# COMP2511

## WEEK 8

What train line do you take?

# ADMIN STUFF

- Assignment-i manual marking feedback released.
  - Available on feedback/ branch in feedback.md
- Assignment-ii requires Moodle submission for Part 2.
- Lab09 is an attendance lab which means that attendance is mandatory in order to get marks
- There will be an sample exam for Lab10 (for in-person labs) using the exam environment.
  - Try to attend an in-person lab if you can!

# A G E N D A

- Introduction to Software Architecture
- C4 Models
- Sequence Diagrams

# Introduction to Software Architecture

# WHAT IS SOFTWARE ARCHITECTURE AND WHY IS IT IMPORTANT?

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 (which we will learn next week!) that shape its evolution.

Specifically, it focuses on:

- How the system is divided into modules, services, and layers for the frontend, backend, database, etc.
- How data flows between different components
- What technologies, frameworks or infrastructure are used
- Architectural characteristics that should be addressed (e.g. security, scalability, performance, etc.)

# WHAT IS SOFTWARE ARCHITECTURE?

**How is it different to software design that we have been learning so far in this course?**

- So far, we have dealt with software design at a **code-level**, think about how individual parts of a system are implemented. This includes detailed decisions about classes, methods, controllers and design patterns.
- Architecture operates at more **macro-level, system-wide** and involve more structural decisions of a system that take more effort to change. Typically involve multiple components or subsystems.
- e.g. For communication, “services communicate via RESTful APIs” (architectural) vs. “use the observer pattern to update UI when data changes” (design).

## Why is it important to learn architecture?

- Helps manage complexity especially with large systems, dividing it into manageable, understandable parts.
- Software is never static, it always grows, adapts and changes according to new requirements and the architecture can guide the technical vision and direction of teams in the long-term.

# C4 Model

## 7. What is the primary purpose of the C4 model in software architecture documentation? Name the four core diagrams in the C4 model, in order of increasing detail.

The C4 model helps teams visualize and communicate the software architecture of a system clearly at 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.

**7. What is the primary purpose of the C4 model in software architecture documentation? Name the four core diagrams in the C4 model, in order of increasing detail.**

Four Core Diagrams (from least to most detailed):

1. **Context Diagram**: Shows the system as a "box" and its interactions with users and external systems.
2. **Container Diagram**: Breaks the system into containers (applications/services/databases) and shows how they interact.
3. **Component Diagram**: Zooms into a specific container to show its internal components and their relationships.
4. **Code (Class) Diagram**: Offers a detailed view of the source code structure (e.g., classes and interfaces) within a component.

