CMPS 350 Web Development Fundamentals - Assignment # 5

Due Date Wednesday, May 2, 2025, @11: 59 PM

Overview

In Assignment 4, you developed the **NutriSnap** app by:

• Creating a backend that managed meal data using a meals.json file.

• Building a frontend that allowed users to view, add, edit, delete, and filter meals using Next JS.

In Assignment 5, you will **upgrade** your **NutriSnap** app by connecting it to a real database using **Prisma ORM** with **SQLite**.

You must build on your existing NutriSnap project — either:

- Use your own NutriSnap code from Assignment 4,**OR**
- Start from the Assignment 4 base code that was previously provided.

⚠ Important:

- No new base code is provided for Assignment 5. You must build on top of what you have from Assignment 4.

Requirement

- Implement a complete **backend database** using Prisma ORM and SQLite.
- Ensure you have a **working frontend UI** connected to the new backend.
- Deliver a complete, functioning **NutriSnap** app — backend and frontend fully integrated.

⚠ **Critical Note**:

- Students who **only implement the Prisma schema and backend** without a **working frontend UI** will receive **zero for the UI portion**.

Part A - Implementing the Database [50%]

You must design your database based on the following **Entity Relationship Diagram (ERD) as shown in figure 1**.

- One **User** can have many **Meals**.
- Each **Meal** belongs to one **User**.
- Tags are stored as an **array** inside the Meal. You are also allowed to make Tags to be their own table. In that case you must create a many to many relationship.

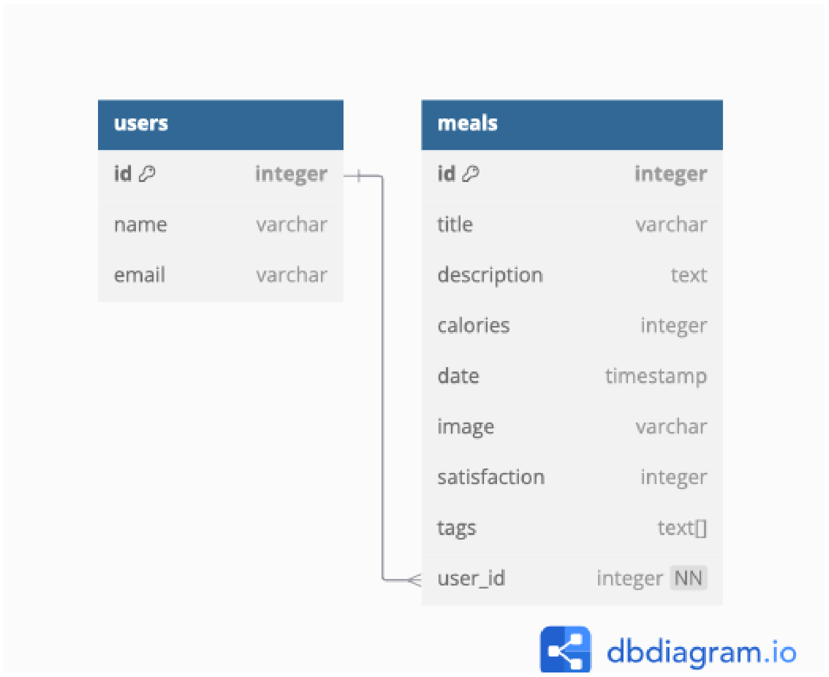

Figure 1 Database ER Diagram

The repository should implement the same functionally as the file-based repository you implemented in assignment 4. See the table below for further details. Also, you should create **seed.js** that will initialize the database using the **data/meals.json and users.json** file given.

You must implement the following backend repository methods using **only Prisma Client methods**:

Method	Description
getMeals()	Fetch all meals.
getMeal(id)	Fetch a meal by ID.
addMeal(meal)	Add a new meal.
updateMeal(id, meal)	Update a meal by ID.
deleteMeal(id)	Delete a meal by ID.
deleteMeals()	Delete all meals.
getMealByDate()	Fetch meals logged on a specific date.
addSatisfaction()	Add/update satisfaction rating for a meal.
getMealByTag(tag)	Fetch meals associated with a specific tag.
getAverageTagSatisfaction	Return the average tag satisfaction
getMealSummary()	Returns a summary

🏳 You must use **Prisma queries** (findMany, create, update, aggregation, etc.).
🏳 You are **not allowed to use** JavaScript array methods like map, filter, or reduce.

Part B - Implementing the UI using Next.js and React [40%]

If you did not complete the user interface (UI) in Assignment 4, you are required to implement the UI now using Next.js. (Please review the UI requirements outlined in Assignment 4.)

If you already built the UI in Assignment 4, you must now connect it to your new Prisma-based backend. The functionality of your app must match the base solution, as demonstrated in the provided base code in assignment 4.

Make sure to test your entire application carefully to verify that all functionalities — viewing, adding, editing, deleting, filtering meals, and viewing summaries — are working correctly.

After completing the assignment, fill out the **TestingDoc-Grading-Sheet.docx**, save it inside your Assignment-5 folder, and push your project to your GitHub repository.

Grading

Part	Points
Prisma database schema and migration	15%
Seeding the data	5
Repository methods using Prisma	40%
Fully working frontend UI	40%