**BR Documentation - Batch 2**

**Team Member-1: Tallati Nikhil Bhushan (284698)**

**Team Member-2: Harini Parsha (284699)**

**Problem Statement:**

**Scenario:** Modern life often complicates efforts to maintain a healthy lifestyle, with busy schedules making it difficult to track and balance physical activity, diet, and mental health. For example, a busy professional may find it challenging to keep track of their daily exercise, maintain balanced diet, and manage stress. Without a comprehensive tool to monitor these aspects, individuals may struggle to identify patterns and make informed decisions about their health. Existing health apps often focus on isolated aspects like diet tracking or workout logging, leaving users without a holistic view of their overall well-being.

**SCOPE:**

**Functional Requirements:**

**1. User Authentication and Authorization:**

- The system must allow users to register, log in, and log out.

- User authentication should be managed through JWT tokens, ensuring secure access to the application.

**2. User Management:**

- Users should be able to update their profile information, including username, email, password, age, gender, height, weight, and activity level.

- The system should support CRUD operations for user data.

**3. Diet Planning:**

- Users should be able to view and manage their diet plans.

- The system should calculate daily calorie intake based on user preferences and activity levels.

- The Diet Planner Service should interact with the Diet DB to store and retrieve diet-related information.

**4. Fitness Tracking:**

- Users should be able to log fitness activities, including activity type, duration, calories burned, and activity date.

- The system should track daily activity summaries, including total calories burned and total duration of activities.

- Fitness activities should be stored and managed within the Fitness DB.

**5. Mental Health Assessments:**

- The system should allow users to complete PHQ-9 and GAD-7 assessments.

- User responses should be stored in the Mental Health DB.

- The system should calculate and display results based on user responses.

**6. Articles and Blog Component:**

- Users should have access to articles and blogs related to mental health.

- The frontend should display this content based on the user's preferences or assessment results.

**7. API Gateway:**

- The API Gateway must route requests to appropriate backend microservices.

- It should handle requests for authentication, user management, diet planning, fitness tracking, and mental health assessments.

**Non-Functional Requirements**

**1. Security:**

- JWT tokens must be used for secure user authentication and authorization.

- Sensitive user data must be encrypted in transit and at rest.

**2. Scalability:**

- The system should be designed to handle an increasing number of users and requests.

- Microservices architecture should allow individual services to scale independently.

**3. Performance:**

- The application should respond to user requests with minimal latency.

- The system should be capable of handling high loads without significant performance degradation.

**4. Reliability:**

- The system should ensure high availability and resilience to failures.

- Each microservice should be independently deployable and upgradable.

**5. Maintainability:**

- The codebase should be modular, allowing for easy updates and maintenance.

- Proper documentation should be provided for each microservice, including API specifications.

**6. Usability:**

- The frontend should provide a user-friendly interface with intuitive navigation.

- Forms and components should be responsive and accessible across different devices.

**7. Interoperability:**

- The system should allow for seamless integration with external services, such as third-party APIs for additional health data.

- Microservices should communicate through well-defined APIs, ensuring compatibility and ease of integration.

**8. Logging and Monitoring:**

- The system should log user interactions, service requests, and errors for audit and troubleshooting purposes.

- Monitoring tools should be implemented to track the health and performance of microservices.

**9. Data Consistency:**

- The system should ensure data consistency across different microservices and databases.

- Appropriate measures should be taken to handle data synchronization and conflict resolution.

**10. Backup and Recovery:**

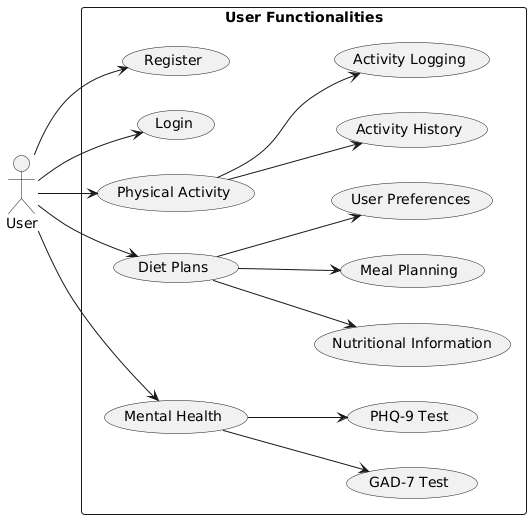
- The system should implement regular backups of all databases.

- A disaster recovery plan should be in place to restore data and services in case of a failure.