# C4 MODEL - CONTEXT DIAGRAM

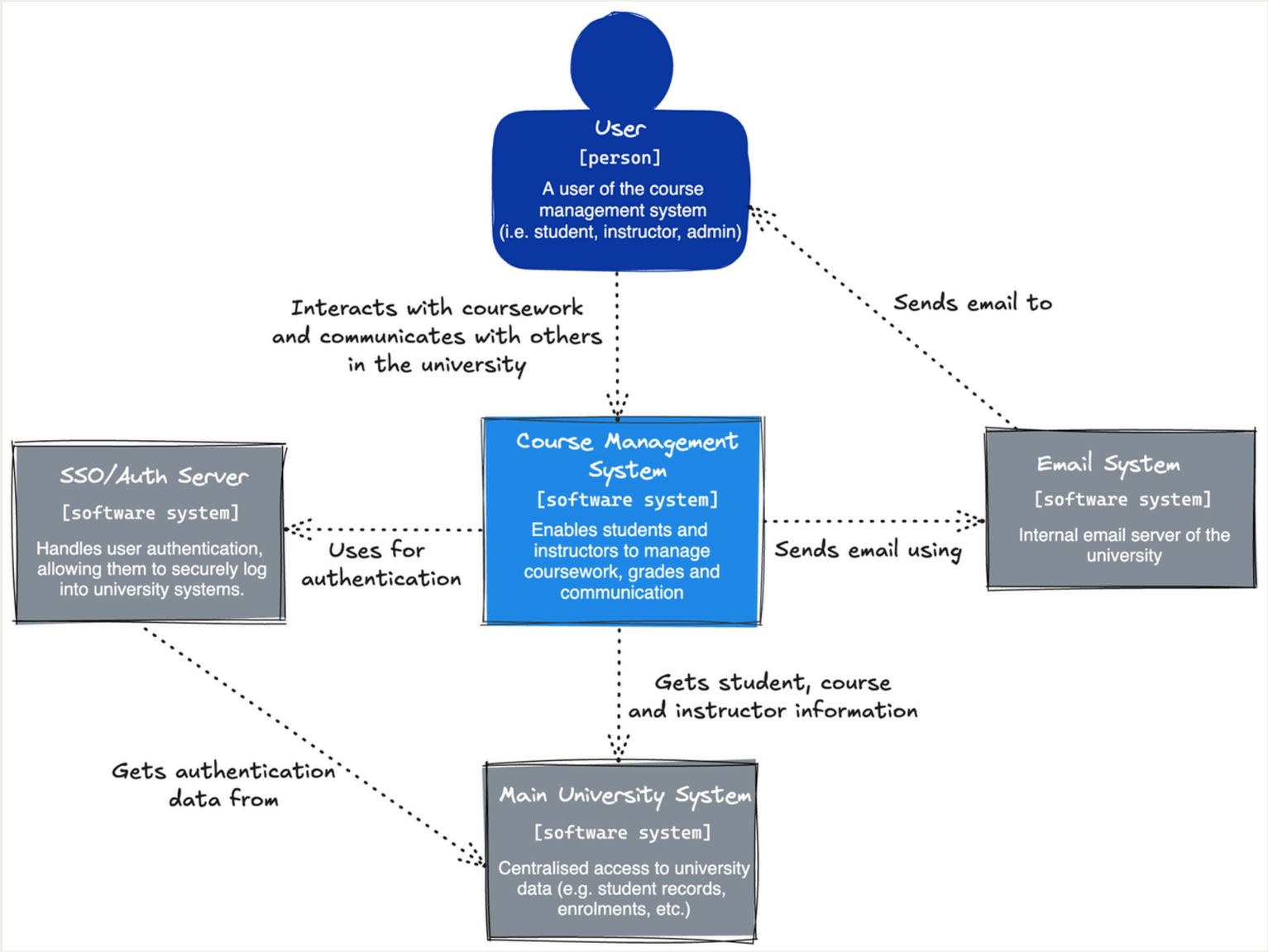
Example: You’re a team designing a university Course Management System (i.e. Moodle) used by students and instructors.

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.

# C4 MODEL - CONTEXT DIAGRAM



## 8. Who is the intended audience for the Context Diagram in the C4 model?

The Context Diagram is meant for:

- Non-technical stakeholders (e.g., business sponsors, product owners)
- External users or clients
- Enterprise architects
- Anyone who needs to understand what the system is, what it interacts with, and who uses it

It communicates the scope and external dependencies of the system in a simple and accessible way.

# C4 MODEL - CONTEXT DIAGRAM

We now extend the diagram by drawing the context diagram. We also use the questions on the following slides to motiviate our choices.

## 10. Why separate the web frontend and backend service into different containers?

- **Separation of concerns**: UI logic is handled in the frontend; business logic and data processing in the backend.
- **Independent deployment**: You can update the frontend without touching the backend and vice versa.
- **Scalability**: Backend and frontend can be scaled independently depending on load.
- **Technology flexibility**: Different teams can use the most suitable tech stacks for each (e.g., React frontend + Node.js backend).
- **Security**: Backend can be isolated in a secure environment, while frontend runs in browsers.

