SOFTWARE ENGINEERING

Assignment 3 Report

SUBMITTED BY:

Ahmad Waleed Akhtar	BSCE22003
Hadia Ahmad	BSCE22017

Structural Model

Structural models describe the static structure of the system. The diagrams in structural model focus on how different parts of the application are organized, how they are related to one another, and how they interact through defined interfaces. These diagrams do not deal with time-based behavior; instead, they give a blueprint of the system's architecture. In the case of our fitness app, structural modeling helps us understand which components exist (like the user manager, activity tracker, and notification system), how they are connected, and how responsibilities are distributed among them.

The following are the diagrams/representations included in the structural model of the Fitness App:

Diagram	Description
Class & Interface Diagram	Shows system classes, their interfaces, and how subsystems implement them.
Design Pattern Diagram	Illustrates the use of the Facade pattern to encapsulate and simplify access to multiple subsystems.
Association Diagram	Depicts how various subsystems (auth, tracker, calendar, etc.) are related and communicate.

1. Class and Interface Diagram

This diagram shows the major classes in our system and the interfaces they implement. It helps in understanding how different components interact using well-defined contracts.

- Interfaces like IUserManagement, IActivityDataProvider, ISocialFeed, etc.
- Concrete classes like WearableDevice, FitnessAppSystem, CalendarService, and others.
- Each interface defines required methods, and each class implements one interface.
- The User interacts only with the FitnessAppSystem, which coordinates with all other classes.

Advantages:

- Supports abstraction and loose coupling.
- Makes testing and maintenance easier.

Diagram attached separately as class_interface_diagram.svg.

2. Design Pattern

The design pattern chosen for the fitness app is the Façade pattern. The reason being that the fitness app has a number of subsystems (User management, Activity Data Providers, Progress Tracking, etc.), if the UI had to invoke each of these directly it would end up being a very disoriented and jumbled up type of code for the system which will be very difficult to maintain.

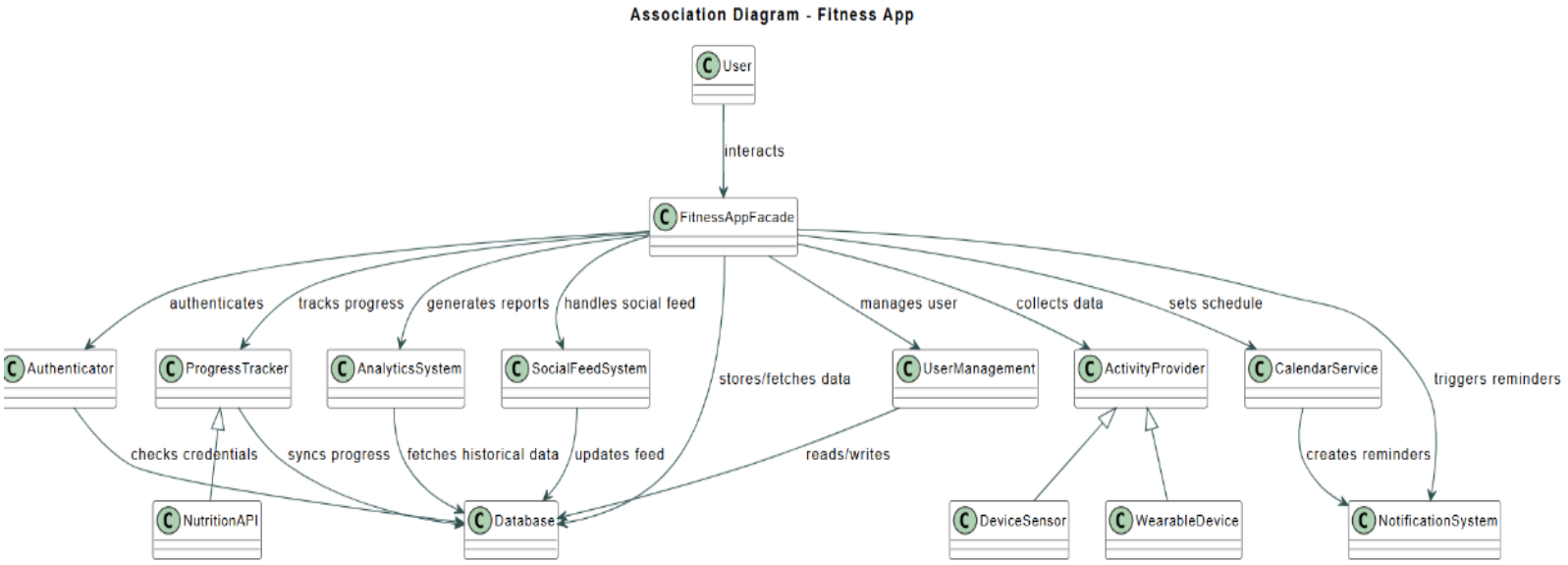
- The façade pattern centralizes access; a single class called FitnessAppFacade that works as a central point of access to all the subsystems. The user interface only needs to interact with this one class.
- This encapsulates complexity as this makes the app easier to use and prevents duplicate logic across different parts of the code.
- It also improves reusability as operations like trackActivity(), scheduleWorkout() can be shared across the Mobile app, Web app, admin dashboard, etc.
- This also makes the code maintenance more efficient as you will not have to change the code in all the interfaces, you will just have to change the base function.
- The façade pattern also allows you to scale the system in an organized and efficient manner. If you want to add a new feature you just extend the façade and connect it to another subsystem instead of updating every interface/subsystem individually.