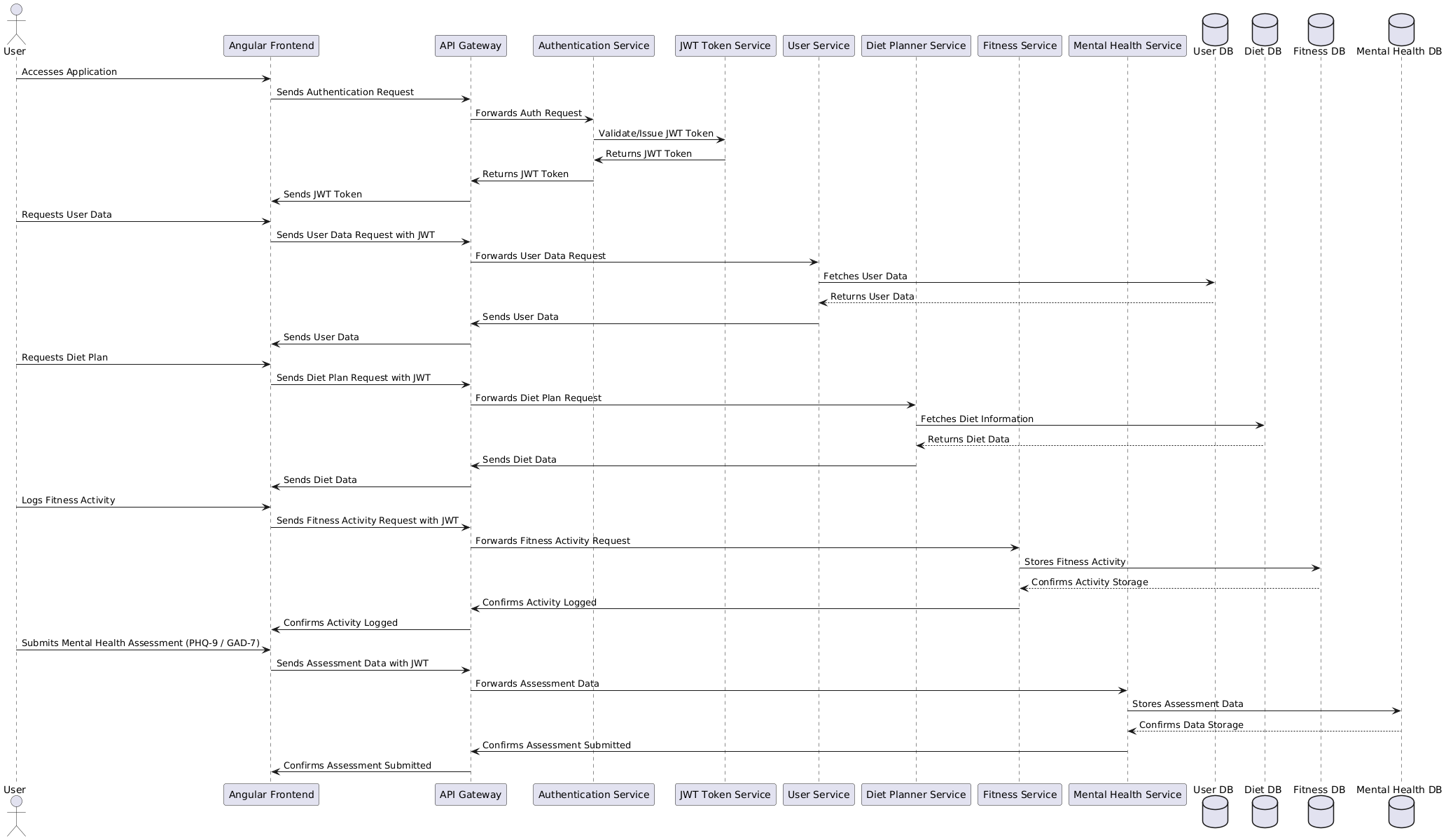
**HIGH LEVEL DESIGN:**

**Use Case Diagram:**

* User: The primary actor interacting with the system
* Register: Allows new users to create an account
* Login: Authenticates existing users
* Manage Physical Fitness: Includes setting preferences and viewing exercise plans
* Manage Diet Plans: Includes setting dietary preferences and viewing meal plans
* Monitor Mental Health: Includes answering questionnaires and viewing resources
* View Dashboard: Displays overall progress and generates reports
* Update Profile: Allows users to modify their personal information
* Log Exercise: Records completed physical activities (included in Manage Physical Fitness)
* Log Meals: Records consumed meals (included in Manage Diet Plans)
* Answer Mental Health Questionnaire: Collects mental health data (included in Monitor Mental Health)



**Sequence Diagram:**

****

The sequence diagram you provided outlines the interactions between various components of the wellness application architecture.

1. User Accesses the Application:

- The user interacts with the Angular Frontend to access the application.

2. User Authentication:

- The Angular Frontend sends an authentication request to the \*\*API Gateway\*\*.

- The API Gateway forwards this request to the Authentication Service.

- The Authentication Service requests the JWT Token Service to validate or issue a JWT token.

- The JWT Token Service validates the credentials and returns a JWT token to the Authentication Service.

- The Authentication Service sends the JWT token back to the API Gateway.

- The API Gateway forwards the JWT token to the Angular Frontend for secure communication.

3. User Data Request:

- The user requests their data via the Angular Frontend.

- The Angular Frontend sends a user data request to the API Gateway, including the JWT token for authentication.

- The API Gateway forwards the request to the User Service.

- The User Service fetches the necessary user data from the User DB.

- The User DB returns the user data to the User Service.

- The User Service sends the user data back to the API Gateway.

- The API Gateway forwards the user data to the Angular Frontend.

4. Diet Plan Request:

- The user requests a diet plan through the Angular Frontend.

- The Angular Frontend sends the diet plan request to the API Gateway, along with the JWT token.

- The API Gateway forwards the request to the Diet Planner Service.

- The Diet Planner Service retrieves the diet information from the Diet DB.

- The Diet DB returns the diet data to the Diet Planner Service.

- The Diet Planner Service sends the diet data to the API Gateway.

- The API Gateway forwards the diet data to the Angular Frontend.

5. Logging Fitness Activity:

- The user logs a fitness activity through the Angular Frontend.

- The Angular Frontend sends the fitness activity request to the API Gateway with the JWT token.

- The API Gateway forwards the request to the Fitness Service.

- The Fitness Service stores the fitness activity in the Fitness DB.

- The Fitness DB confirms the storage of the activity.

- The Fitness Service sends a confirmation of the activity being logged to the API Gateway.

- The API Gateway forwards the confirmation to the Angular Frontend.

6. Submitting Mental Health Assessments:

- The user submits a mental health assessment (either PHQ-9 or GAD-7) via the Angular Frontend.

- The Angular Frontend sends the assessment data to the API Gateway with the JWT token.

- The API Gateway forwards the data to the Mental Health Service.

- The Mental Health Service stores the assessment data in the Mental Health DB.

- The Mental Health DB confirms the data storage.

- The Mental Health Service sends a confirmation of the assessment submission to the API Gateway.

- The API Gateway forwards the confirmation to the Angular Frontend.

- The sequence diagram illustrates how the Angular Frontend interacts with the API Gateway and how the gateway routes requests to various backend services, each handling different aspects of the application (user data, diet planning, fitness tracking, and mental health assessments).

- The JWT Token Service plays a crucial role in securing the communication between the frontend and backend by issuing tokens that authenticate requests.