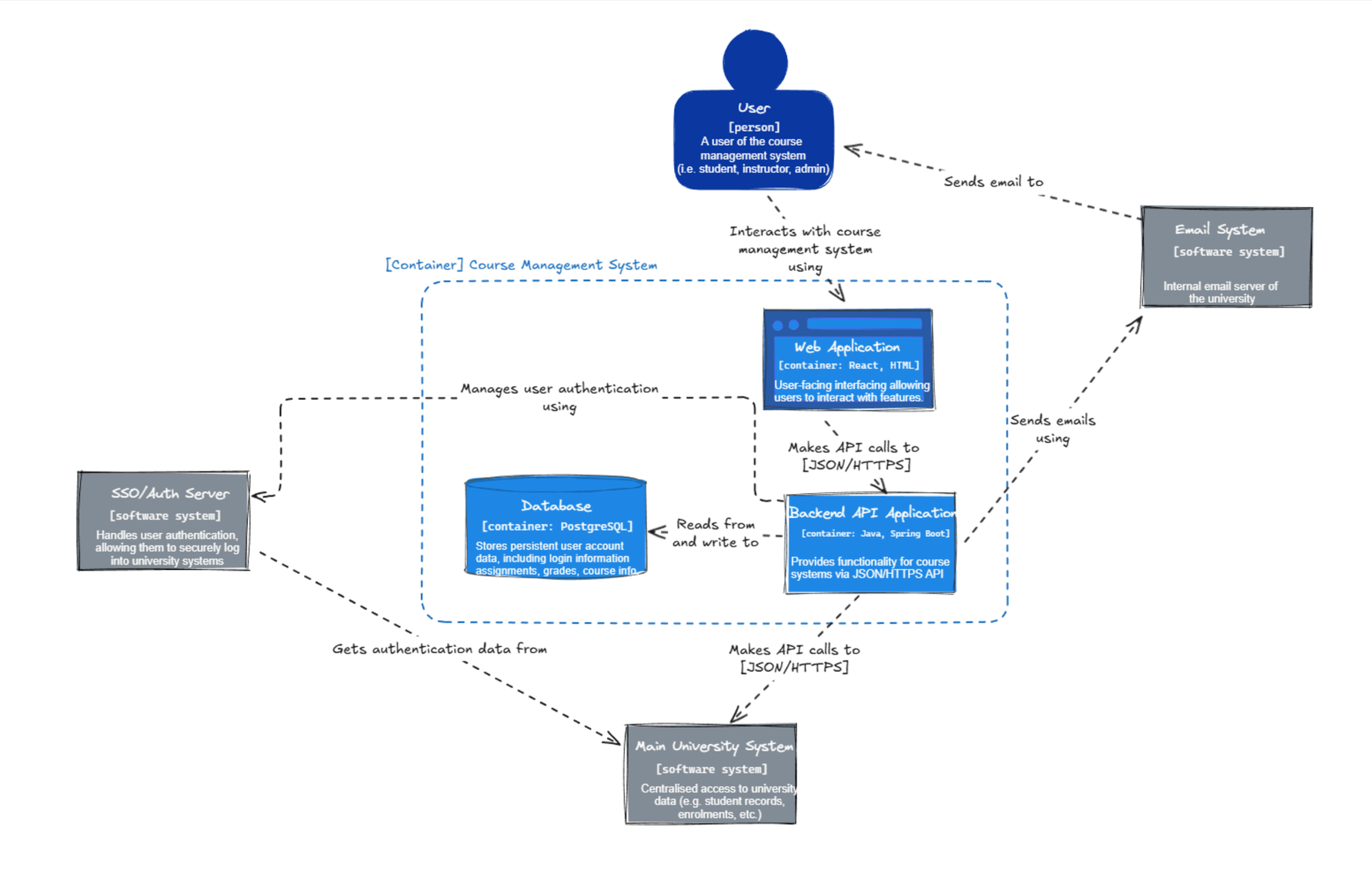
## 12. Why is it important to describe the responsibilities of each container in the diagram?

- **Clarifies Purpose**: Immediately conveys what each container is designed to do, helping stakeholders understand its role in the overall system.
- **Facilitates Understanding**: Makes the diagram more readable and understandable for a diverse audience, including new team members, business stakeholders, and operations teams.
- **Aids Communication**: Provides a common vocabulary for discussing the system's architecture.
- **Supports Decision Making**: Helps in evaluating dependencies, potential bottlenecks, and areas for improvement or refactoring. For example, if a container has too many responsibilities, it might indicate a need for decomposition.

