CSCI 5448 Project - 7

Project

- Title: Calorie Tracker
- Members: Gaurav Roy & Varun Bhaskara

Final State of System Statement

- Our goal was to build a web application that would allow users to track their calories based on their input of meals consumed and exercises performed and see how they stand against the Calorie or Macro goal they set for themselves. I think we have done a very good job of achieving this goal.

  These were some of the features that we set out to achieve during the project planning.

  1. User registration and login: The application will allow users to register and log in to their accounts, which will enable them to access and use the application.

     Status: Achieved
     The application successfully implements user registration and login functionality using Spring Boot's spring-security feature with JWT-based authorization and authentication.

  2. Setting User Profile: The application will take input from users such as height, weight, age and setting up the user's profile; this should be useful to calculate the user's maintenance calories for the day.

     Status: Achieved
     The application collects user input for height, weight, and age to set up the user's profile. This information is utilized to calculate the user's maintenance calories for the day.

  3. Food intake tracking: The application will allow users to track the food they eat throughout the day, including the name of the food, serving size, and nutritional content.

     Status: Achieved

The application effectively allows users to track their daily food intake by entering the name of the food, serving size, and nutritional content. This functionality is achieved by leveraging the Nutritionix API.

4. Calorie calculation: The application will calculate the total number of calories consumed by the user based on the food they have entered.

   Status: Achieved
   The application calculates the total number of calories consumed by the user based on the entered food items. Interactive components using React are utilized to display the calculated calories and other macros consumed per day, helping users track their progress towards their goals.

5. Notifier and Report: The application will notify or remind the user about the analysis of the macro-nutrients that the user has consumed for the day.

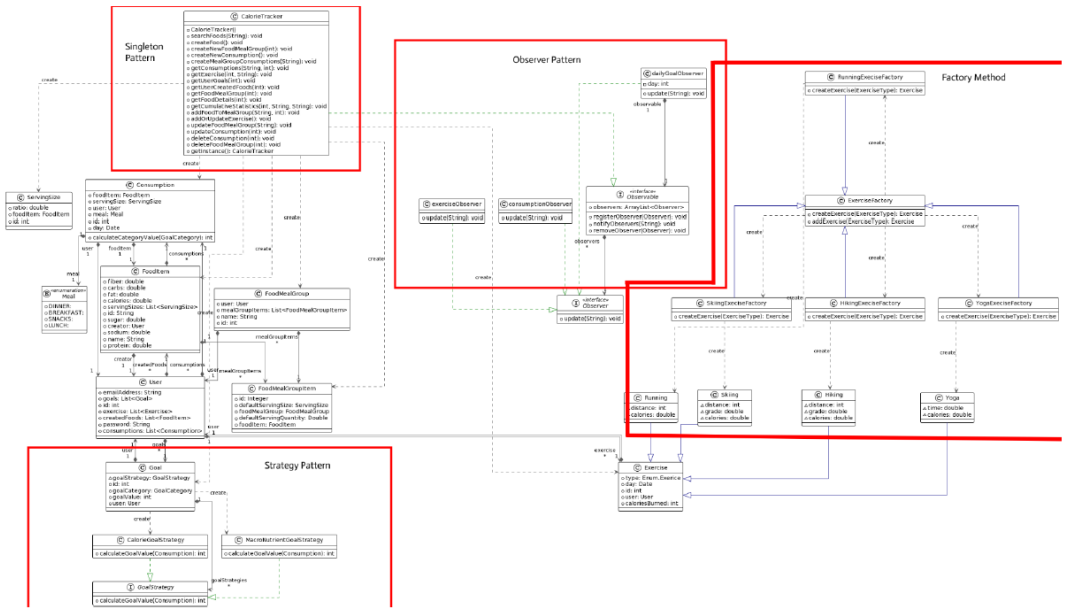
   Status: Achieved
   The application successfully implements customizable summaries and reports, such as averages and a comparison of calories eaten vs. calories burned, which are generated using interactive React components.

6. Scalability: The application should be scalable to accommodate a growing user base and varying requirements changes as new features are added.

   Status: Not Achieved
   Although the backend code can easily accommodate new features, the frontend code in React requires cleanup to improve scalability. Once the frontend code is organized and optimized, it will be easier to integrate new features and accommodate a growing user base.

Final Class Diagram and Comparison Statement

- Project 5 class diagram:



- Project 7 class diagram: (Attached in the end as a PDF to maintain quality)

  We have made a lot of changes since making the above shown Project 5 class diagram

  - Initially, we thought that storing user consumptions in a relational database like MySQL would be feasible. Hence, we had decided to use normalized tables for faster querying. This resulted in more entities representing FoodItem in the older UML compared to the new one, but we decided against it and used MongoDB.

