking Application**

**Status Codes:**

* **100-199:** Informational. Eg: Resource has been processed.
* **200-299:** **Request got successful.**
* 201: Created
* 200: Success
* 204: No Content
* **300-399:** **Redirection.**
* 301: Resource has been permanently moved to another URL
* 302: Resource has been temporarily moved to another URL
* **400-499: Client Side Error**
* 404: Not Found
* 401: Unauthorized, you need to login first.
* 403: You are not the admin, but normal user.
* **500-599: Server Side error**
* 502: Bad Gateway

**ResponseEntity class:**

Represents the entire HTTP Response including

* Status code (e.g., 200 OK, 404 Not Found)
* Headers (e.g., Content-Type, Authorization)
* Body (the actual data being returned)

**Note: Headers** are key-value pairs that provide metadata about the response. They are not part of the actual response body (the data), but they give important information to the client (like a browser or another service) about how to handle the response.

**Why Extend RuntimeException for Custom Exceptions in Spring?**

**Cleaner code**: To avoid try-catch and throws

**Flow of Exception Handling:**

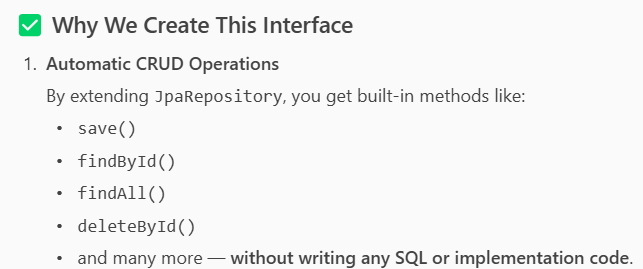
**Postman Client 🡪 Controller 🡪 Service 🡪 Repository 🡪 DB**

Now suppose error occurs at Service, so it will throw to **GlobalExceptionHandler 🡪** **Error Response Model**. After formatting ERM wil send it back to client.

1. Configuration in the application.properties file.
2. In entity package create the Account class.

* **@Getter, @Setter -** generates getters and setters for all the fields.
* **@NoArgsConstructor –** generates a no argument constructor.
* **@AllArgsConstructor –** generates a constructor with all fields as parameters.
* **@Table(name=”accounts”) -** specifies the table name in the database that this entity maps to. Otherwise, the table name would default to the class name (account).
* **@Entity –** marks the class as JPA entity.
* **@Id –** marks the field as primary key.
* **@GeneratedValue(strategy = GenerationType.IDENTITY) –** specifies that primary key is auto increment.

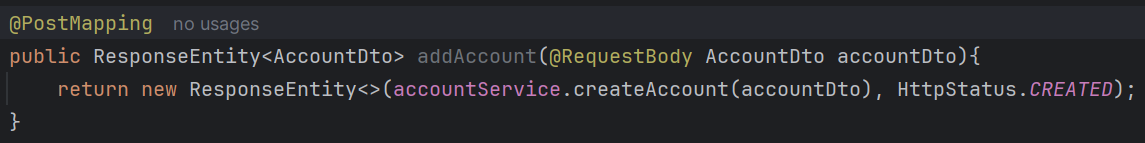
1. In the repository package, create AccountRepository Interface.



* Repository layer is responsible for interacting with databases.
* Seperation of concerns: separates business logic(service) with data access logic(repository).
* If we need to change the database, we only need to change the repository layer, not business logic.

1. Start creating REST APIs. Eg: addAccount()

In Controller Layer:



In Service Layer(AccountServiceImpl):

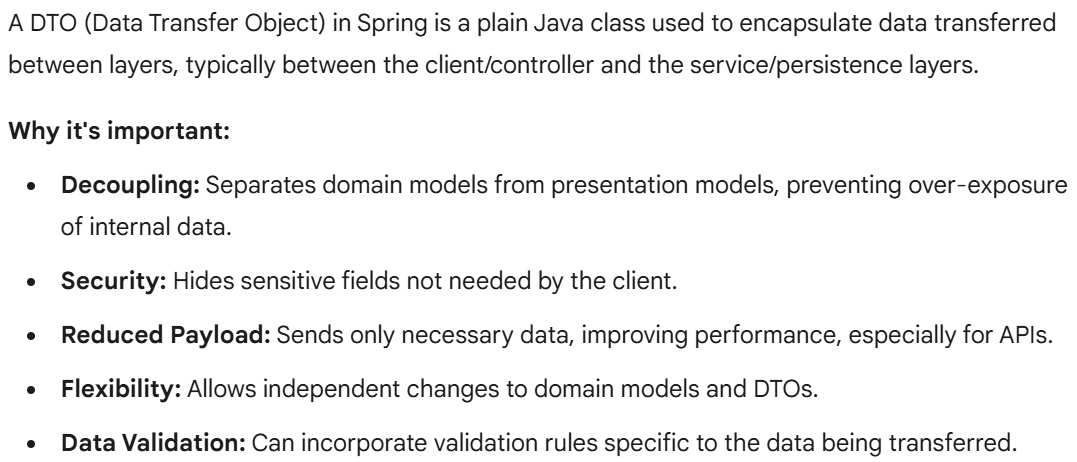
A screen shot of a computer

AI-generated content may be incorrect.

**@RequestBody –** reads the JSON body of client HTTP request, and converts it into Java object(AccountDTO) using JSON parser Jackson, and then pass the object to method as parameter.

A white background with black text

AI-generated content may be incorrect.

****

1. **In the Service layer first, we created AccountService interface, then implemented it. Why?**

**A white background with black text

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**