## 12. Why is it important to describe the responsibilities of each container in the diagram?

- **Guides Development**: Provides clear boundaries and responsibilities for development teams.
- **Assists Troubleshooting**: When issues arise, knowing a container's responsibilities helps in quickly identifying the potential source of the problem.
- **Enables Future Planning**: Helps in assessing the impact of changes or additions to the system.

# C4 MODEL - CONTEXT DIAGRAM



## C4 MODEL - CONTEXT DIAGRAM

Currently, all functionality of the Course Management System is implemented within a **single monolithic backend**. What might be the benefits of splitting this monolith into **smaller, independent services**, and how could we divide the system's functionality to achieve better modularity, cohesion, and scalability?

Monolithic backend means all functionality (user management, course handling, grading, forums, etc.) exists in one large codebase and is deployed together. While this is simpler initially, it becomes harder to maintain and scale as the system grows.

- Better modularity and cohesion (teams can work on their specific services and not have to touch other parts of the backend). Codebases are then smaller, easier to understand and test.
- By decoupling the backend into each service, there is independent deployability and fault isolation. When one service fails, it does not bring down the entire system.

# Sequence Diagrams

# SEQUENCE DIAGRAMS

A sequence diagram is an interaction diagram showing the **temporal order of interactions** between **objects** or **components** to achieve a **specific functionality** or use case.

- Show how operations are carried out through message exchanges
- Excellent for visualising the flow of control and messages over time.

# SEQUENCE DIAGRAMS

1. When would you choose to use a Sequence Diagram instead of a Class Diagram?

**Sequence Diagram**: when you need to illustrate the **temporal order of interactions** between objects or components to achieve a specific functionality or use case. It's excellent for visualizing the flow of control and messages over time.

**Class Diagram**: when you want to show the **static structure of a system**, including classes, their attributes, methods, and the relationships between them (like inheritance, association, aggregation, composition). It describes what the system is made of, not how it behaves over time.

## SEQUENCE DIAGRAMS

2. Describe the key elements of a Sequence Diagram (actor, lifeline, activation bar, messages). Provide one example of each element.

The key elements of a Sequence Diagram are:

- Actor: Represents an external entity (user, another system) that interacts with the system.
  - Example: User (Initiates a login process).
- Lifeline (or Object Lifeline): Represents an individual participant (object or component instance) in the interaction over time. It's a dashed vertical line.
  - Example: AuthenticatorService (The service responsible for authenticating users).

