📌 Deep-Dive Backend Review (Full Summary)

✅ We'll go layer by layer, covering each class & its purpose.

1️⃣ Entity Layer (Models)

✅ **Location:** `com.cts.smartspend.entity`
✅ **Purpose:** Represents database tables using JPA.

Class	Purpose	Key Relationships
User	Stores user details like `username`, `password`, `role`.	One-to-Many with `Expense`.
Category	Stores different expense categories (`Food`, `Transport`, etc.).	One-to-Many with `Expense`, One-to-Many with `Budget`.
Expense	Stores expense details like `amount`, `date`, `description`.	Many-to-One with `Category`, Many-to-One with `User`.
Budget	Stores budget limits for categories within a date range.	Many-to-One with `Category`.

🔹 Important Notes:

- `@ManyToOne` & `@OneToMany` manage relationships.
- `@JsonManagedReference` & `@JsonBackReference` prevent infinite recursion in JSON responses.
- `@Enumerated(EnumType.STRING)` in `User` stores `role` as `ADMIN` or `USER`.

2️⃣ Repository Layer (Database Queries)

✅ **Location:** `com.cts.smartspend.repo`
✅ **Purpose:** Handles database interactions with `JpaRepository`.

Repository	Entity Managed	Custom Methods
UserRepo	`User`	`findByUsername(username)`, `findByRole(role)`.
CategoryRepo	`Category`	`findByName(name)`.

Repository	Entity Managed	Custom Methods
ExpenseRepo	`Expense`	`findByCategoryId(id)` , `findByUserId(id)` , `findByCategoryIdAndDateRange(...)` .
BudgetRepo	`Budget`	`findBudgetByCategoryAndDate(...)` , `findByCategoryId(id)` .

🔹 Important Notes:

- Queries automatically work because `JpaRepository` provides built-in CRUD operations.
- Custom queries like `findByCategoryIdAndDateRange()` help fetch specific data.

3️⃣ DTO Layer (Data Transfer Objects)

✅ **Location:** `com.cts.smartspend.dto`
✅ **Purpose:** Separates entity objects from request/response data.

DTO Class	Purpose
UserDTO	Used for user-related API requests. **Hides password in response.**
LoginDTO	Stores login request data (`username` , `password`).
LoginResponseDTO	Sends back JWT token after successful login.
ExpenseDTO	Represents expense data in API responses.
ExpenseResponseDTO	**Includes** `remainingBudget` when retrieving expenses.
BudgetDTO	Used for setting & updating budgets. **Includes** `categoryId` **in request and** `categoryName` **in response.**

🔹 Important Notes:

- DTOs **prevent exposing sensitive entity details** in API responses.
- **BudgetDTO was updated to include** `categoryName` **in response.** ✅

4️⃣ Service Layer (Business Logic)

✅ **Location:** `com.cts.smartspend.serviceImpl`
✅ **Purpose:** Implements actual logic & interacts with repositories.

Service Class	Key Responsibilities
UserService	Handles user creation, login, role-based retrieval, encryption with `BCryptPasswordEncoder`.
CategoryService	Handles category creation, checking for duplicates, prevents deletion if expenses exist.
ExpenseService	Adds expenses, calculates `remainingBudget`, prevents deletion if linked to other entities.
BudgetService	Manages budget settings, ensures `categoryId` is included in request and `categoryName` in response.

🔹 Important Notes:

- `@Transactional` ensures database consistency.
- **Password is always stored as an encrypted hash using `BCryptPasswordEncoder`.**
- **JWT token is generated upon login and must be sent for accessing protected endpoints.**

5️⃣ Controller Layer (API Endpoints)

✅ **Location:** `com.cts.smartspend.controller`
✅ **Purpose:** Exposes APIs for frontend communication.

Controller Class	Main Endpoints
UserController	`/users/add`, `/users/get/all`, `/users/delete/{id}`, `/login` (for authentication).
CategoryController	`/category/add`, `/category/get/all`, `/category/delete/{id}`.
ExpenseController	`/expenses/add`, `/expenses/get/all`, `/expenses/delete/{id}`.
BudgetController	`/budget/set`, `/budget/get/all`, `/budget/delete/{id}`.

🔹 Important Notes:

- **Only ADMIN users can add/delete categories and budgets.** (`@PreAuthorize("hasAuthority('ADMIN')")`)
- **JWT token must be sent in Postman for protected endpoints.**
- `ResponseEntity` properly returns HTTP status codes (`200 OK`, `201 Created`, `404 Not Found`, etc.).

6️⃣ Security Layer (Spring Security + JWT)

✅ **Location:** `com.cts.smartspend.security`
✅ **Purpose:** Manages authentication & authorization using JWT tokens.

Class	Purpose
CustomUserDetails	Implements `UserDetails`, providing authentication details.
CustomUserDetailsService	Loads user data from `UserRepo` for authentication.
JwtUtils	Generates & validates JWT tokens.
JwtAuthFilter	Intercepts requests to check JWT validity.

🔹 Important Notes:

- **JWT tokens are generated using `JwtUtils` and sent back upon successful login.**
- **All protected routes check JWT before allowing access.**
- **Spring Security config (`SecurityConfig.java`) ensures only admins can modify budgets & categories.**

📌 Backend Flow from Login to API Calls

1️⃣ **User logs in (`POST /login`)** → Spring Security authenticates the user, generates JWT token.
2️⃣ **Frontend sends API requests with JWT token** (e.g., `GET /expenses/get/all`).
3️⃣ **JWT token is validated in `JwtAuthFilter`** before allowing access.
4️⃣ **Controllers process the request** and return the required data (e.g., expenses, categories, budgets).