COMP2511

Tutorial 8

Last Tutorial

- Abstract Factory Pattern
- Decorator Pattern
- Singleton Pattern

This Tutorial

- Introduction to Software Architecture
- C4 Diagrams
- Sequence Diagrams

Introduction to Software Architecture

- **Software architecture** defines the fundamental high-level structure of a software system.
- It defines how system components are structured, how they interact and the guiding **characteristics** (topics for next week!) that shape its evolution.
- Specifically, it focuses on:
  - How the system is divided into modules (components), services and layers for the front-end, back-end, database etc.
  - How data moves between different components
  - What technologies, frameworks and infrastructure are used
  - Considerations such as security, scalability, performance

Relation to Software Design

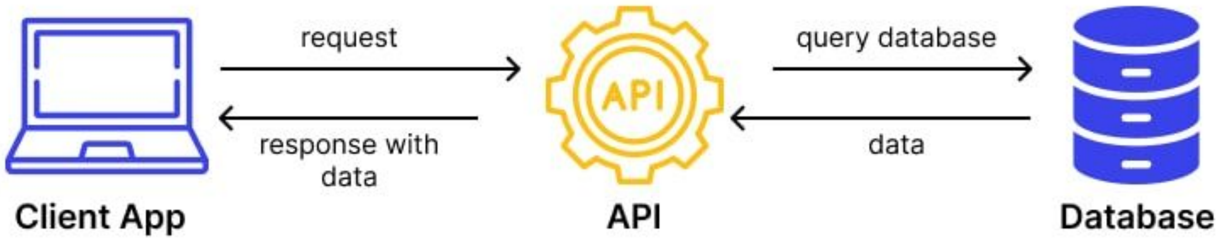
- So far, we have dealt with software design at a code level, thinking about how individual parts of a system are **implemented**.
  - This includes detailed decisions about the classes we model, the methods we write and the design patterns we decide to use.
- Architecture operates at a more **macro-level**, and involve decisions regarding the **structure** of a system; i.e. its underlying basis. These decisions are much more difficult to change than design decisions.

Design vs. Architecture

- The foundation of a building is a macroscopic detail - it guides the overall structure of the building, and once it is set it is difficult to change anything else - this is like **architecture**.
- Specific details like what wallpaper is used, the design of the door etc. are examples of the smaller details - this is like **design**.

Application Programming Interfaces (APIs)

- APIs are ways for different applications to communicate and exchange data with each other.
  - Especially for web applications, these typically consist of HTTP requests/responses via. JSON or XML, using GET, POST, PUT and DELETE methods.
  - Think of HTTP as the common 'language' that defines how messages are formatted and exchanged between web applications/servers.