  - Our initial intention was to allow users to choose the goal they wanted to track, and we had planned to use the Strategy pattern to

separate the logic accordingly. However, during implementation, we decided to incorporate this logic in the frontend instead and ended up not utilizing the Strategy pattern.

- In the new UML diagram, it is evident that we have deviated from the originally planned patterns and incorporated a greater number of patterns than before, thanks to the integration of SpringBoot.

- Some patterns we ended up using are:
    - **Singleton**: In Spring Boot, Singleton's are used extensively to manage the lifecycle of Beans, which are objects managed by spring Inversion of Control container. When a Spring Boot application starts up, the Spring IoC container creates a single instance of each bean defined in the application context. These beans are then shared by all components that require them, ensuring that there is only one instance of each bean in the application (Reference:https://www.baeldung.com/spring-boot-singleton-vs-beans)
    - **Proxy**: In Spring Boot, the proxy pattern is extensively used, wherein one object the proxy control's access to another object, in our code the repository classes directly access the DB via model classes which are the proxies.
    - **Template**: We use the template pattern to calculate maintenance calories for a user based on their height, weight, age, gender and exercise frequency. Calculation of maintenance calories requires the calculation of basal metabolic rate which varies from one gender to another.By using the template pattern, we were able to implement two subclasses (Male and Female) that provide specific implementations for the BMR calculation. This results in code reusability and maintainability, as the common code for calculating maintenance calories is abstracted into a superclass, and the subclass implementations can be easily updated without affecting the overall algorithm.
    - **Factory**: We use factory method to allow users to create different types of exercises and calculate calories burned doing that exercise based on approximate estimation of Metabolic Equivalent of Task of that exercise. With this we ensure the encapsulation of the creation of different types of

exercises and their corresponding calorie calculation algorithms. This allows for a flexible and extensible design, where new types of exercises can be added without modifying the existing code.

- **Iterator:** We use the Iterator pattern in our code to calculate statistics for the user. When it comes to computing statistics, we need to iterate over a collection of objects, and the Iterator pattern offers an elegant solution for sequentially accessing the elements of an aggregate object without revealing its internal structure. By leveraging the Iterator pattern, we ensure an efficient and scalable approach to calculate statistics while promoting code flexibility, reusability, and maintainability.
- **MVC:** The controller's and the model's in the backend make up for the MV part of MVC and the frontend with React is the view we render which interacts with the Models and Controllers.

Third-Party code vs. Original code Statement

- To use Spring Security in our project, we picked code's from this repository as https://github.com/bezkoder/spring-boot-security-jwt-auth-mongodb
- To understand how to model entities in MongoDB and create DTO's, we used this repository as a reference: https://github.com/Jelani-Ak/twitter-clone-project
- To learn to use React with Redux to manage state, we used code from this repository https://github.com/bradtraversy/proshop_mern
- Idea for the components in the frontend was largely inspired from this repository https://github.com/mileshenrichs/easycal
- To get started with SpringBoot we found this tutorial extremely helpful https://youtu.be/qVNOw9TWwxo

- Plus a lot of help from StackOverflow and Baeldung for a few parts within the code, references cited in comments within the code.