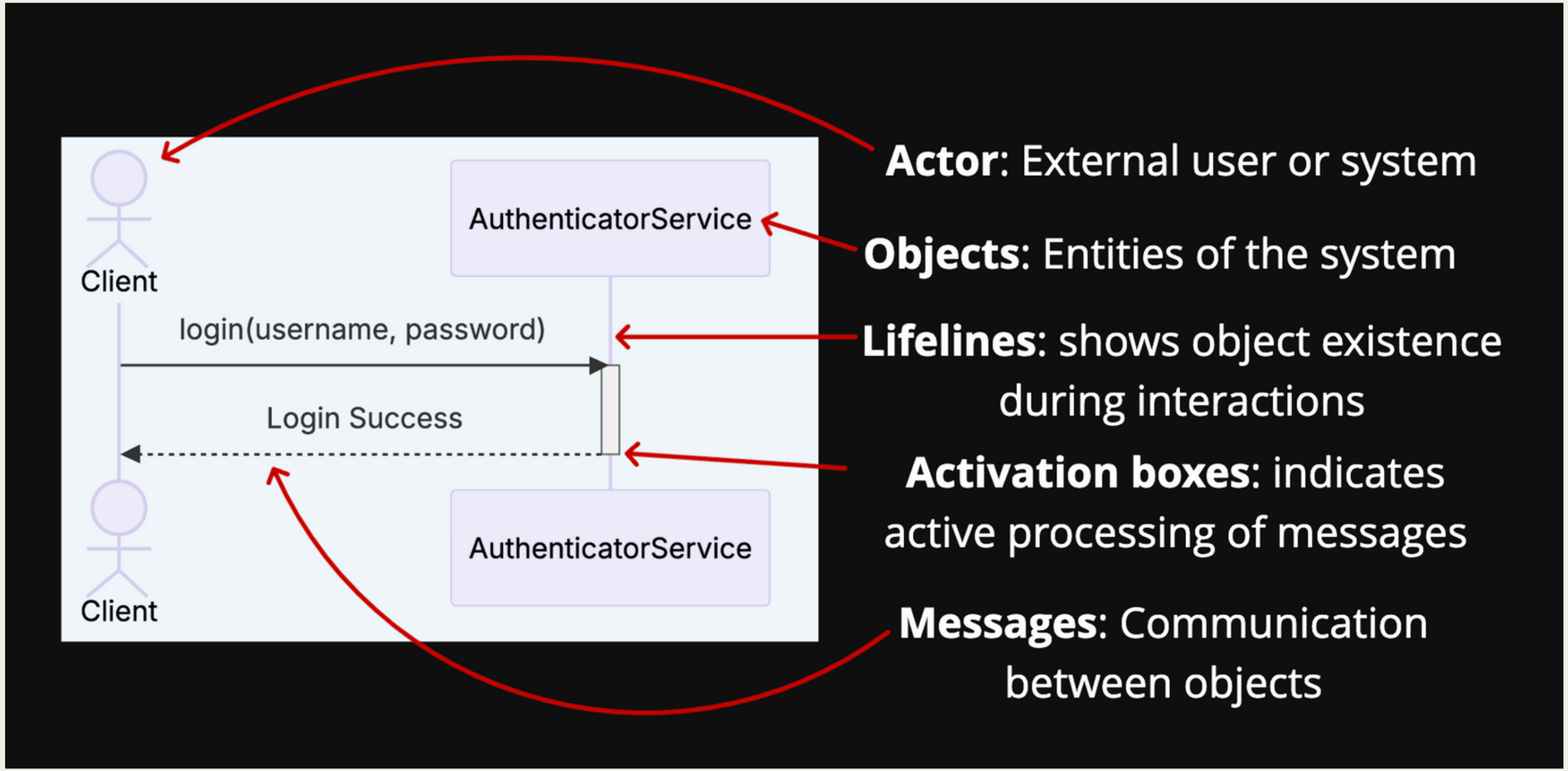
## SEQUENCE DIAGRAMS

2. Describe the key elements of a Sequence Diagram (actor, lifeline, activation bar, messages). Provide one example of each element.

- Activation Bar (or Execution Occurrence): A thin rectangle on a lifeline indicating when an object is actively performing an operation (i.e., when it has control).
  - Example: A bar on the AuthenticatorService lifeline from the moment it receives a login() message until it sends a verifyCredentials() message.
- Messages: Represent communication between objects. They are horizontal arrows between lifelines.
  - Example: login(username, password) (A message from User to AuthenticatorService).

## SEQUENCE DIAGRAMS

2. Describe the key elements of a Sequence Diagram (actor, lifeline, activation bar, messages). Provide one example of each element.