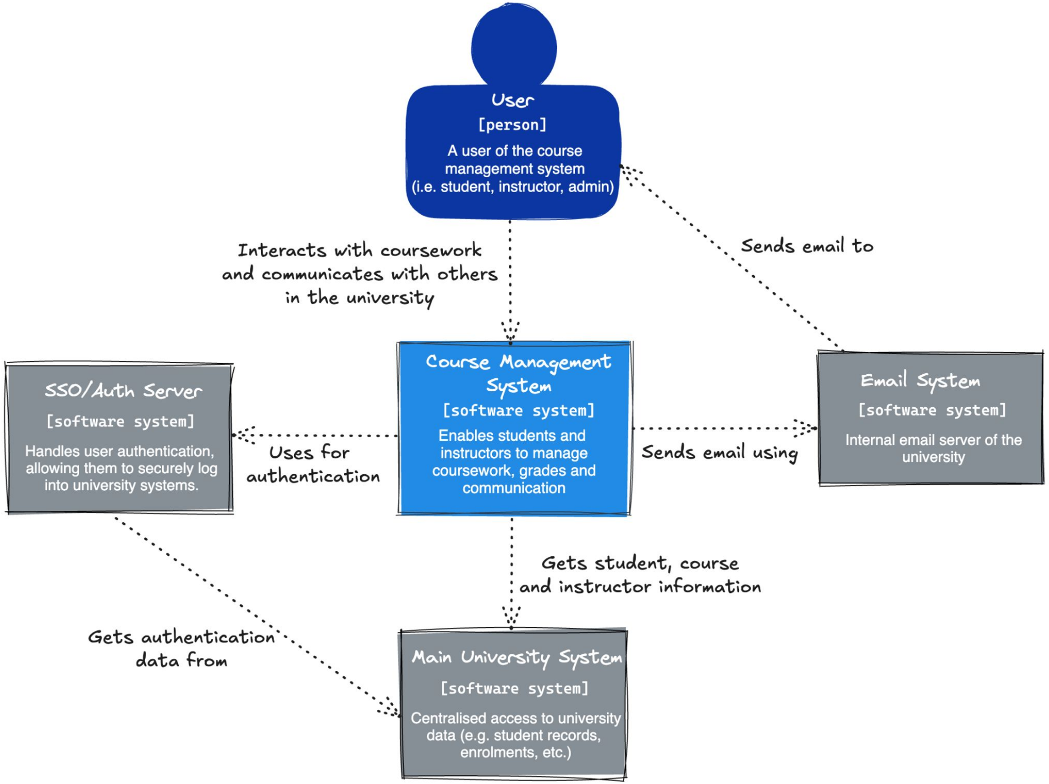
C4 Model

- The C4 Model is a graphical notation technique for modelling the architecture of software systems, at **four** different levels of abstraction.
- It bridges the gap between high-level system overviews and low-level code details, ensuring alignment between stakeholders, developers, and architects.
- The four levels that can be conveyed by the C4 Model are:
  - **Context Diagram**: Shows the system as a "box" and its interactions with users and external systems.
  - **Container Diagram**: Breaks the system into containers (applications/services/databases) and shows how they interact.
  - Component Diagram: Zooms into a specific container to show its internal components and their relationships.
  - Code (Class) Diagram: Offers a detailed view of the source code structure (e.g., classes and interfaces) within a component - e.g. your UML diagrams!

Context Diagrams

- A Context Diagram communicates the scope (primary users, what the application does) and external dependencies of the system in a simple and accessible way. Hence, it is primarily meant for people with little to no technical background.
- **Example:** You're a team designing a university Course Management System (i.e. Moodle) used by students and instructors. The system allows:
  - Students to view and submit assignments, receive grades and participate in forums
  - Instructors to create and grade assignments, post announcements and manage students
  - The system has a web frontend, backend, database and integrates with other university systems like authentication and email.

Context Diagram Example



User

[person]

A user of the course management system (i.e. student, instructor, admin)

Interacts with coursework and communicates with others in the university

Sends email to

Course Management System

[software system]

Enables students and instructors to manage coursework, grades and communication

SSO/Auth Server

[software system]

Handles user authentication, allowing them to securely log into university systems.

Uses for authentication

Sends email using

Email System

[software system]

