



Intaliq

SE322: Software Design and Architecture

Section: 1340

Instructor

Dr. Suad Ramouni

Prepared by:

YARA FARIS FAIHAN ALBUGAMI

ROSE SAEED RAKAN AL RAKAN

SHOUG FAWAZ ABDULLAH ALOMRAN

FAJR ALGHOFAILI

MOZOUN ALMUQRIN

MUNIRA YAZEED ABDULAZIZ ALFARIS

Table Of Contents

1. Introduction.....	3
1.1 Date of Issue.....	3
1.2 Context.....	3
1.3 Scope.....	3
1.4 Authorship.....	4
1.5 Change history.....	4
1.6 Summary.....	4
2. Software Architecture.....	5
2.1 Overview.....	5
2.2 Stakeholders.....	5
2.3 System Design Concerns.....	5
2.4 Client-Server Architectural Viewpoint.....	6
2.4.1 Client-Server Design View.....	6
Deployment Diagram.....	8
2.4.2 Justification.....	8
2.5 Mobile Application Architectural Viewpoint.....	9
2.5.1 MVC Design View.....	9
MVC Design Diagram.....	9
2.5.2 Justification.....	9
2.6 Backend Architectural Viewpoint.....	10
2.6.1 Layered Architecture Design View.....	10
Layered Architecture Design Diagram.....	10
2.6.2 Justification.....	10
3. Detailed Design.....	11
3.1 Overview.....	11
3.2 Factory Method (Creational).....	11
3.2.1 How It Is Applied.....	11
3.2.2 Justification.....	12
3.3 Observer (Behavioral).....	12
3.3.1 How It Is Applied.....	12
3.3.2 Justification.....	13
3.4 Adapter (Structural).....	14
3.4.1 How It Is Applied.....	14
3.4.2 Justification.....	15
4. Reference.....	15
5. Glossary.....	16

1. Introduction

1.1 Date of Issue

The Software Design Document was done on October 25, 2026

1.2 Context

Intaliq is a fitness app for mobile phones that facilitates engagement in outdoor sports like running, cycling, and hiking in the Kingdom of Saudi Arabia. This platform caters to the objectives of the Vision 2030 program by helping achieve more exercise participation weekly and enhanced quality of life. Through this platform, one is able to register activities, set goals, connect with partners, attend coach-led sessions, and receive fitness tips through an AI chatbox.

This document builds on earlier work on the project:

- SE311's Software Requirements Specification (SRS), that provided functional and non-functional requirements.
- The high fidelity prototype from SE365 that was made using JavaScript, Vite, Supabase, and Vercel.
- Software Architecture Design (SAD) from Deliverable 1 that used 4 + 1 View Model to model the system.
- Architecture Documentation from Deliverable 2 that had four quality attributes (Security, Availability, Performance, and Usability) and Client-Server architectural style.

The SDD moves from these architecture-level decisions down to the design level by applying design patterns to Intaliq's components.

1.3 Scope

Intaliq is a fitness software that was created to promote outdoor exercises in Saudi Arabia. Users can log activities such as running, cycling, and hiking, set personal fitness

goals, and take part in fitness coach-led group sessions. Coaches can also benefit by promoting their sessions to reach a broader audience. Through social connections and organized activities, the system seeks to increase the accessibility, motivation, and social engagement of outdoor fitness.

1.4 Authorship				
Prepared by	Project Name	Course	Institution	Document Type
Yara Albugami Rose Al Rakan Shoug Alomran Fajr Alghofaili Mozoun Almuqrin Munira Alfaris	Intaliq	SE322: Software Design And Architecture	Prince Sultan University	SDD: Software Design Description

1.5 Change history			
Version	Date	Deliverable	Description of Changes
1.0	30/09/2026	Deliverable 1: SAD	Initial architecture using the 4+1 View Model, including use case, class, sequence, activity, component, and deployment diagrams.
2.0	10/10/2026	Deliverable 2: Architecture Documentation	Added four quality attributes with scenarios and selected the Client-Server architectural style.
3.0	25/10/2026	Deliverable 3: SDD	Added design patterns, their justification, and how they apply to Intaliq.

