System Design for HealthQuest App

EECS 3311 E

Group 6

Sprint 1

By:
Humza Inam,
Luqmaan Irshad,
Abrar Noman,
Aditya Bhalla,
Prabhjyot Grewal

CRC CARDS

Calories

Responsibilites	Collaborators
: • Store user-specific calorie burn data for a given date. • Maintain fields for `userId`, `date`, and `caloriesBurned`. • Default `caloriesBurned` to 0 if not specified.	• **User**: References the `User` model for linking `userId`. • **Mongoose**: Used to define the schema and manage data persistence in the database.

Hydration

Responsibilites	Collaborators
• Stores water consumption data for users. • Tracks the number of cups consumed (`cupsConsumed`) on a specific date (`date`). • Links each entry to a user (`userId`). • Provides a default value of 0 for `cupsConsumed` if not specified.	• **User**: References the `User` model for linking `userId`. • **Mongoose**: Used for schema definition and data management.

Login

Responsibilites	Collaborators
• Stores user login data, including login time and IP address. • Tracks the login time (`loginTime`) with a default value of the current time. • Links each login entry to a user (`userId`). • Stores the IP address (`ipAddress`) associated with the login.	• **User**: References the `User` model for linking `userId`. • **Mongoose**: Used for schema definition and data persistence.

Meal

Responsibilites	Collaborators
• Stores meal data, including the meal name, nutritional information (calories, protein, carbohydrates, fats), and the associated user. • Links each meal entry to a user (`userId`). • Tracks the meal date (`date`), with a default value of the current date in a specific timezone.	• **User**: References the `User` model for linking `userId`. • **Mongoose**: Used to define the schema and manage data persistence in the database.

Steps

Responsibilites	Collaborators
• Stores step count data for users, including the number of steps (`steps`) on a specific date (`date`). • Links each entry to a user (`userId`). • Provides a default value of 0 for `steps` if not specified.	• **User**: References the `User` model for linking `userId`. • **Mongoose**: Used to define the schema and manage data persistence in the database.

User

Responsibilites	Collaborators
• Stores user data, including personal details (name, email, bio, etc.), credentials (password), and user progress (level, `xp`, goals). • Handles password hashing before saving the user document in the database. • Tracks user goals for calories and hydration, as well as streak and last login date. • Provides default values for certain fields, such as `level`, `xp`, `calories`, and `hydration`.	• **bcrypt**: Used for hashing the password before saving it to the database. • **Mongoose**: Used for schema definition, validation, and database operations.

Authentication Controller

Responsbillites	Collaborators
• Handles user registration and login. • Validates email and password during login. • Hashes password during registration and compares it with the stored hashed password during login. • Generates JWT tokens upon successful registration or login. • Logs user login events in the database. • Responds with appropriate messages and status codes.	• **User**: References the `User` model to find and manage user data. • **JWT**: Used for generating authentication tokens. • **bcryptjs**: Used for password hashing and comparison. • **Login**: References the `Login` model to log login events.

Meal Controller

Responsbillites	Collaborators
• Handles the addition of a new meal (`addMeal`), including meal name, calories, protein, carbohydrates, and fats. • Retrieves all meals for a specific user (`getMeals`). • Sends appropriate responses for success and error situations.	• **Meal**: References the `Meal` model to store and retrieve meal data for users. • **User**: The controller assumes the `userId` is passed via middleware for user-specific meal tracking.

User Controller

Responsbillites	Collaborators
• **Login**: Validates user credentials, generates a JWT token, and logs login events. • **Register**: Registers a new user by checking for existing users, hashing the password, and saving the new user. • Handles error cases such as missing fields, invalid credentials, and server issues. • Logs successful user actions and errors for debugging purposes.	• **User**: References the `User` model for storing and retrieving user data. • **JWT**: Used to generate JWT tokens for authenticated users. • **bcryptjs**: Used to hash and verify passwords. • **Login**: References the `Login` model to log user login events (IP address, login time).

Software Architecture Diagram