Internal email server of the university

Gets student, course and instructor information

Gets authentication data from

Main University System

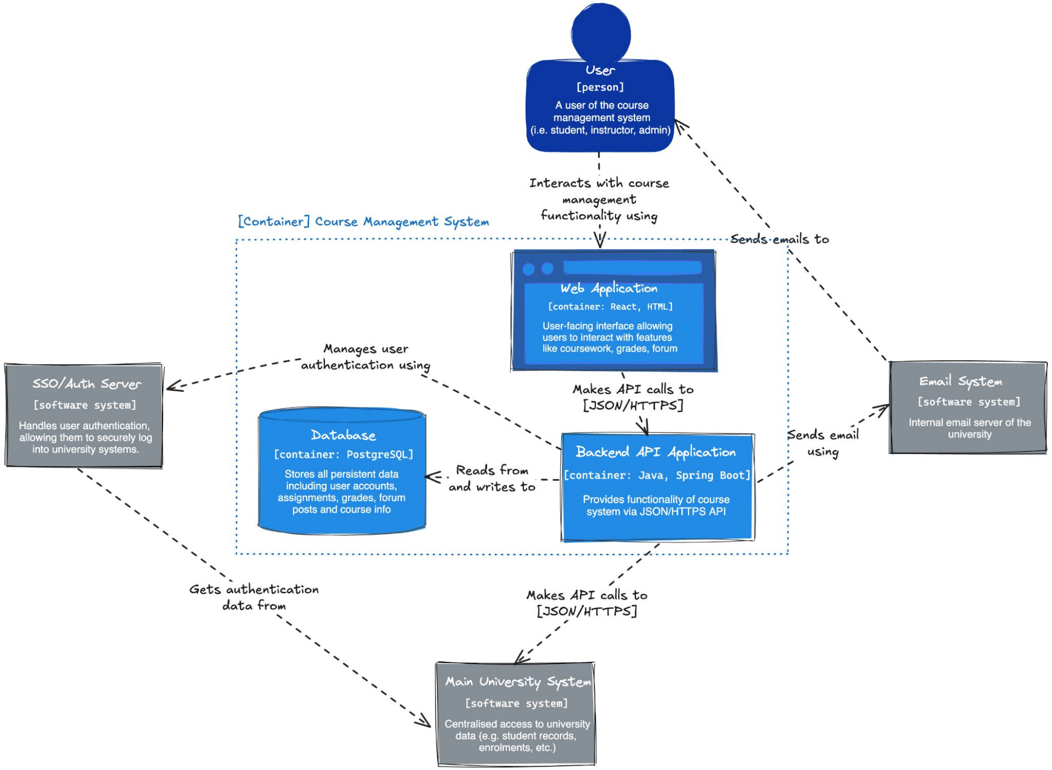
[software system]

Centralised access to university data (e.g. student records, enrolments, etc.)

Container Diagrams

- Think of the Container Diagram as a Context Diagram, but **zoomed into the actual system** - this means representing its major deployable units (e.g. web apps, mobile apps, databases, microservices etc. - these are the *containers*) within the system, and how they communicate with each other.

Container Diagram Example



User

[person]

A user of the course management system (i.e. student, instructor, admin)

Interacts with course management functionality using

Sends emails to

[Container] Course Management System

Web Application

[container: React, HTML]

User-facing interface allowing users to interact with features like coursework, grades, forum

Manages user authentication using

SSO/Auth Server

[software system]

Handles user authentication, allowing them to securely log into university systems.

Makes API calls to [JSON/HTTPS]

Database

[container: PostgreSQL]

Stores all persistent data including user accounts, assignments, grades, forum posts and course info

Reads from and writes to

Backend API Application

[container: Java, Spring Boot]

Provides functionality of course system via JSON/HTTPS API

Sends email using

Email System

[software system]

Internal email server of the university

Gets authentication data from

Makes API calls to [JSON/HTTPS]