1.6 Summary

This SDD outlines the implementation of Intaliq's Client Server Architecture by means of design patterns. Introduction is provided in Section 1, where background, scope, and the authorship of the document are introduced. The subsequent sections outline the design pattern, where it is applied in Intaliq and the rationale behind the selection of that particular pattern. These patterns assist in achieving the objectives of Intaliq in terms of security, availability, performance, and usability.

2. Software Architecture

2.1 Overview

Intaliq follows a Client-Server architecture, as selected in Deliverable 2. The system has four main parts:

- **Intaliq Mobile Application (Client):** The client through which the user and coach interact with logging of activities, goal setting, scheduling sessions, finding partners, and using the AI Chatbox.
- **Intaliq Backend (Application Server):** This is the server that will accept requests by the mobile application using HTTPS protocol. Authentication, user, activities and goals, sessions, partner finding, notification and AI chatbox will be handled by the server.
- **Intaliq Database:** This holds the user, coach, activities, goals, sessions, and partner information.

The design patterns in this document are applied mainly inside the Intaliq Backend, where the business logic and external integrations are handled.

2.2 Stakeholders	
Stakeholder	Description
End Users	People who do outdoor running, hiking, or cycling, from beginners to experienced
Coaches	Intaliq's fitness team who create and manage group sessions
Development Team	The people who build the system from this design
Testers	The people who check that the system behaves as specified
Maintainers	The people who update and extend the system after release

2.3 System Design Concerns		
No.	Stakeholder	Concerns to Address

2.3 System Design Concerns		
1	Development Team, Maintainers	Adding new activity types without changing existing code
2	End Users, Coaches	Receiving timely updates about sessions and booking decisions
3	Development Team, Product Owners	Using external services without depending on one provider
4	Testers	Testing each part of the system on its own
5	End Users	Keeping personal data secure

2.4 Client-Server Architectural Viewpoint

This viewpoint shows Intaliq's overall structure: how the system is divided between the client and the server, and how they communicate. It addresses concerns 4 and 5 by separating the system into independent parts and keeping all data access behind the server.

2.4.1 Client-Server Design View

In Intaliq, there is a client-server pattern; hence, Intaliq can be considered a Distributed System. There are two major parts of the system: the client and the server. The client refers to the part of the program that works with users, while the server is used for processing requests made by users. There are many clients (both users and coaches) that communicate with one server only; there is no direct communication between them.

Primary presentation: deployment diagram showing the User Smartphone running the Intaliq Mobile Application, connected over HTTPS to the Application Server running the Intaliq Backend, which connects to the Intaliq Database and the External AI Service.