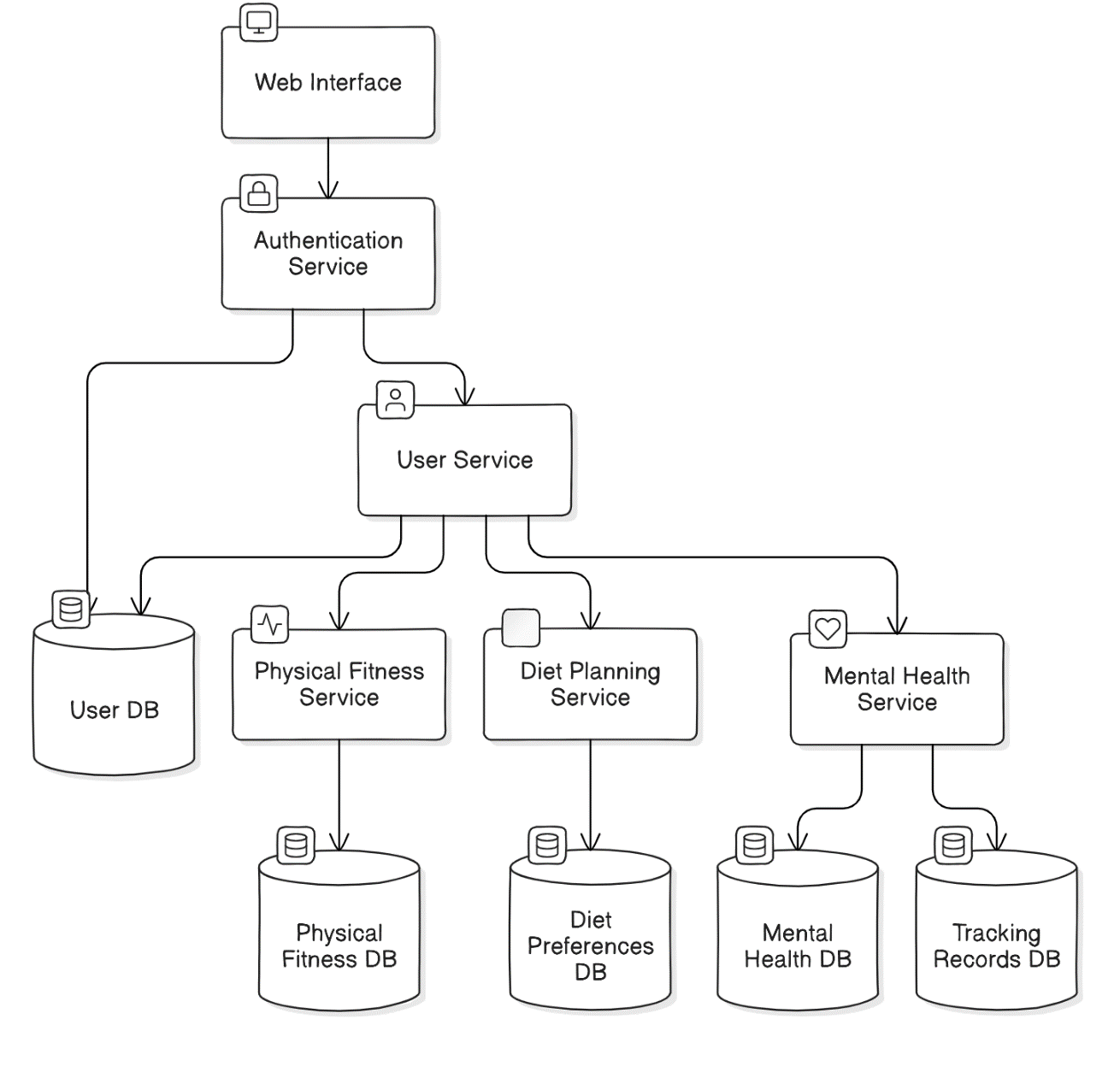
- The API Gateway acts as the central point of entry, managing all incoming requests and ensuring they are routed to the appropriate service.

- Each backend service interacts with its respective database to store or retrieve data, ensuring that the application functions efficiently and securely.

