Which database you have chosen and why?

I have chosen **PostgreSQL** as the database because it is **robust, scalable, and supports relational data**. It works well with **Prisma ORM**, ensuring efficient data handling, migrations, and type safety. PostgreSQL is also great for handling structured data like categories, subcategories, and items with relationships.

3 things that you learned from this assignment?

- Database Design & Relationships – You structured a relational database with **categories, subcategories, and items**, understanding how **foreign keys** and **relations** work in PostgreSQL.

- API Documentation with Swagger – You explored how to integrate **Swagger** for documenting and testing APIs, making it easier for developers to understand and interact with your endpoints.

- **Error Handling & API Responses** – You learned how to **gracefully handle errors** in Express.js, sending proper status codes and messages for both success and failure cases.

What was the most difficult part of the assignment?

The **most difficult part** of the assignment likely involved:

◇ Database Schema & Relationships

- Designing the **category, subcategory, and item structure** while ensuring proper **foreign key constraints** in PostgreSQL.
- Managing **cascading deletes** (e.g., deleting a category should remove related subcategories and items).

◇ Prisma Migrations & Debugging

- **Applying migrations** correctly without breaking the database.
- Handling **schema changes** efficiently when modifying models.

◇ API Error Handling & Validation

- Implementing **proper error responses** for missing fields, invalid data, or failed database operations.
- Ensuring **atomicity** (e.g., preventing partial updates that could leave the database in an inconsistent state).

What you would have done differently given more time?

1️⃣ Implement User Authentication

- Create a **User model** in Prisma with fields like `id`, `email`, `password` (hashed), and `role`.
- Use **bcrypt** to hash passwords before storing them.

2️⃣ Add Cookie-Based Login

- On successful login, set an **HTTP-only** cookie with a **JWT or session ID**.
- Use `express-session` for **session-based authentication** or `jsonwebtoken` for **JWT-based authentication**.

3️⃣ Protect Routes with Authentication Middleware

- Middleware to check if a user is **logged in** before allowing access to protected routes.
- Restrict actions (e.g., only admins can **add/edit/delete categories**).

4️⃣ Implement Logout & Token Expiry

- Clear the authentication cookie on **logout**.

- Set **automatic session expiration** for better security.