Main University System

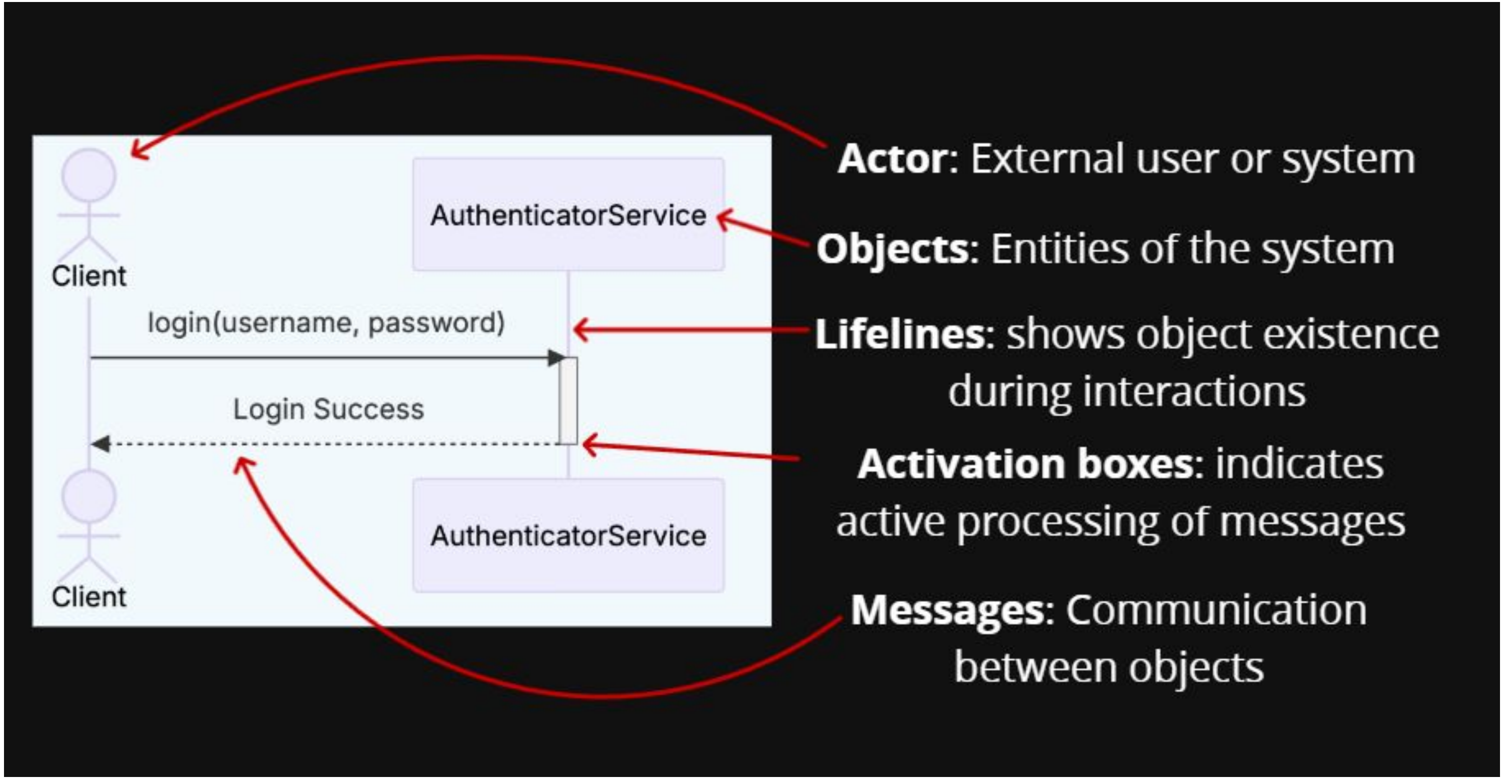
[software system]

Centralised access to university data (e.g. student records, enrolments, etc.)

Sequence Diagrams

- A Sequence Diagram illustrates the temporal order of interactions (i.e. the order w.r.t. time) between objects or components to achieve a specific functionality or use case. It is suitable for visualising the flow of control and messages between the components of a system over time.
  - This is contrast to a **class diagram** (e.g. UML diagrams) - these show the **static** structure of a system, including classes, their attributes, methods and the relationships between them - these describe *what* the system is made of, while sequence diagrams describe *how* the system behaves over time.
- Sequence diagrams often do not show the entire system, instead they are only modelling a specific functionality of the system.
- Course Notes on Sequence Diagrams - excellent resource!

Sequence Diagrams: Key Elements (credit to Rebecca!)