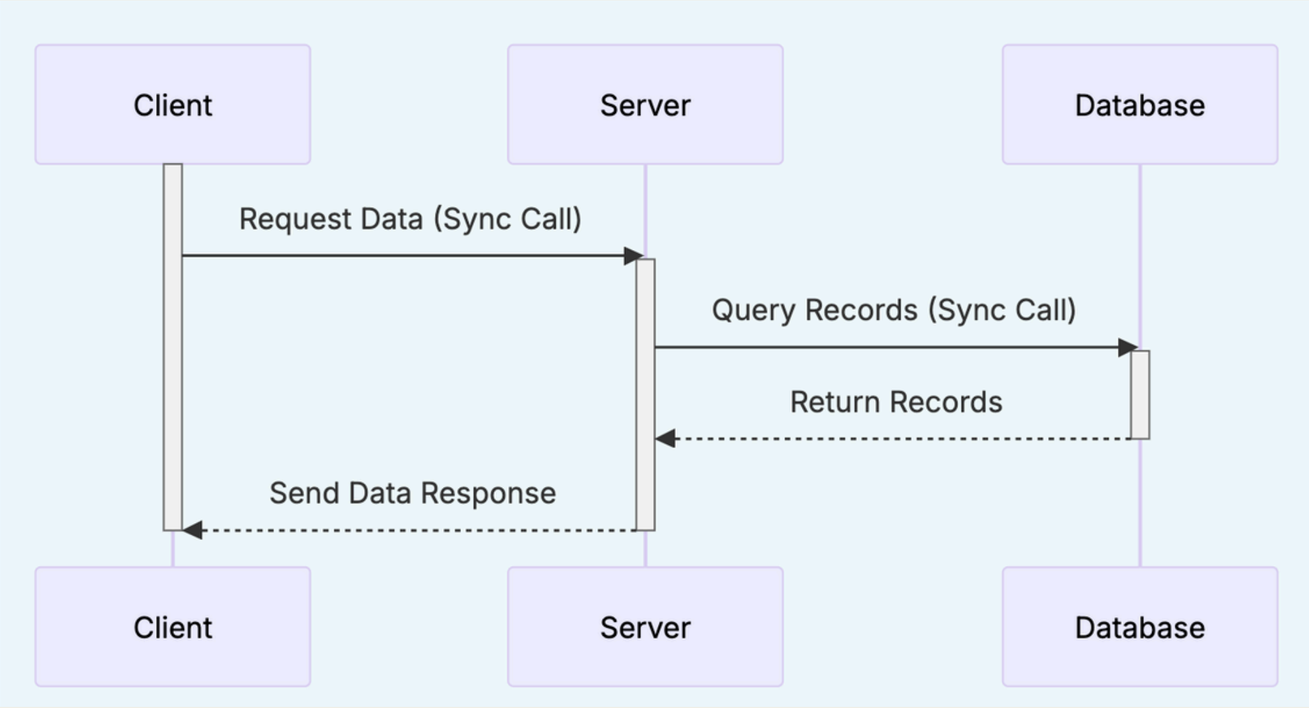


# SEQUENCE DIAGRAMS - AXES

- 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

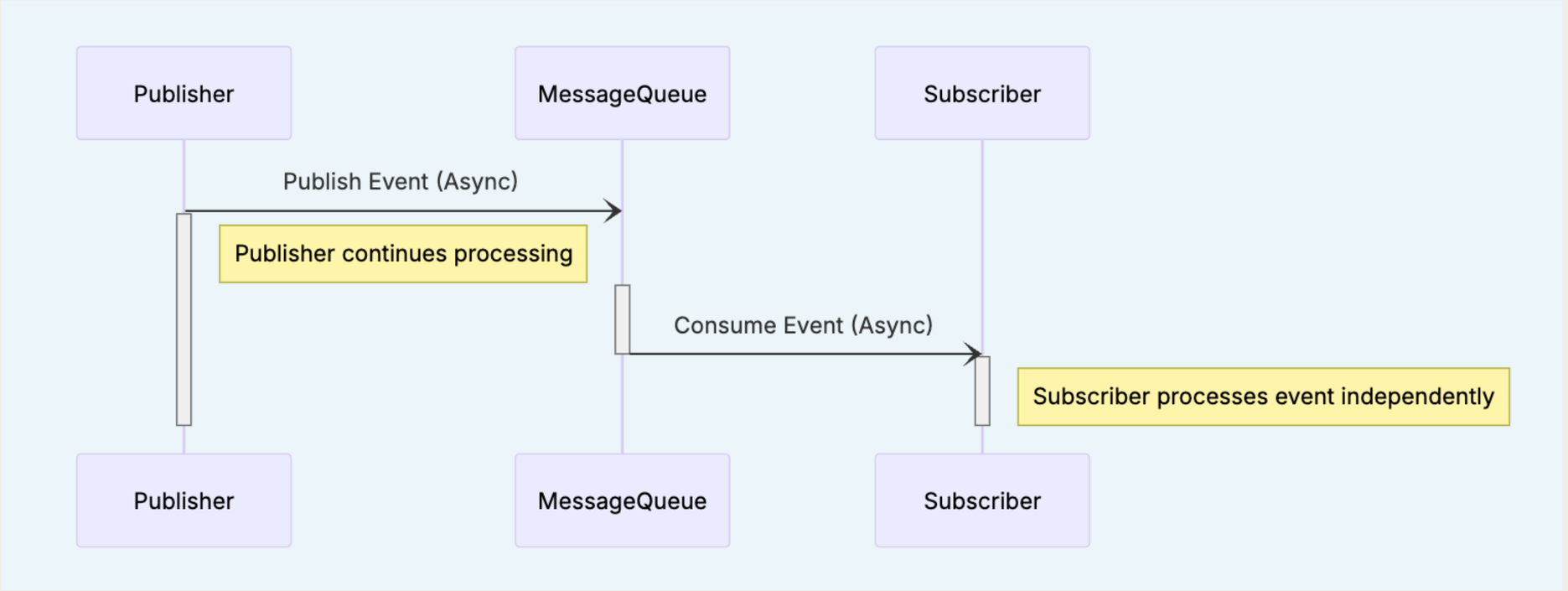
3. How does a Sequence Diagram represent synchronous vs. asynchronous communication? Provide one example each.