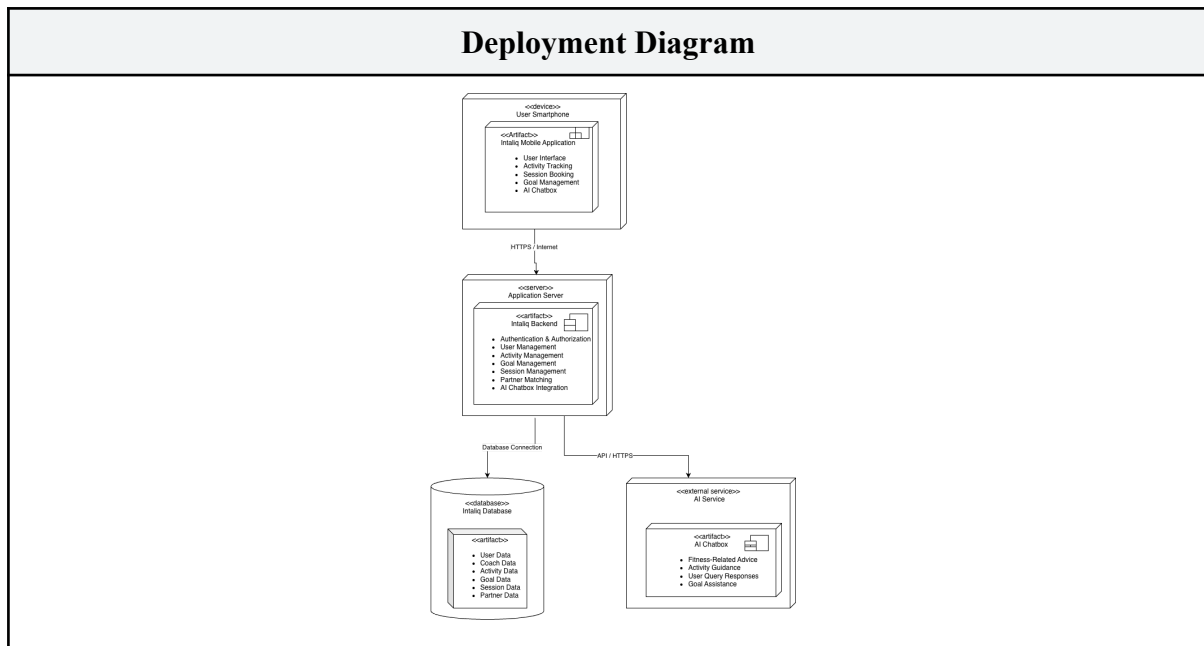
Element catalog		
Element	Role	Responsibility
Intaliq Mobile Application	Client	Displays screens for users and coaches, sends requests to the server, and stores actions locally through Offline Sync when the connection is lost

Element catalog		
Intaliq Backend	Server	Receives requests over HTTPS, checks authentication and permissions, runs the business logic, and returns results
Intaliq Database	Data store	Stores user, coach, activity, goal, session, and partner data
External AI Service	External service	Answers fitness questions for the AI Chatbox, reached by the server through an API over HTTPS

Communication flow:

1. The request is made by the user from a particular function in the mobile application, say, making an appointment.
2. The request is passed to the backend through HTTPS.
3. The backend verifies and processes the request.
4. The backend reads from or writes to the database, or calls the external AI service when needed.
5. The backend returns the result to the application.
6. The application displays the outcome to the user.

Rationale: Client-Server fits Intaliq because all users and coaches share the same data, such as sessions and partner requests, which must stay consistent in one central place. Keeping authentication on the server supports **security**, since the client never accesses the database directly. Server changes reach every client at once, which supports **modifiability**, and the server can be backed up to support **availability**. Broker was considered but rejected, since Intaliq has a single backend and doesn't need a mediator between multiple servers.



2.4.2 Justification

In Intaliq, users and coaches work with the same information at the same time. When a coach creates a session, every user must see the same details and the same number of available spots. As stated in the "Book a Session" scenario, the coach also needs to accept or decline the booking request, and the user must see the result. If each mobile application kept its own copy of the data, the information could become different between users, and two people might book the last spot.