The design pattern diagram is attached separately as “design pattern.svg”.

3. Association Diagram

It is used to show the connections and interactions between system components.

- The FitnessAppFacade connects to subsystems such as Authenticator, AnalyticsSystem, NotificationSystem, etc.
- Systems like CalendarService and NotificationSystem work together to send reminders. (*The diagram is also attached separately as “association diagram.svg”.)*



Dynamic Model

Dynamic models describe the behavior of the system over time. They focus on how the system reacts to user input, internal events, and external interactions. These diagrams show the flow of control and data, the order of operations, and how the system changes state based on different scenarios. For our fitness app, dynamic modeling helps visualize user sessions, workflows, background processes, and state transitions, making it easier to design features that work smoothly and handle both normal and error cases properly.

The following diagrams are included in the dynamic model of the fitness app:

Diagram	Description
Sequence Diagram	Shows step-by-step interaction between user, facade, and all subsystems across all use cases.
Sequence Diagrams (per use case)	Each sequence shows how objects interact for a specific user story (e.g., set goals, track activity).
Activity Diagram	Captures the complete workflow of the app: from login to all feature interactions, including background processes.
State Diagram	Models all possible system states (e.g., Authenticated, Unauthenticated) and transitions triggered by actions or events.
State Diagrams (per use case)	Each use case has its own lifecycle model (e.g., Share Progress → Validating → Posting → Confirming).

1. Sequence Diagram

The Sequence Diagram for the fitness app explains how different parts of the system interact with each other when the user is using the app. It shows the flow of actions from the moment the user opens the app to when they use different features. This diagram focuses on how the user, the facade layer, and various subsystems communicate during a session.

The flow begins with:

- **Login or Registration:**
  The user either logs in (if they already have an account) or registers (if they are new). The system validates the credentials or registration input and gives access accordingly.
- **Main Session Loop:**
  After login, the user can use any feature provided by the app. These include:
  - Setting fitness goals
  - Generating a workout plan
  - Tracking activity
  - Viewing nutrition tips

    - Scheduling workouts
    - Viewing progress reports and analytics

  The diagram uses a loop to show that the user can use these features multiple times in one session.

- **Session & Input Validation:**
  Before any action is performed, the system:
    - Checks if the session is still active (to ensure the user is logged in)
    - Validates the input data (e.g., makes sure a number isn't entered in a text-only field)
- **Alternative Paths (alt blocks):**
  If the session is invalid or the input is incorrect:
    - The system shows an error message
    - The action is not completed

  These alternate flows are shown using alt blocks to explain how the system handles errors.

The sequence diagram is helpful as it gives a complete view of how the system behaves during real use. It does not just show the ideal situation where everything works correctly; it also shows how the system reacts to common problems. This helps developers and testers make sure the app is easy to use, reliable, and secure.