**Synchronous Communication**: The sender waits for the receiver to complete the operation and return a response before continuing its own execution.



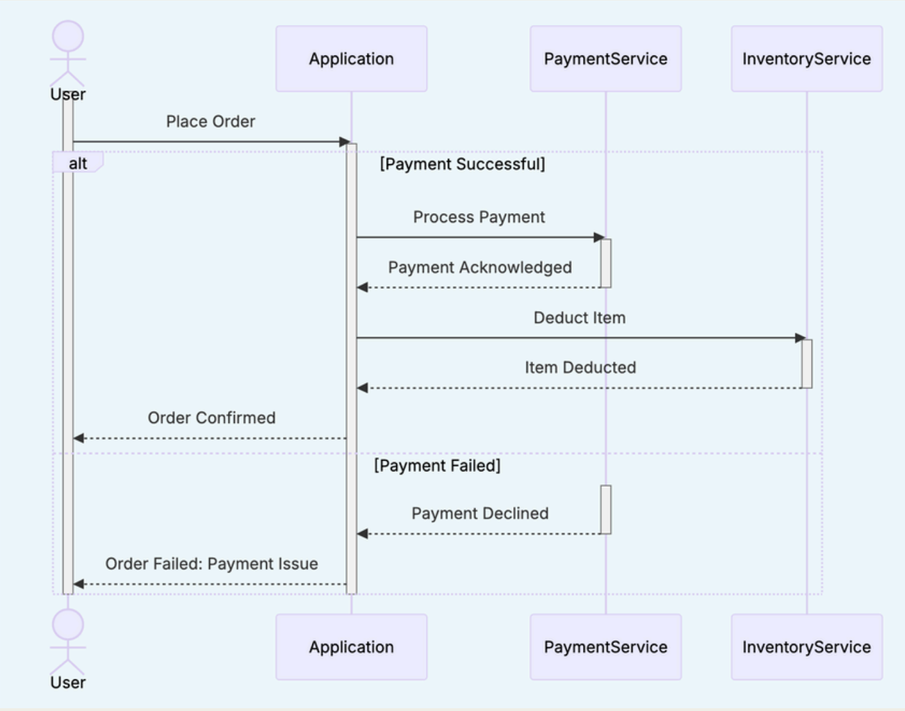
3. How does a Sequence Diagram represent synchronous vs. asynchronous communication? Provide one example each.

**Asynchronous Communication:** The sender does not wait for the receiver to complete the operation; it sends the message and continues its own execution immediately.