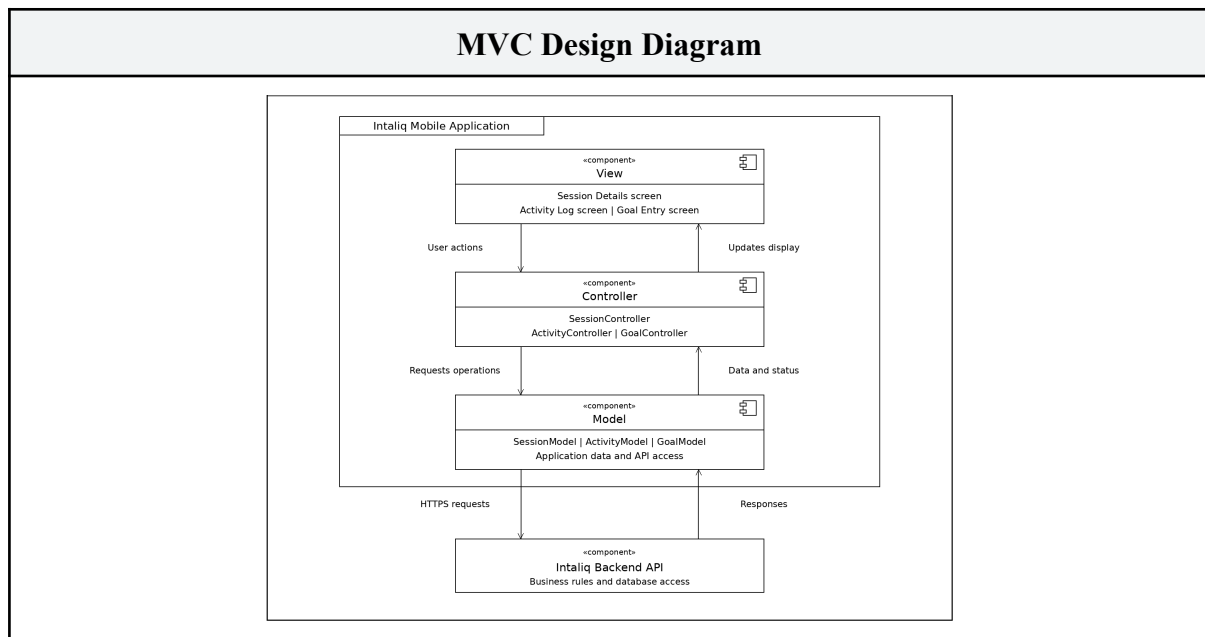
Client-Server was the best choice for Intaliq because the data has to be shared. In this architecture, the mobile application is just the part the user sees. It shows the screens and sends requests over HTTPS. The server receives listed requests and works with the database, storing everything in one place. That way all users and coaches get the same data. The phone never talks to the database directly, and the server checks every request first this leads to improved security. Maintainability is also better ,because if we need to update something in the backend the mobile application doesn't have to change. This supports the security requirement from phase 2 and also makes the system much easier to maintain.

2.5 Mobile Application Architectural Viewpoint

2.5.1 MVC Design View

The application of the Intaliq follows the MVC framework. The model takes care of the application data and interfaces with the backend API while the view handles the display of the data and the inputs of the users, and the controller dictates the users' actions.

Element	Responsibility
Model	Maintain data about sessions, activities and goals. Communicates with the backend API to retrieve and update information.
View	Displays screens for users, collects user input and presents results and error messages.
Controller	Handles actions from the View, coordinates operations through the Model, and updates the View with the results.