**LOW LEVEL DESIGN:**

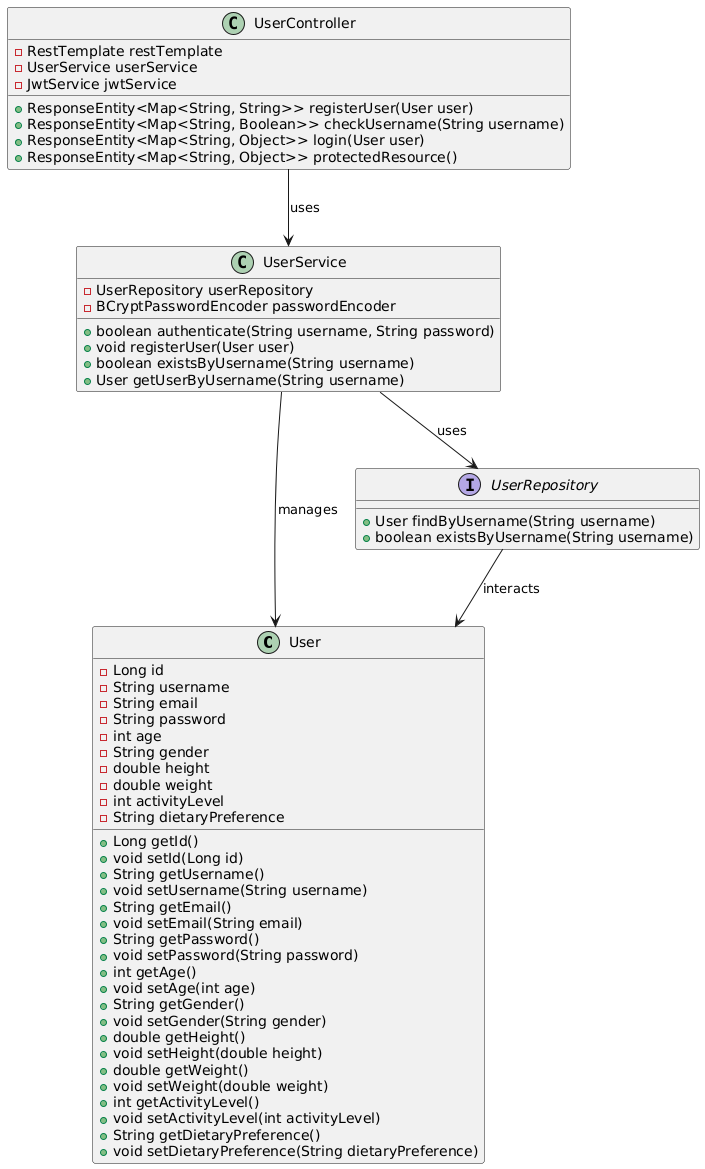
**Component Diagram:**

* Web Interface: The frontend component that users interact with
* Authentication Service: Handles user registration and login
* User Service: Manages user profiles and data
* Physical Fitness Service: Manages exercise preferences, plans, and tracking
* Diet Planning Service: Handles meal preferences, plans, and tracking
* Mental Health Service: Manages questionnaires and resources
* Databases: Separate databases for each domain (User, Physical Fitness, Diet Preferences, Mental Health)



**Class Diagrams:**

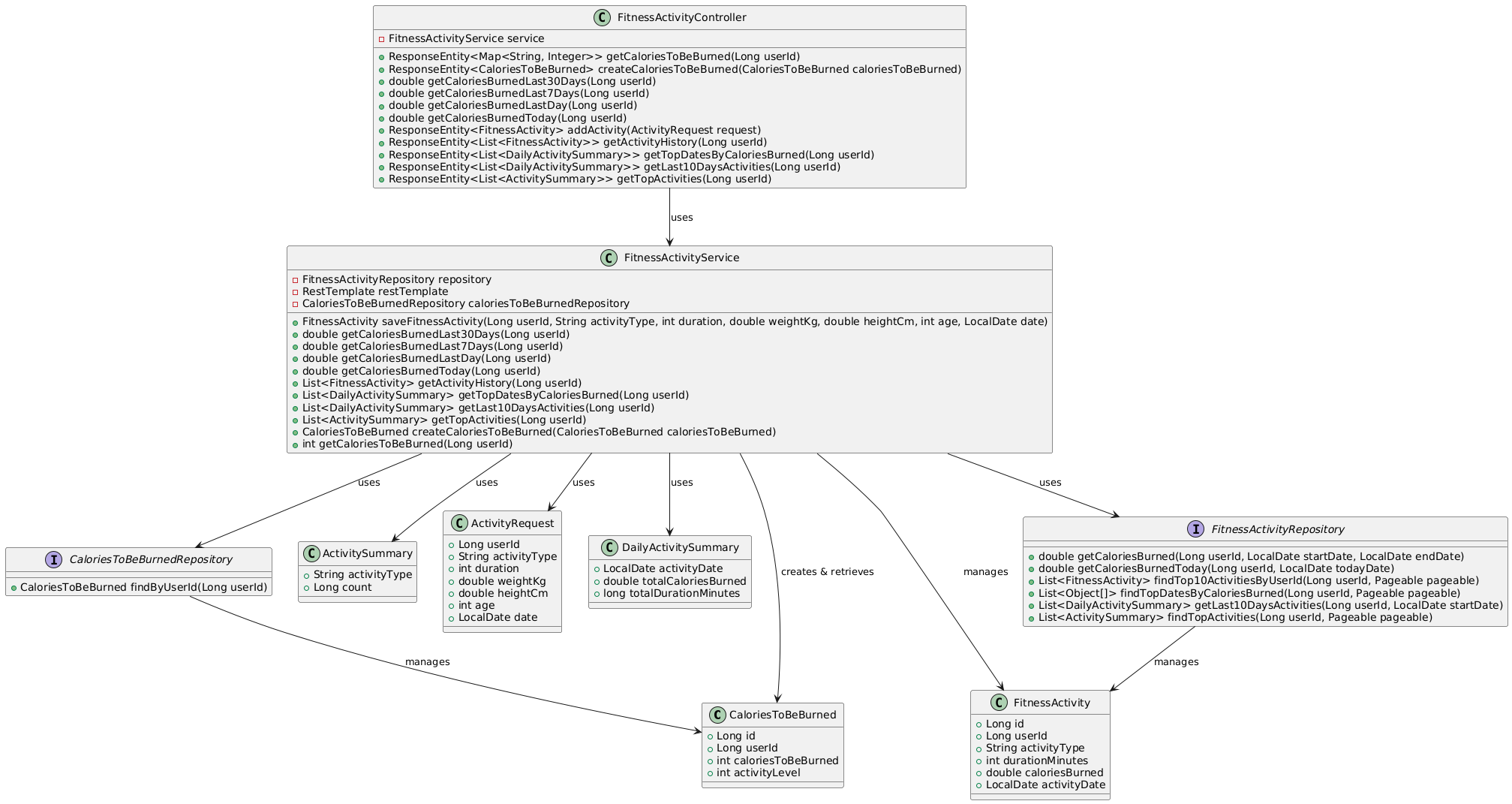
1. **User Service Class Diagram:**

****

The system is designed with clear interaction flows between components. The UserController interacts with the UserService to perform user operations, such as registration and authentication. The UserService in turn relies on the UserRepository to access and manipulate user data stored in the database. The User entity is at the core of these interactions, being created, managed, and retrieved as needed.

This design ensures a well-structured and maintainable system where each component has a clear responsibility. The encapsulation of data in the User entity, the separation of business logic in the UserService, and the clear API endpoints provided by the UserController contribute to a robust user management solution that is secure, scalable, and easy to extend.

1. **Physical Fitness Service Class Diagram:**

****

The Fitness Activity Management System operates through a well-defined interaction between its various components, including repositories, services, and controllers. Each component has a specific role, contributing to the overall functionality and efficiency of the system.