Actor: External user or system

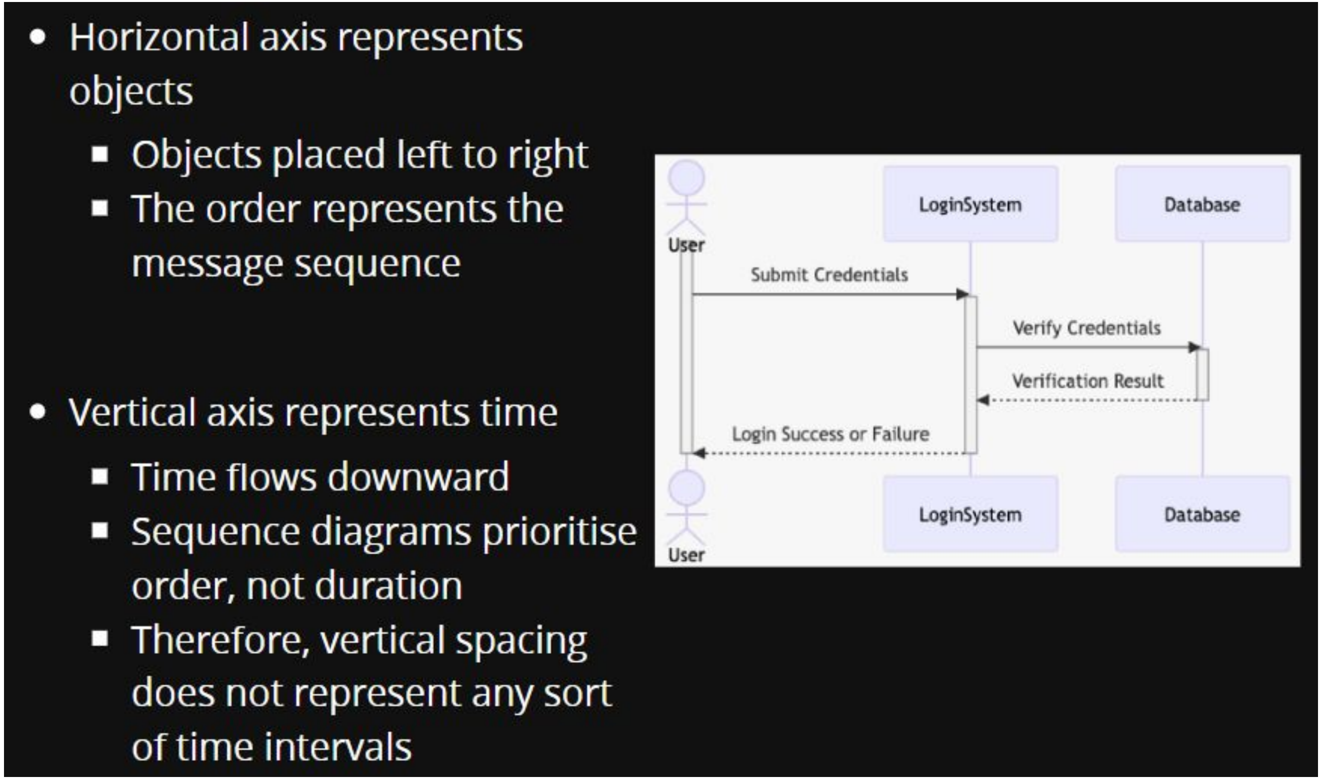
Objects: Entities of the system

Lifelines: shows object existence during interactions

Activation boxes: indicates active processing of messages

Messages: Communication between objects

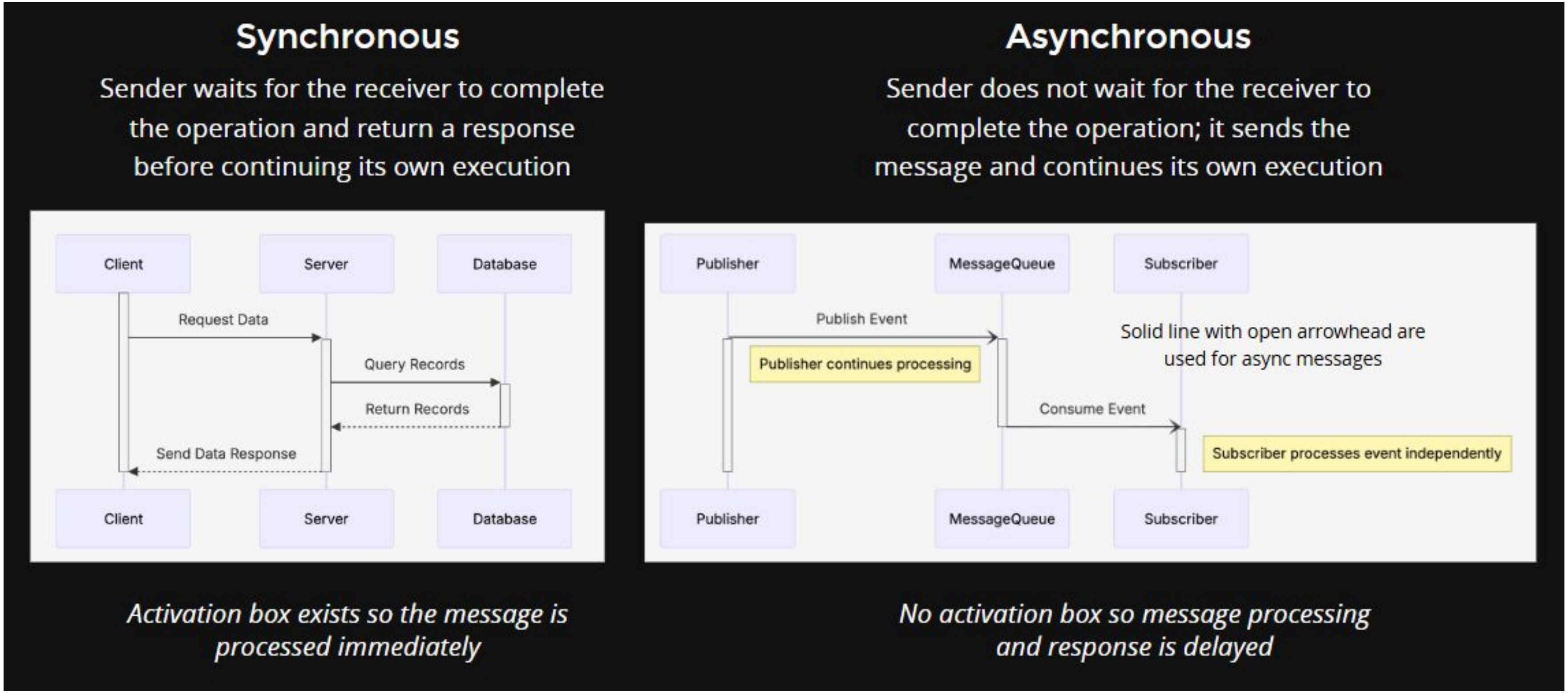
Sequence Diagrams: Axes (credit to Rebecca!)



- Horizontal axis represents objects
  - Objects placed left to right
  - The order represents the message sequence

- Vertical axis represents time
  - Time flows downward
  - Sequence diagrams prioritise order, not duration
  - Therefore, vertical spacing does not represent any sort of time intervals

Sequence Diagrams: Messages (credit to Rebecca!)



Synchronous

Sender waits for the receiver to complete the operation and return a response before continuing its own execution