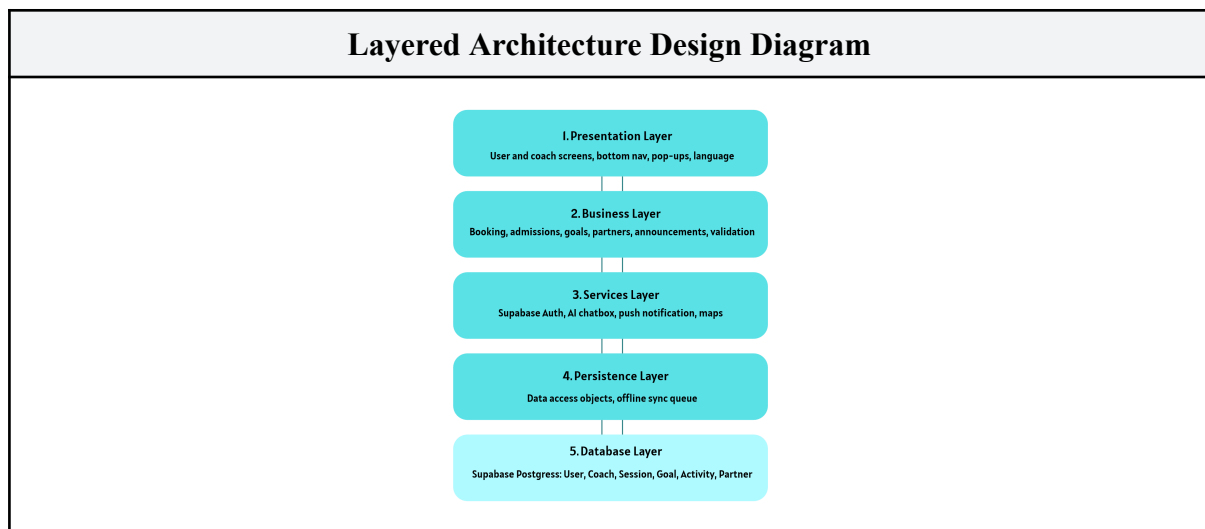
2.5.2 Justification

The MVC architectural pattern is a pattern that divides the architecture into three main blocks: the Model, which handles application data; the View, which handles the user interface; and finally, the Controller, which processes user actions. MVC is an appropriate

pattern for the Intaliq application because it separates the application into three main components: the user interface, application data, and control logic. The purpose of this separation is to improve the application’s maintainability, extensibility, and testability. The Model manages the data used by the application. The View represents the UI and how the data is presented to users; it also allows users to interact with the application. Finally, the Controller manages and processes actions received from the View and updates the View. For example, when a user books a class or group session, the View is responsible for sending the user’s request to the Controller. Then, the Controller processes the request and communicates with the backend through the Model. It also instructs the View to display the output, either a confirmation or an error message.

2.6 Backend Architectural Viewpoint

2.6.1 Layered Architecture Design View



2.6.2 Justification

For Intaliq, using a layered architecture for the backend comes with multiple benefits. This architectural approach divides the system into different layers in which each layer has its own responsibilities and performs its own functions and only communicates with its adjacent layer. This approach helps the system be easier to maintain due to changes in one layer not affecting the other layers. Direct access to the database is restricted because requests must

pass through the layer above it before this results in advancements in security. An additional benefit is improved testability due to each layer being tested and examined on its own. This architectural approach promotes reusability as the lower layers can be accessed by multiple components of the application. Ultimately, it improves development productivity and efficiency by allowing several different developers to work on different layers at the same time.

3. Detailed Design

3.1 Overview

This section presents Intaliq's detailed class-level design using Gang of Four (GoF) design patterns to achieve maintainability, testability, and loose coupling:

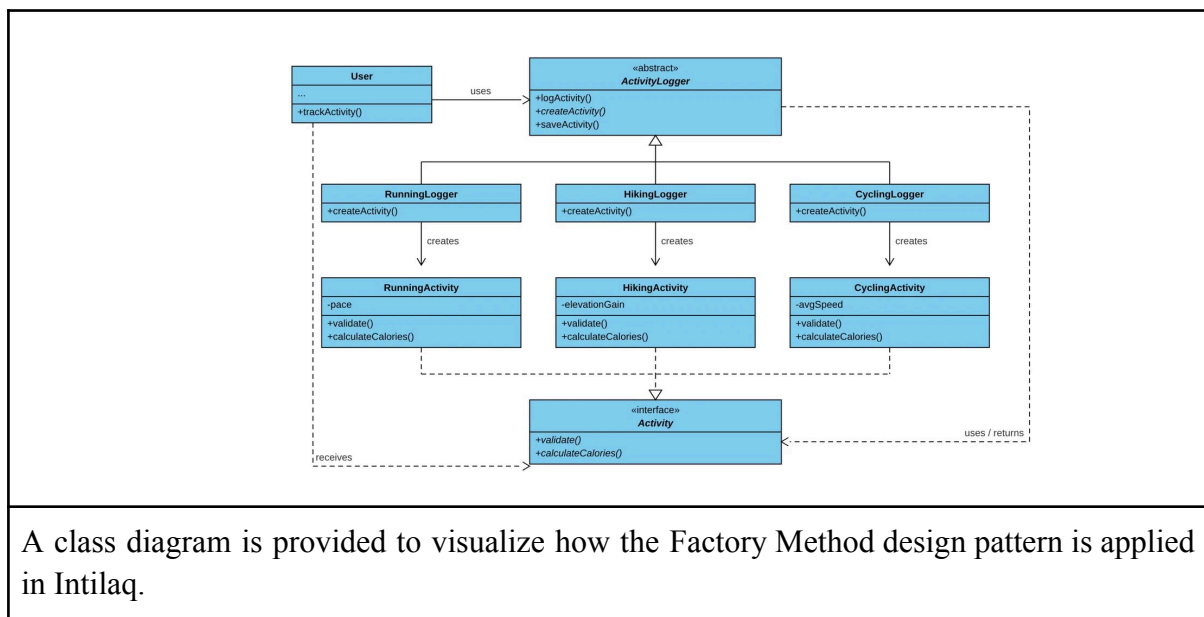
- **Factory Method (Creational):** Encapsulates creation logic for exercise activities (Running, Hiking, and Cycling) so new activity types can be added without altering existing code.
- **Observer (Behavioral):** Automatically notifies users and coaches of session updates and booking decisions while keeping session data decoupled from messaging logic.
- **Adapter (Structural):** Integrates third-party services (such as external AI chat services) by adapting incompatible external interfaces into standardized backend calls.

3.2 Factory Method (Creational)

3.2.1 How It Is Applied

The Factory Method design pattern can be applied for Intaliq to log their outdoor activities. This feature allows users to log 3 different types of outdoor activities: running, hiking, and cycling. These activities carry out the same logging process which consists of: validating the activities data, calculating the calories being burnt, and then saving the activity. Yet, each type of activity employs its own approach for validation of data and calculates burnt calories. Rather than relying on enabling the main logging process to determine the

corresponding activity object to be created, the task is assigned to a separate logger for each activity type. For example, when a user chooses hiking the HikingLogger creates a HikingActivity object which is utilized by the main logging process without having to know the exact type of activity. This design makes our system much more flexible if we decide to add another activity in the future such as padel by creating its own dedicated activity loggers and classes without changing the current logging process.



3.2.2 Justification

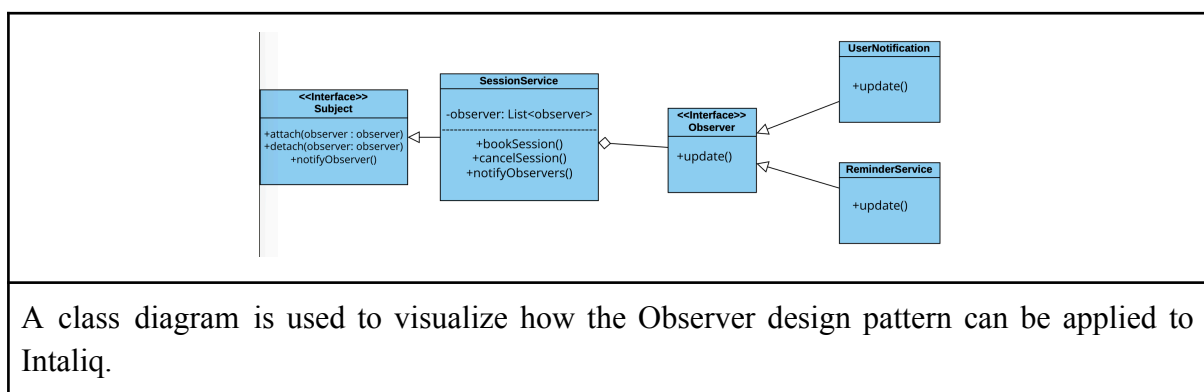
For Intiliq, using the factory method comes with many advantages. That being said, the factory method actually improves testability, while making it easier to maintain specific parts of the code. This method also allows reuse of specific parts of the code. Moreover, we can have the same code working with various existing or newly created product classes. Finally, this method allows development to reach peak efficiency by allowing different developers to work on different parts of the project simultaneously.