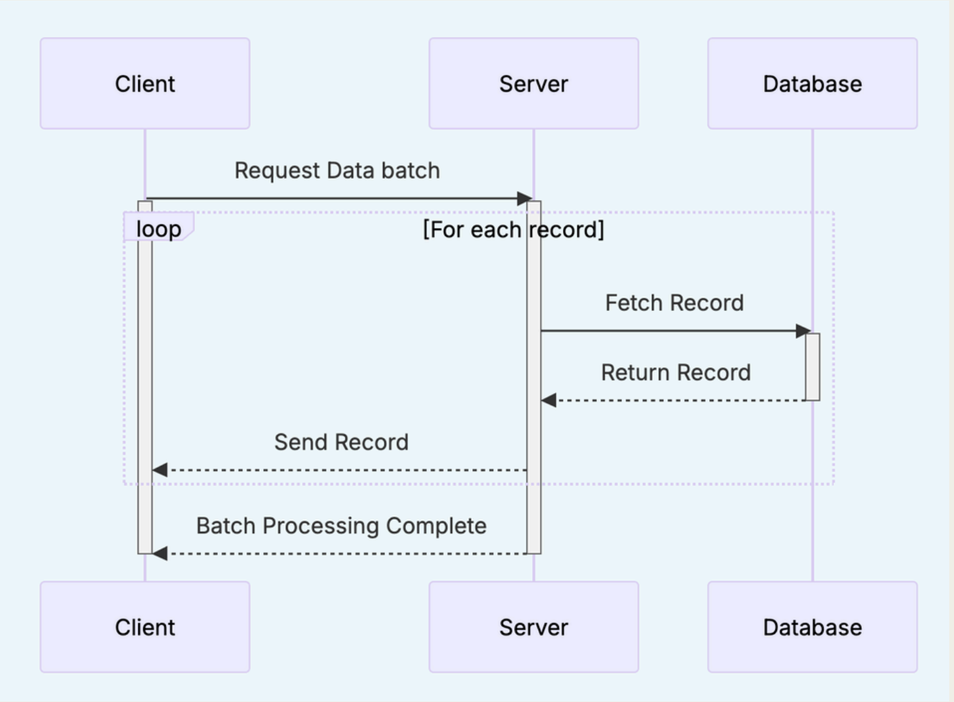
4. How would you model conditional behaviour and looping in a sequence diagram? Provide one example each.

**Conditional Behaviour (Alternatives)**: Modelled using an **alt (alternative) combined fragment**. Only the fragment whose condition is true is executed.



4. How would you model conditional behaviour and looping in a sequence diagram? Provide one example each.

**Looping**: Modelled using a loop **combined fragment**. The fragment has a guard condition [condition] that specifies when the loop continues.



# 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

6. Sequence diagrams are often criticised for becoming unreadable in complex systems. What are some techniques to manage this complexity? (see "Sequence Diagram Best Practices").

- **Refactor/Decompose**: Break down a large, complex sequence into smaller, more manageable diagrams, each representing a specific sub-process or use case.
    - **Example**: A high-level diagram might show WebApp communicating with OrderService, while a low-level diagram details how OrderService interacts with InventoryService and PaymentGateway.
- **Focus on a Single Use Case**: Each diagram should ideally focus on one specific scenario or path through a use case (e.g., "successful login," "failed login").

## SEQUENCE DIAGRAMS

6. Sequence diagrams are often criticised for becoming unreadable in complex systems. What are some techniques to manage this complexity? (see "Sequence Diagram Best Practices").

- **Fragments**: Effectively use combined fragments (alt, opt, loop, par) to group related messages and reduce visual clutter, even if they add some complexity themselves.
- **Use High-Level Messages**: Instead of showing every granular message, use higher-level messages that abstract away internal details of a component.
- **Use descriptive and consistent names for lifelines, messages, and guard conditions**: Ambiguous naming increases cognitive load.
- **Activation bars, spacing**: Ensure activation bars clearly show the active period. Arrange lifelines to minimize crossing lines.

# LABBY YOO

WOOO