P.S: *Individual sequence diagrams (per use case) and the whole system sequence diagram attached separately in the sequence diagrams folder.*

2. Activity Diagram

The Activity Diagram gives a clear picture of how the user interacts with the app from start to finish. It represents the step-by-step workflow a user follows when using the app, as well as what the system is doing in the background at the same time.

At the beginning, the user is presented with the option to log in or register. Once successfully authenticated, the app takes the user to the main menu, where they can choose from different features such as setting fitness goals, generating a personalized workout plan, tracking activity, checking nutrition information, or viewing reports.

- **Feature Selection Loop**:
  The diagram shows a loop that allows the user to select and perform multiple actions without logging in again each time. After completing one action, the user is brought back to the main menu to choose another feature. This reflects how a real session works.
- **Parallel Background Tasks**:
  While the user is actively using the app, the system is also doing work in the background. These background processes include:
    - Sending notifications and reminders based on the user's calendar and workout schedule.

- Running analytics to track the user's progress and generate reports.
  These processes run in parallel with the user's actions, and the diagram shows this using a "fork" structure to represent parallel workflows.

The activity diagram shows both what the user experiences and what the system does in the background. It provides a complete view of how the app handles multiple tasks at once and keeps the experience smooth for the user. It also helps designers and developers understand how different parts of the app work together during a typical user session.

P.S: *Individual activity diagrams (per use case) and the whole system sequence diagram attached separately in the activity diagrams folder.*

3. State Diagram

The State Diagram models the overall lifecycle of the fitness app. It focuses on how the app behaves in different states, such as when the user is logged out, logged in, or actively using the app. It also shows how the system's behavior changes depending on the user's actions or internal system events.

The diagram begins in the Unauthenticated state, which represents the app when a user has not yet logged in. From here, the user can either go to the login screen or choose to register as a new user. Once the login or registration is successful, the app transitions to the Authenticated state.

- **Main App State: Authenticated**
  In this state, the app is fully active and the user has access to all features. Within the Authenticated state, the diagram is divided into two parallel regions:
  - **User Actions Region**:
    This part shows states like:
    - Idle (waiting for user action)
    - SetGoals, GeneratePlan, TrackActivity, ViewReports, etc.
      Each feature has its own state, and after completing the action, the system returns to the Idle state. If input validation fails, it loops back to allow retrying.
  - **Background Tasks Region**:
    This section runs in parallel and includes:
    - MonitoringCalendar for checking scheduled workouts
    - TriggeringNotifications to send reminders
    - AnalyticsProcessing to analyze user data and generate progress insights

    These tasks continue to operate regardless of the user's direct actions.
- **Session Handling**
  If the user remains inactive for too long or logs out, the system calls sessionExpired() and transitions back to the Unauthenticated state.

The state diagram is very useful as it clearly shows how the app changes behavior depending on the current state and how multiple processes run simultaneously. It helps developers manage transitions and ensures that the app always responds correctly whether the user is interacting directly or the system is working in the background. It also supports better error handling and consistent user experience across different features.

P.S: *Individual activity diagrams (per use case) and the whole system sequence diagram attached separately in the activity diagrams folder.*

Conclusion

By using both structural and dynamic UML diagrams, we were able to clearly represent how our fitness app is designed and how it works. The structural diagrams helped us show the blueprint of the system; what components are involved, how they are connected, and what each component is responsible for. These include diagrams like the Class and Interface Diagram, Design Pattern and Association Diagram. They gave us a clear idea of how the app is organized internally and how all the subsystems (like user management, workout planner, tracker, social feed, and calendar) work together through well-defined connections.

On the other hand, the dynamic diagrams focused on how the app behaves when it is running. These included the Sequence Diagrams, Activity Diagrams, and State Diagrams. These diagrams show how the user moves through the app, how data is processed, how errors are handled, and what happens behind the scenes. For example, the activity diagram showed the full user flow and how background processes like reminders and analytics run at the same time. The sequence diagrams explained the step-by-step process of different user actions like setting goals or tracking activity. The state diagrams helped us understand how the app switches between different modes, like being logged out, logged in, or idle.

Together, these two types of models give us a complete and easy-to-understand view of both the structure and behavior of the app. This makes it much easier for developers, designers, and testers to work on the app, fix issues, add new features, and ensure everything works smoothly. It also ensures that the app can grow in the future without becoming messy or confusing.