Activation box exists so the message is processed immediately

Asynchronous

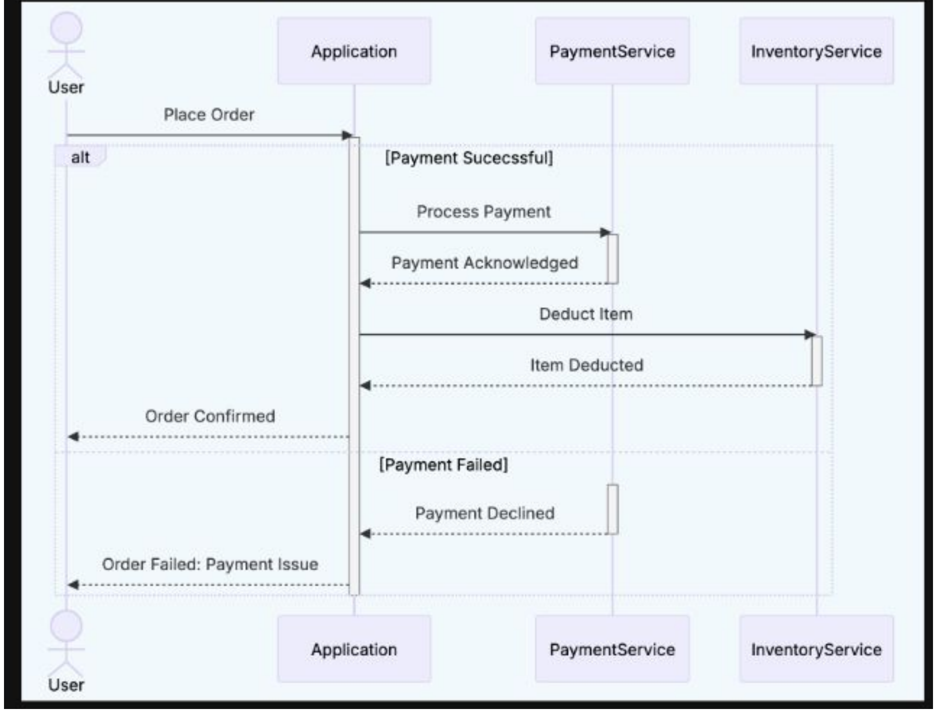
Sender does not wait for the receiver to complete the operation; it sends the message and continues its own execution

Solid line with open arrowhead are used for async messages

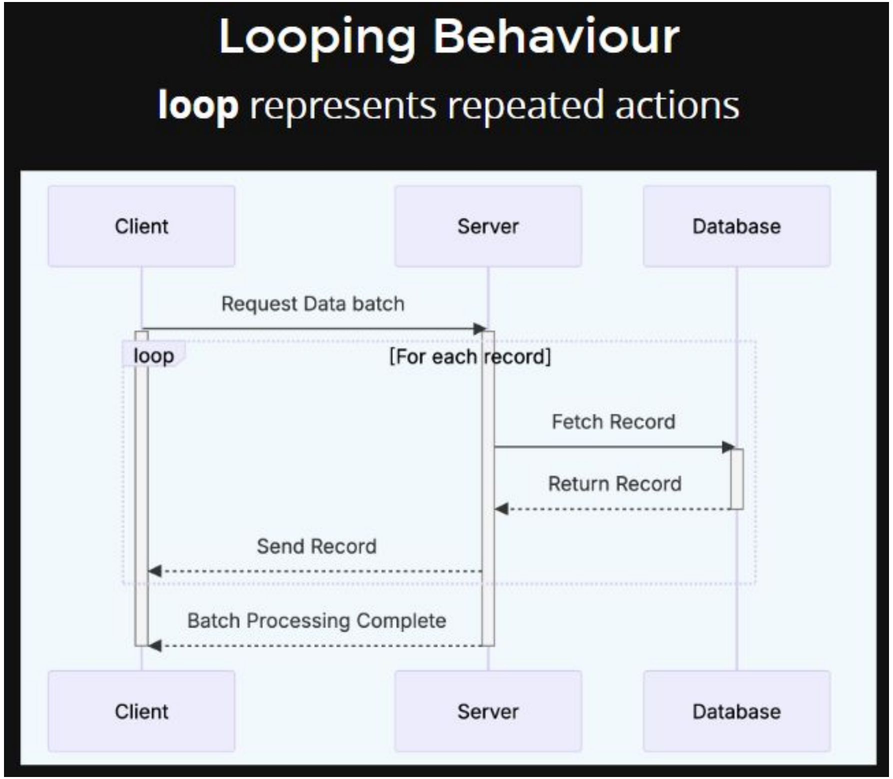
No activation box so message processing and response is delayed

Sequence Diagrams: Conditionals (credit to Rebecca!)

alt represents alternate scenarios



Sequence Diagrams: Loops (credit to Rebecca!)