3.3 Observer (Behavioral)

3.3.1 How It Is Applied

The Observer design pattern can be applied to Intiliq's notification and update system. The subject is the sessions service, while the observers will be the user notification

and reminder services. When an event occurs, for example, a group class is cancelled, the subject can notify all the observers. Then each observer can perform its own independent action. The user notification observer will send notifications to the user and the reminder service may change or modify the reminder schedule. This pattern is useful because it improves flexibility because new response mechanisms like notification reminders can be modified, added, removed without changing the main session service. This pattern also reduces the dependency between business logic and the UI and notification components.



3.3.2 Justification

Several events in Intaliqu must be communicated to more than one party at the same time. When a coach announces an update to a scheduled session, every admitted member must be informed. As stated in the “Book a Session” scenario, when the coach accepts or declines the booking request, the requesting user needs to be informed about it. In case the class “Session” would be in charge of informing every single user directly, it will get tightly coupled to the notifying operation and any additional notification method would need changes in the class “Session.”

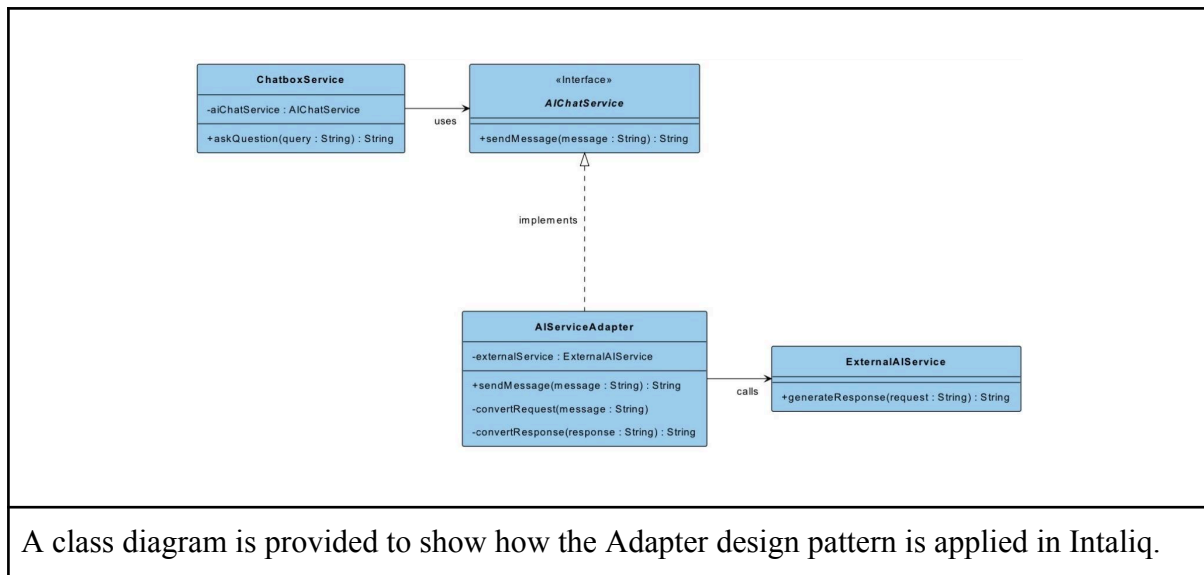
The Observer pattern was chosen because it solves this problem directly. The Session acts as the subject and only needs to announce that a change has occurred. The registered observers, such as the admitted users and the Notifications component, are then updated automatically. This keeps the Session class focused on managing session data rather than on how messages are delivered.

The pattern also supports the quality attributes identified in Phase 2. It promotes usability because users get notifications for any modifications that happen to the session they have joined. At the same time, it facilitates maintainability in that additional observers like email or in-app notification services can be added without changing the current Session class.