The CaloriesToBeBurnedRepository and FitnessActivityRepository are key interfaces responsible for managing and interacting with the underlying data related to calories and fitness activities. The CaloriesToBeBurnedRepository manages the CaloriesToBeBurned entity, handling operations like retrieving the calories that need to be burned by a specific user based on their unique `userId`. Similarly, the FitnessActivityRepository is responsible for managing FitnessActivity entities, providing methods to calculate calories burned over various time periods, retrieve fitness activity histories, and identify top activities and dates based on calories burned.

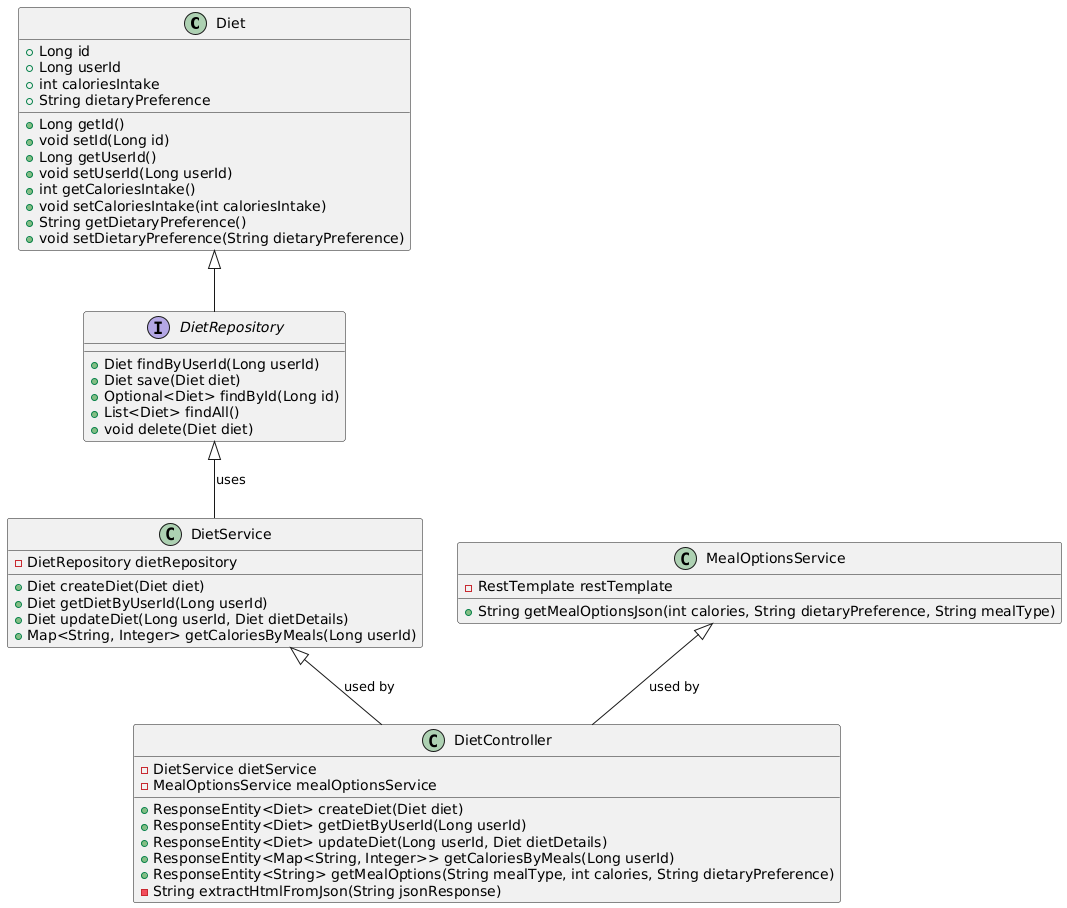
The FitnessActivityService acts as the primary business logic layer, orchestrating the interactions between repositories and other components. This service interacts with the FitnessActivityRepository to manage fitness activities and with the CaloriesToBeBurnedRepository to manage and retrieve information about calories to be burned by users. The service also utilizes a RestTemplate for potential external service calls, enabling it to perform complex operations like calculating calories burned, saving fitness activities, and aggregating fitness data for various reports. The service methods include creating and retrieving calorie data, managing fitness activities, and providing summary reports on user activities.

The FitnessActivityController serves as the entry point for handling HTTP requests related to fitness activities and calories. It interacts with the FitnessActivityService to execute the necessary business logic and return appropriate responses to the client. The controller facilitates user interactions by providing endpoints to add new fitness activities, retrieve historical activity data, and generate reports on calories burned and top fitness activities. Through its methods, the controller enables users to interact with the system, whether they need to track their daily activities, monitor their calorie burn over time, or analyze their fitness progress.

The interaction flow begins with the FitnessActivityController, which receives requests from the user interface or other services. The controller delegates these requests to the FitnessActivityService, which then interacts with the necessary repositories to retrieve or store data. The service manages the data and performs the required calculations or aggregations before passing the results back to the controller. Finally, the controller returns the processed data to the client, ensuring that the user receives accurate and timely information about their fitness activities and calorie management.

This structured interaction between components ensures that the system is modular, scalable, and capable of handling complex fitness and calorie tracking requirements efficiently.

