Level 1 – Project Tasks

1. Refactor the code in `ItemController` to prevent code duplication.
2. Prevent SQL Injection by using `PreparedStatement`.
3. Explain the difference between `RequestDispatcher.forward()` and `response.sendRedirect()`.
4. Finalize the project as a complete CRUD application and make sure the application is tested.
5. When adding an item successfully, show a popup message: **"Item added successfully"**.
6. When updating an item successfully, show a popup message: **"Item updated successfully"**.
7. When deleting an item successfully, show a popup message: **"Item deleted successfully"**.
8. When inserting an item and the item name already exists, show a popup message: **"Item name already exists in the system"**.
9. When updating an item and the item name already exists, show a popup message: **"Item name already exists in the system"**.
10. When deleting an item, do not remove it from the database; it must be removed only from the screen.
11. Add front-end validation for all inputs: **name**, **price**, and **total number**.

Level 2 – Project Tasks

1. Create a new table named `ITEM_DETAILS`.
2. Each row in the `ITEM` table must have only one corresponding row in `ITEM_DETAILS` (One-to-One relationship).
3. Add a **UNIQUE constraint**.
4. Table columns:
5. `id`
6. `desc`
7. `issue_date`
8. `expiry_date`
9. Attach the SQL file.
10. Modify `load-items.jsp` to add new columns:
11. `desc`
12. `issue_date`
13. `expiry_date`
14. Use **JOIN** in the query.

UI Buttons and Behavior

Add Item Details Button (`add_item_details`)

- If a row in the `ITEM` table has a corresponding row in `ITEM_DETAILS`, the button must be **hidden**.
- If a row in the `ITEM` table does not have a corresponding row in `ITEM_DETAILS`, the button must be **visible**.

Delete Item Details Button (`delete_item_details`)

- If a row in the `ITEM` table has a corresponding row in `ITEM_DETAILS`, the button must be **visible**.
- If a row in the `ITEM` table does not have a corresponding row in `ITEM_DETAILS`, the button must be **hidden**.

Impact

- Handle the **delete** button to remove the row from `ITEM_DETAILS`, then remove it from `ITEM`.
- The **update** button must include all inputs from both `ITEM` and `ITEM_DETAILS` and update all of them.

Authentication (Login & Signup)

1. Create a new page for **Login** and **Signup**.
2. Create a table named `USERS` with the following columns:
3. `id`
4. `name`
5. `email`
6. `password`
7. Add a **UNIQUE constraint** on `email`.
8. Create a **User model**.
9. Create a **Service** and **Service Implementation** for users.
10. Create a new **Controller** to handle login and account creation.

Login

- If login is successful, move the user to the **main page**.
- If login fails, redirect to the **login page** with a validation message:
- **"Invalid username or password"**.

Signup

- If signup is successful, move the user to the **main page** or **login page**.
- If signup fails, redirect to the **signup page** with a validation message.

Session and Cookies

1. Apply **Session** and **Cookies**.
2. Add a **Logout** button that is visible only when the user is logged in.
3. Logout action must:
4. Redirect the user to the **login page**.
5. Set the session attribute to `false` or remove the session attribute.
6. Remove cookies.