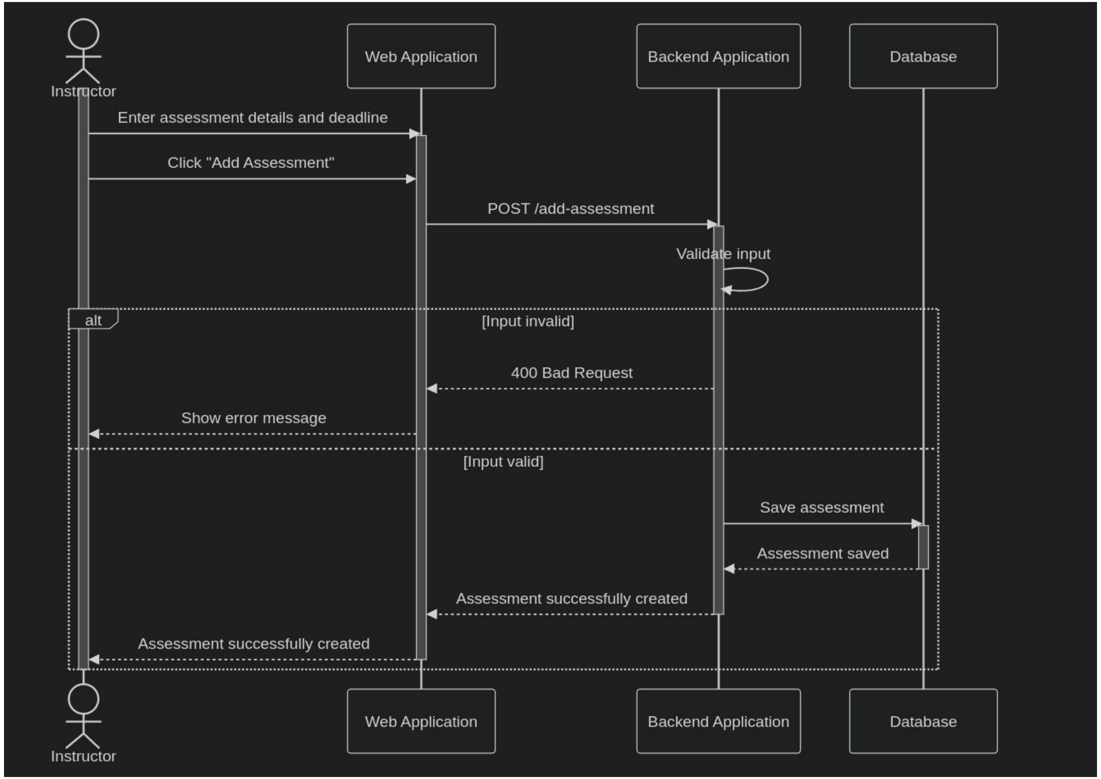
Looping Behaviour

loop represents repeated actions

Live Example: Sequence Diagrams

- **Example:** You are part of the development team designing the Course Management System. Create a Sequence Diagram illustrating the flow when an instructor making an assessment with a deadline.
  - Instructor creates a new assessment by clicking on “Add Assessment” button on the Web Application
  - Web Browser sends data to the Backend API Application
  - Backend validates input
    - If input invalid, Backend sends to frontend 400 Bad request
    - Else, continues with saving the assessment
  - Backend attempts to save the assessment data to Database
  - Database confirms save
  - Backend sends confirmation of successful assessment creation to Frontend
  - Frontend provides result notification to the Instructor

Sequence Diagram Example


Instructor
Web Application
Backend Application
Database
Enter assessment details and deadline
Click "Add Assessment"
POST /add-assessment
Validate input
alt
[Input invalid]
400 Bad Request
Show error message
[Input valid]
Save assessment
Assessment saved
Assessment successfully created
Assessment successfully created
Instructor
Web Application
Backend Application
Database