1. **Diet Planning Service Class Diagram:**

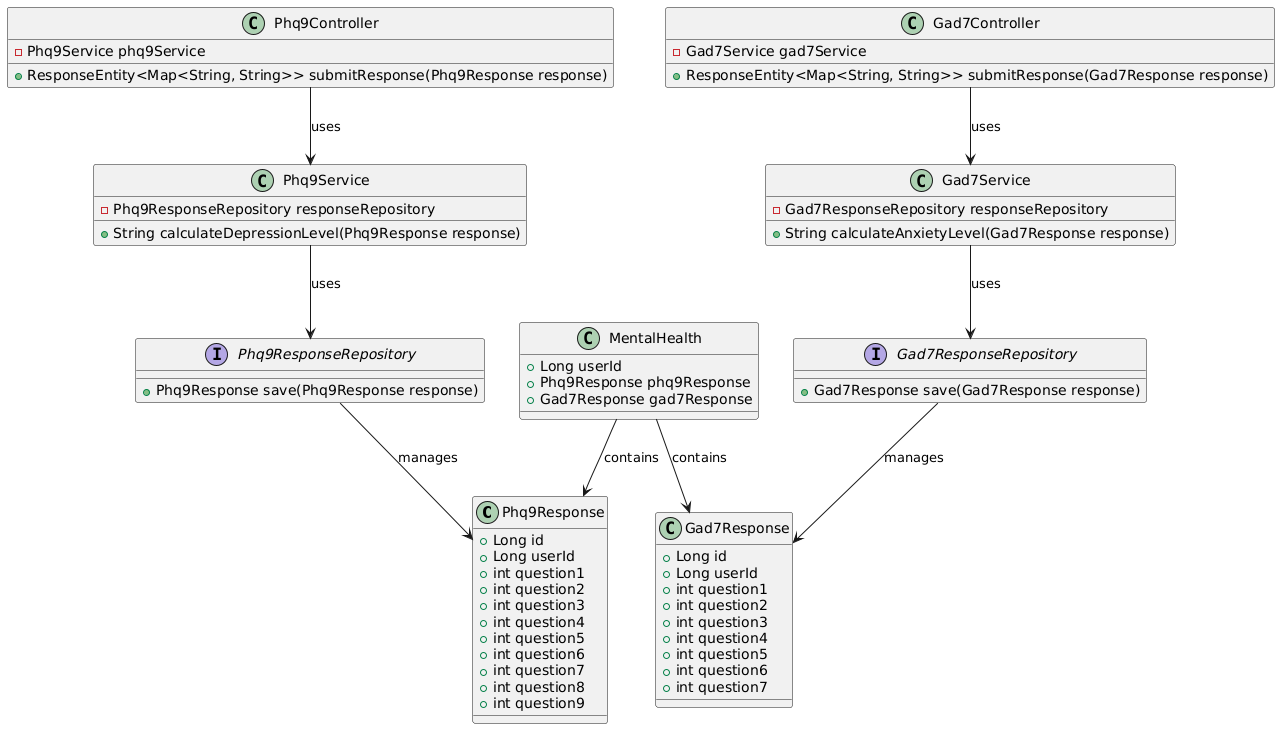
****

The interaction within the diet management system follows a structured flow among various components, each playing a specific role in managing and retrieving diet-related information. The DietRepository serves as the primary interface for interacting with the diet data stored in the database. It provides methods to find, save, update, and delete diet records based on user information. The DietService acts as a mediator between the repository and the business logic, utilizing the repository to perform operations such as creating a new diet record, retrieving diet information by user ID, and updating existing diet details. It also includes functionality to calculate calories based on meal information.

The MealOptionsService is another essential component in the system, responsible for interacting with external services through a RestTemplate to fetch meal options based on dietary preferences and calorie requirements. This service provides JSON responses containing meal data, which can be used to suggest appropriate meals to users.

The DietController is the component that interfaces with the client or user interface, handling HTTP requests related to diet management. It makes use of the DietService to process requests such as creating, retrieving, or updating diet records. The controller also interacts with the MealOptionsService to retrieve meal suggestions, which are then processed and returned to the client. Through these interactions, the diet management system ensures that users receive accurate and personalized dietary recommendations and can effectively track their dietary intake.

1. **Mental Health Service Class Diagram:**

****

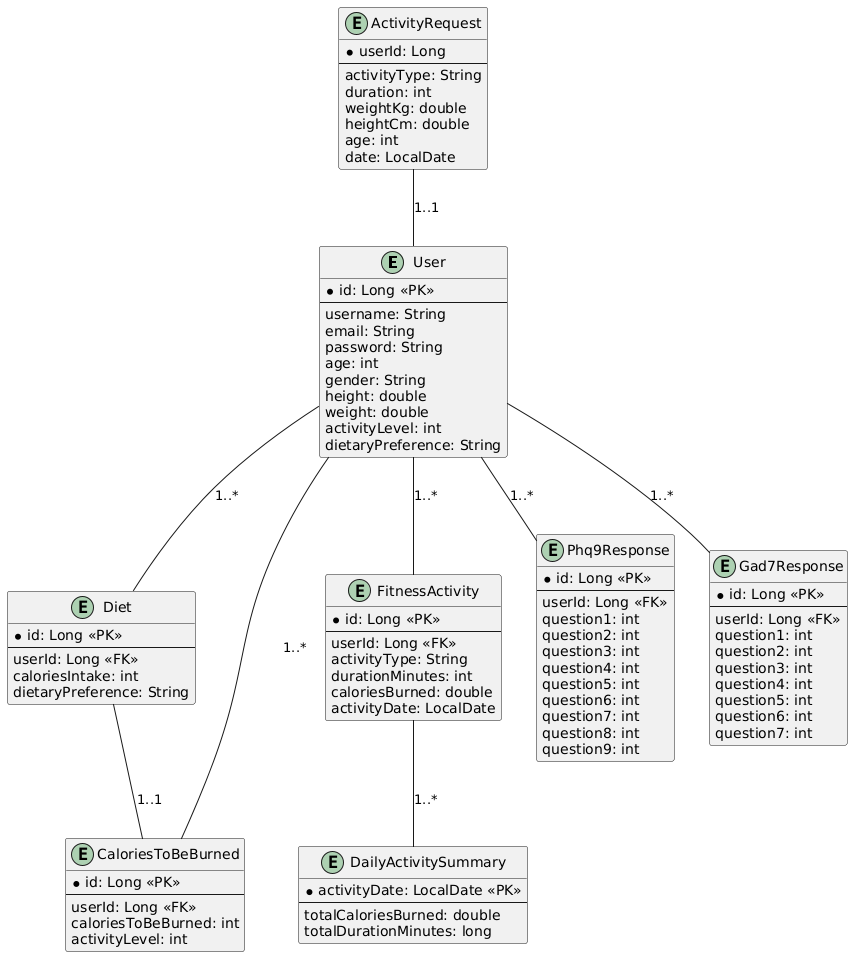
The interaction within the mental health assessment system is structured around managing and processing responses to the PHQ-9 and GAD-7 assessments. The MentalHealth class acts as an aggregator that contains both Phq9Response and Gad7Response objects, representing a user's responses to these assessments.