3.4 Adapter (Structural)

3.4.1 How It Is Applied

The Adapter design pattern can be applied to our AI Chatbox in Intaliqu. In this feature, the user types a fitness related question and eventually gets an answer from an external AI service. The problem is that this external service has its own distinctive way of receiving requests and sending back answers, and it is not the same format our backend uses. So the backend could not talk to it directly. To fix this, our solution was putting an adapter in the middle. The ChatboxService only calls a standard interface named AIChatService, using its sendMessage() method. The AIServiceAdapter implements that interface. When a user asks a question, the adapter will change it to the format the external service understands and sends it over HTTPS. After the service replies the adapter changes the answer back to the format our backend uses, and then the user sees it in the app. This way the ChatboxService doesn't need to know anything about how external service works. If we decide to use a different AI service later on, we just write a new adapter for it and the rest of the backend stays exactly the same.



3.4.2 Justification

The reason for using the Adapter pattern is the ability to make the ChatboxService in Intaliq use some external AI service with a unified interface. This way, provider-specific request/response processing logic is decoupled from chatbox logic in the application.

This makes the software more maintainable since any changes in the API of the external service can be processed in AServiceAdapter. At the same time, it becomes possible to add another AI service provider by implementing the same AChatService interface; only little modifications will be needed in ChatboxService.

4. Reference

- Albugami, Y., Al Rakan, R., Alomran, S., Alghofaili, F., Almuqrin, M., & Alfari, M. (2026). *Intaliq: Software architecture design* [Project Deliverable 1, SE322]. Software Architecture. <https://software-architecture.shoug-tech.com>
- Albugami, Y., Al Rakan, R., Alomran, S., Alghofaili, F., Almuqrin, M., & Alfari, M. (2026). *Intaliq: Architecture documentation* [Project Deliverable 2, SE322]. Software Architecture. <https://software-architecture.shoug-tech.com>

AlOmayrini, D., Alrakan, R., Albagma, W., & Albugami, Y. (2026). *Intaliq 2.0 software requirements specification* [SE311 project document]. Software Architecture. <https://software-architecture.shoug-tech.com>

5. Glossary

Term	Definition
Activity	An outdoor exercise logged by a user: Running, Hiking, or Cycling
Adapter Pattern	A structural pattern that converts one interface into another that the system expects
API	Application Programming Interface: a defined way for software components to communicate
Architectural Style	A general structure for organizing a system's components, such as Client-Server
Architectural Viewpoint	A perspective used to describe the design for a specific set of stakeholder concerns
Behavioral Pattern	A pattern concerned with how objects communicate and share responsibilities
Client-Server Architecture	A style where clients send requests to a central server that processes them and returns results
Coach	A member of Intaliq's fitness team who creates and manages group sessions
Creational Pattern	A pattern concerned with how objects are created
Design Pattern	A reusable solution to a common problem in software design
Design View	The application of a viewpoint to a specific part of the system
Factory Method Pattern	A creational pattern that lets a dedicated class decide which object type to create
GoF	Gang of Four: Gamma, Helm, Johnson, and Vlissides, authors of the original design patterns book
Goal	A fitness target a user sets for an activity, with a time, calories, and deadline

HTTPS	Hypertext Transfer Protocol Secure: encrypted communication between the app and the server
Observer Pattern	A behavioral pattern where a subject automatically notifies all of its observers when its state changes
Offline Sync	A mobile app feature that stores actions locally and syncs them when the connection returns
Partner	Another user a person connects with to exercise together
SAD	Software Architecture Design document
SDD	Software Design Document
Session	A coach-led group activity with a date, time, location, capacity, and intensity
SRS	Software Requirements Specification
Stakeholder	Anyone with an interest in the system, such as users, coaches, or developers
Structural Pattern	A pattern concerned with how classes and objects are combined into larger structures