Statement on the OOAD process for your overall Semester Project

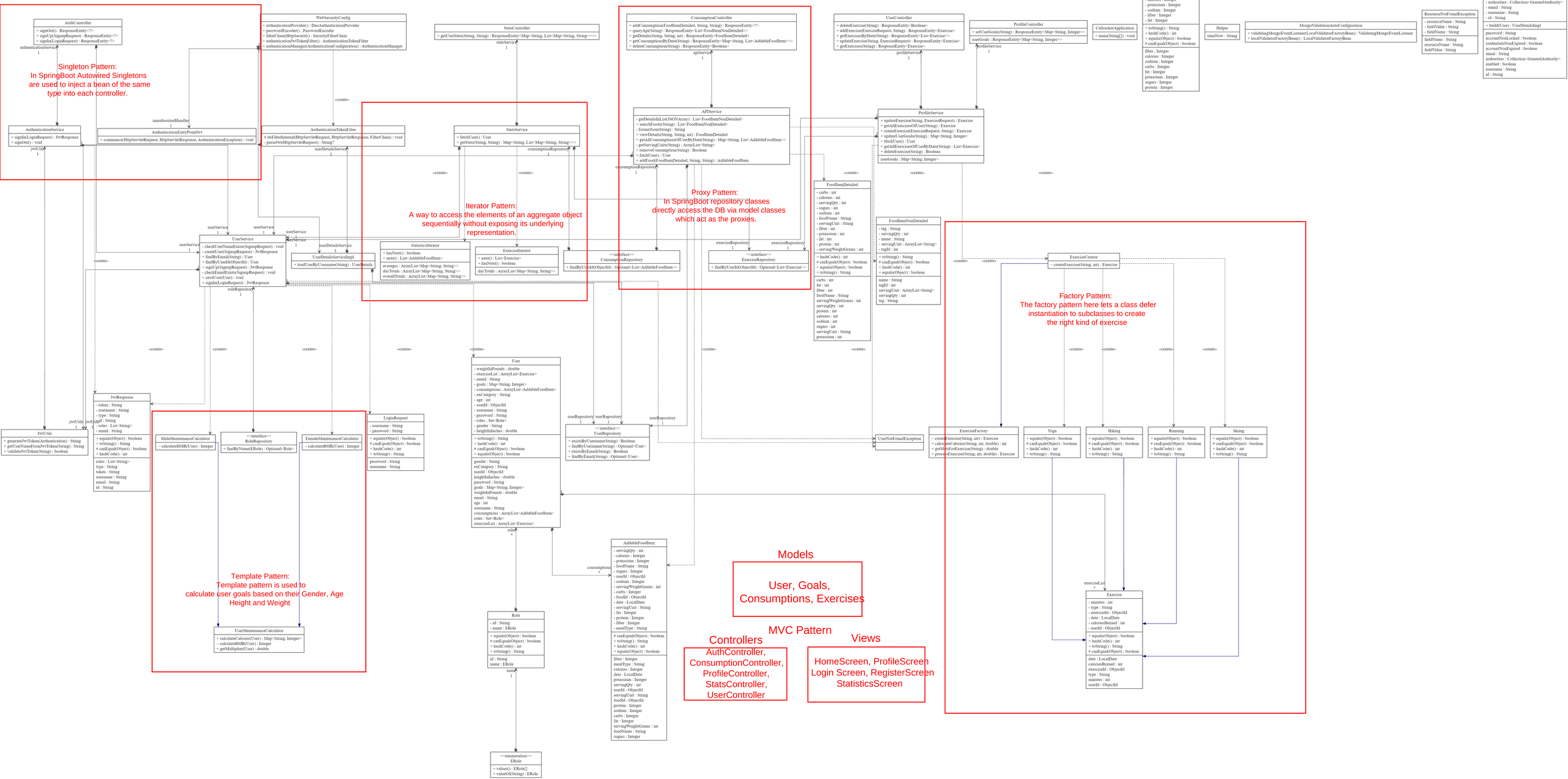
- ☐ Positive Experiences:

- Spring Boot: Despite being familiar with Java, neither of us had prior experience with Spring Boot due to its steep learning curve. However, once we dived into it, we discovered just how convenient it is. Spring Boot's modularity, excellent support for dependency injection, and inversion of control made our development process a breeze. It empowered us to build robust and scalable applications with ease.

- Planning: Planning played a crucial role in the successful implementation of this project. It cannot be overstated how valuable it was to create the class diagram and activity diagram before diving into coding. Without proper planning, we would have faced numerous challenges along the way. Thanks to the thorough planning process we undertook in Project 5, we were able to apply the appropriate design patterns and make the code highly scalable and efficient. The time invested in planning upfront paid off immensely!

☐ Negative Experiences:

- Frontend: While I wouldn't describe the experience as entirely negative, it wasn't particularly enjoyable. Given the project's design requirements, we found ourselves in need of a frontend application to showcase the various functionalities. However, since neither of us had prior experience with frontend development, we faced significant challenges while working with React. In particular, managing the application's state proved to be quite demanding, and it took us some time to fully comprehend and utilize React's state management capabilities effectively (Redux). Despite the difficulties, we persevered and eventually gained a better understanding of React's concepts and how they can be leveraged to maintain the application's state efficiently.


Singleton Pattern:
In SpringBoot Autowired Singletons
are used to inject a bean of the same
type into each controller.
Iterator Pattern:
A way to access the elements of an aggregate object
sequentially without exposing its underlying
representation.
Proxy Pattern:
In SpringBoot repository classes
directly access the DB via model classes
which act as the proxies.
Factory Pattern:
The factory pattern here lets a class defer
instantiation to subclasses to create
the right kind of exercise
Template Pattern:
Template pattern is used to
calculate user goals based on their Gender, Age
Height and Weight
Models
User, Goals,
Consumptions, Exercises
MVC Pattern
Controllers
AuthController,
ConsumptionController,
ProfileController,
StatsController,
UserController
Views
HomeScreen, ProfileScreen
Login Screen, RegisterScreen
StatisticsScreen