The Phq9ResponseRepository and Gad7ResponseRepository interfaces are responsible for handling the persistence of PHQ-9 and GAD-7 responses, respectively. They provide the necessary methods to save these responses to the database.

The Phq9Service and Gad7Service classes are responsible for the business logic related to these assessments. The Phq9Service uses the Phq9ResponseRepository to save PHQ-9 responses and includes a method to calculate the depression level based on the response data. Similarly, the Gad7Service interacts with the Gad7ResponseRepository to save GAD-7 responses and calculate the anxiety level.

Finally, the Phq9Controller and Gad7Controller serve as the entry points for handling HTTP requests related to the PHQ-9 and GAD-7 assessments. They use the respective services to process and submit responses, returning the calculated levels of depression or anxiety to the user. Through these interactions, the system ensures that user assessments are accurately recorded and analyzed, providing valuable insights into the user's mental health.

**Database Diagram:**

****

The Wellness Application employs a comprehensive and structured data model designed to manage and interrelate various aspects of user health and wellness data effectively. At the core of this model is the \*\*User\*\* entity, which serves as the primary entity containing essential user information such as `username`, `email`, `password`, `age`, `gender`, `height`, `weight`, `activityLevel`, and `dietaryPreference`. Each user is uniquely identified by an `id`, which serves as the primary key (PK) and is referenced by other entities through foreign keys (FK) to establish relationships.

1. User:

- Attributes\*\*: `id`, `username`, `email`, `password`, `age`, `gender`, `height`, `weight`, `activityLevel`, `dietaryPreference`.

- Relationships: A single user can have multiple associated records in other entities, reflecting the user's diverse health and wellness activities.

2. Diet:

- Attributes: `id`, `userId`, `caloriesIntake`, `dietaryPreference`.

- Relationships: Each diet plan is linked to one user (`userId` as FK), allowing a user to have multiple diet plans. Additionally, each diet plan is associated with exactly one CaloriesToBeBurned entry, establishing a one-to-one relationship.

3. CaloriesToBeBurned (CTB):

- Attributes: `id`, `userId`, `caloriesToBeBurned`, `activityLevel`.

- Relationships: Each CTB record is tied to a specific user and diet plan, ensuring that calorie-burning targets are personalized based on the user’s activity level and dietary preferences.

4. FitnessActivity:

- Attributes: `id`, `userId`, `activityType`, `durationMinutes`, `caloriesBurned`, `activityDate`.

- Relationships: Each fitness activity is associated with one user. Furthermore, a fitness activity can generate multiple DailyActivitySummary records, summarizing daily fitness metrics.

5. DailyActivitySummary:

- Attributes: `activityDate`, `totalCaloriesBurned`, `totalDurationMinutes`.

- Relationships: Each summary record is linked to a specific fitness activity, aggregating data to provide users with daily insights into their fitness performance.

6. ActivityRequest:

- Attributes: `userId`, `activityType`, `duration`, `weightKg`, `heightCm`, `age`, `date`.

- Relationships: Each activity request is directly associated with one user, capturing detailed information about the user’s fitness activities for accurate tracking and analysis.

7. Phq9Response:

- Attributes: `id`, `userId`, `question1` to `question9`.

- Relationships: Each PHQ-9 response is linked to a user, enabling the system to monitor and assess the user’s depression levels over time.

8. Gad7Response:

- Attributes: `id`, `userId`, `question1` to `question7`.

- Relationships: Each GAD-7 response is associated with a user, facilitating the tracking and evaluation of the user’s anxiety levels.

The data model ensures data integrity and consistency through the use of primary and foreign keys, which maintain the relationships between users and their various health-related records. For instance, every diet plan, fitness activity, and mental health assessment is directly associated with a user, ensuring that all data is correctly attributed and easily retrievable. The model supports the application's functional requirements by enabling efficient CRUD (Create, Read, Update, Delete) operations across different domains such as diet planning, fitness tracking, and mental health assessments.

- User Management: The User entity manages essential user information, allowing users to register, update their profiles, and securely authenticate through associated services.

- Diet Planning: The Diet and CaloriesToBeBurned entities work together to provide personalized diet plans and calorie targets based on user preferences and activity levels.

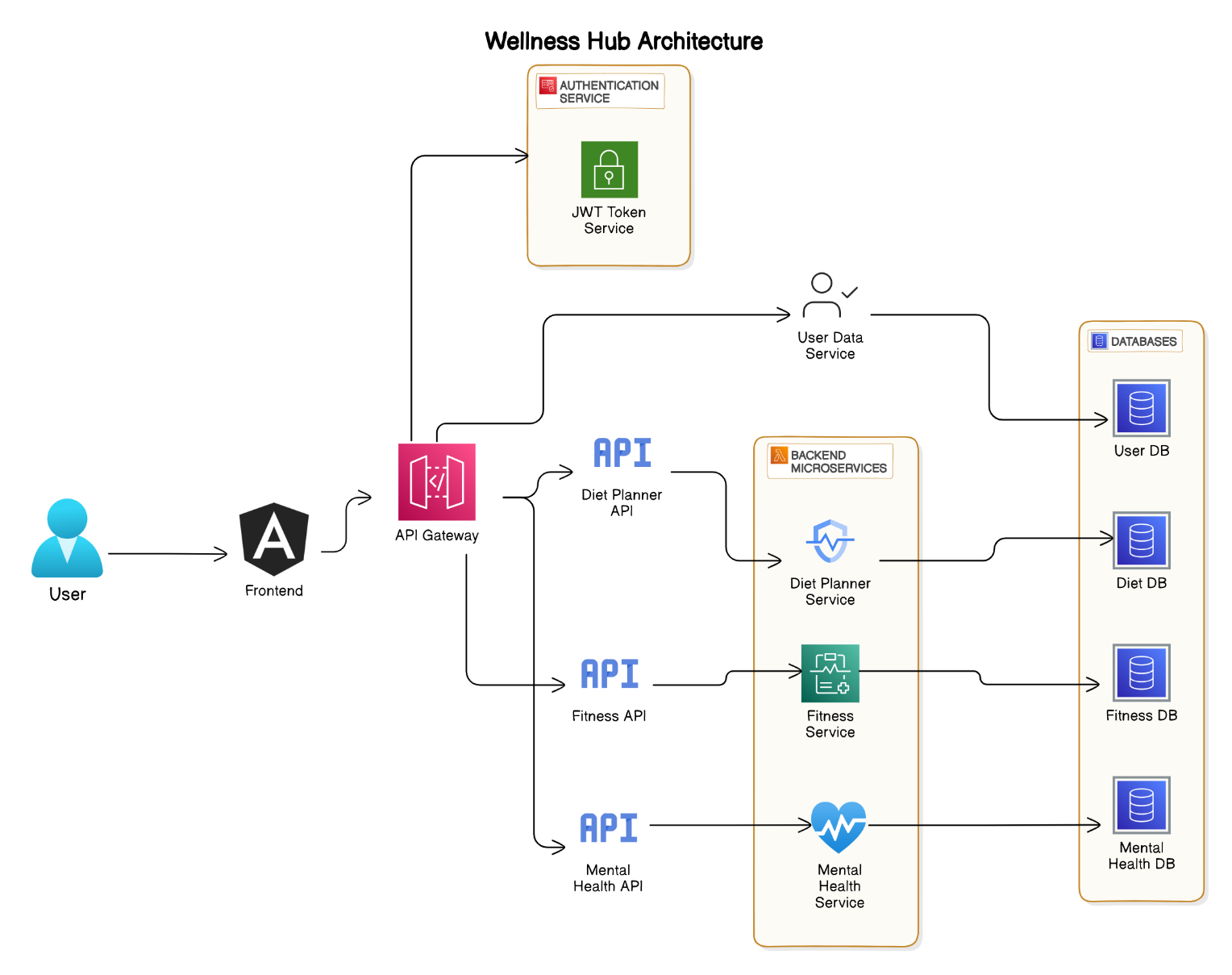
- Fitness Tracking: The FitnessActivity and DailyActivitySummary entities facilitate detailed logging and summarization of fitness activities, helping users monitor their physical progress.

- Mental Health Assessments: The Phq9Response and Gad7Response entities store user responses to standardized mental health assessments, enabling the application to track and analyze users’ mental well-being over time.

The structured relationships and clear separation of concerns among entities ensure that the data model is scalable and maintainable. As the application grows, new features or additional data points can be integrated without disrupting existing functionalities. The use of distinct entities for different aspects of wellness data allows for modular development and easier updates.

The data model for the Wellness Application is meticulously designed to support a wide range of functionalities related to user health and wellness. By establishing clear relationships between users and their diet, fitness, and mental health data, the model ensures seamless data management, integrity, and scalability. This robust structure underpins the application's ability to deliver personalized and comprehensive wellness services to its users.

**Architecture:**

****

In the Wellness Application architecture, detailing how various components interact to deliver functionalities like user management, diet planning, fitness tracking, and mental health assessments. The architecture employs a microservices-based approach, ensuring scalability, flexibility, and secure communication between the frontend and backend services.

The key components in the architecture include the Angular Frontend, which serves as the user interface, allowing users to interact with the application. The API Gateway acts as the central point for routing requests from the frontend to the appropriate backend microservices. These microservices include the Authentication Service, responsible for user authentication, and the JWT Token Service, which generates and validates JWT tokens to secure communication. The User Service handles user data, interacting with the User Database, while the Diet Planner Service, Fitness Service, and Mental Health Service manage diet, fitness, and mental health-related functionalities, respectively, each interacting with their corresponding databases.

The sequence begins when a user accesses the application through the Angular Frontend. The user initiates an authentication request, which the frontend forwards to the API Gateway. The API Gateway then directs this request to the Authentication Service. The Authentication Service communicates with the JWT Token Service to validate the user credentials and issue a JWT token. Once the token is generated, it is returned through the Authentication Service to the API Gateway, which then sends it back to the Angular Frontend for future secure requests.

Following authentication, the user may request data such as their profile, diet plan, fitness activities, or mental health assessments. For instance, when the user requests their data, the Angular Frontend sends the request to the API Gateway, including the JWT token for authentication. The API Gateway forwards this request to the User Service, which retrieves the necessary data from the User Database. The data is then returned to the User Service, which sends it back through the API Gateway to the Angular Frontend, where it is presented to the user.

Similarly, when the user requests a diet plan, the process involves the Angular Frontend sending the request, including the JWT token, to the API Gateway. The API Gateway forwards this request to the Diet Planner Service, which fetches the required diet information from the Diet Database. The retrieved data is sent back through the same pathway, ensuring that the user receives their personalized diet plan securely.

Throughout this architecture, the use of JWT tokens ensures that all interactions between the frontend and backend are authenticated, enhancing the security of user data and other sensitive information. Each microservice is responsible for a specific domain, enabling clear separation of concerns, easier maintenance, and the ability to scale individual services as needed. This architecture effectively supports the wellness application's needs, offering users a seamless and secure experience across multiple health-related functionalities.

**WireFrames:**

**A computer screen shot of a computer

Description automatically generated**

**Database Schemas:**

1. **app\_user**

A screenshot of a computer

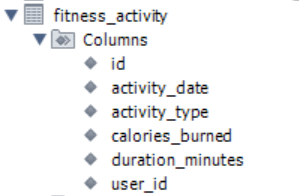
Description automatically generated

1. **calories\_to\_burned**

A screenshot of a computer

Description automatically generated

1. **fitness\_activity**



1. **diet:**

A screenshot of a computer

Description automatically generated

1. **phq9\_responses**

A screenshot of a computer

Description automatically generated

1. **gad7\_responses:**

