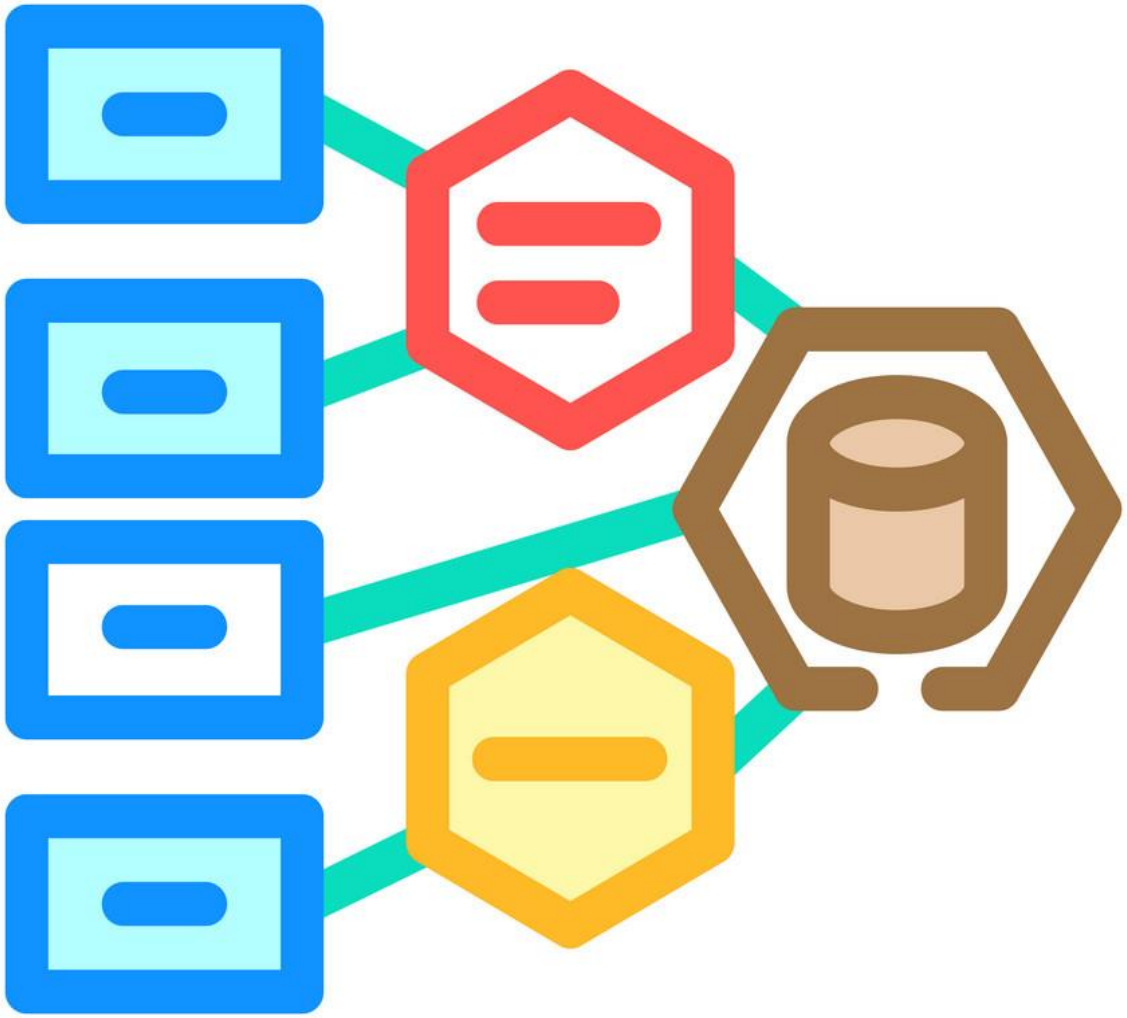
Software Design and architecture

Fall 2024

Dr. Natalia Chaudhry



Outline

- Component, collaboration and deployment diagram

Component diagram

A **component diagram** is used to visualize the structural relationships between components of a system.

It shows how different components (i.e., modules, services, or subsystems) interact and depend on each other.

Components: These are the main building blocks of your system. A component could represent a software module, a database, or a web service. In UML, a component is typically represented as a rectangle with two smaller rectangles at the top.

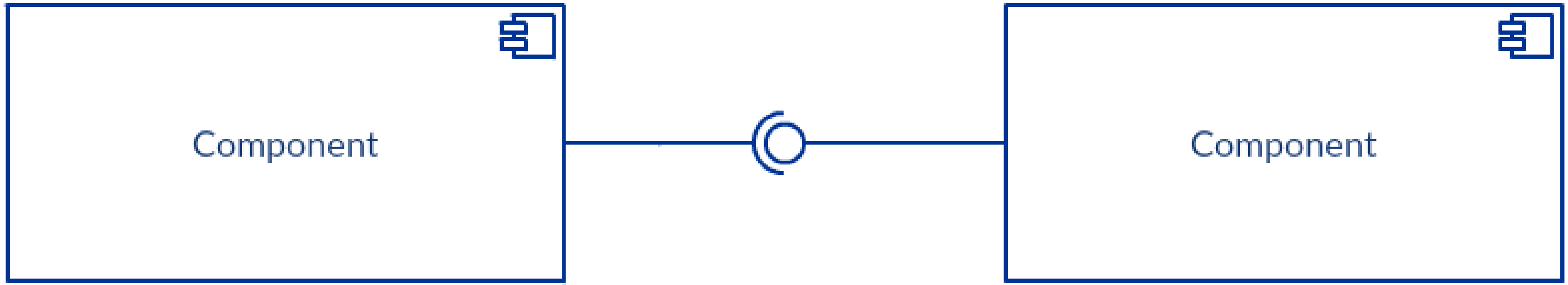
Interfaces: These represent how a component interacts with others. They can be provided (required by the system) or required (needed by the system).

Dependencies: Components often depend on other components or interfaces to function properly. Dependencies are shown as dashed arrows pointing from the dependent component to the dependency.

- Draw each component as a rectangle with its name written inside it.
- Use a smaller rectangle or a "lollipop" symbol to represent the interfaces offered or required by each component.

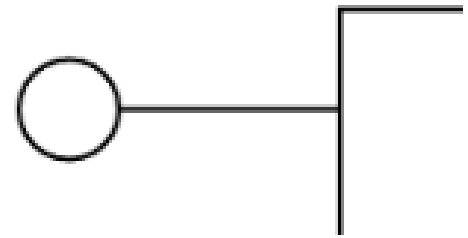
- **Dependencies**: Use a dashed arrow to show dependencies between components. The arrow points from the dependent component to the component it depends on.
- **Interfaces**: Use lines to connect components to the interfaces they implement or require. Provided interfaces are shown with a circle, and required interfaces use a half-circle symbol.

Provided Interface and the Required Interface



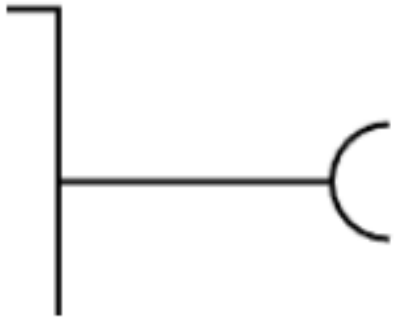
rfaces

- A provided interface shows that a component offers a service or functionality that other components can use.
- It is depicted by a lollipop symbol (a circle with a line), attached to the component.
- The component that provides the interface is responsible for implementing the service described by the interface.

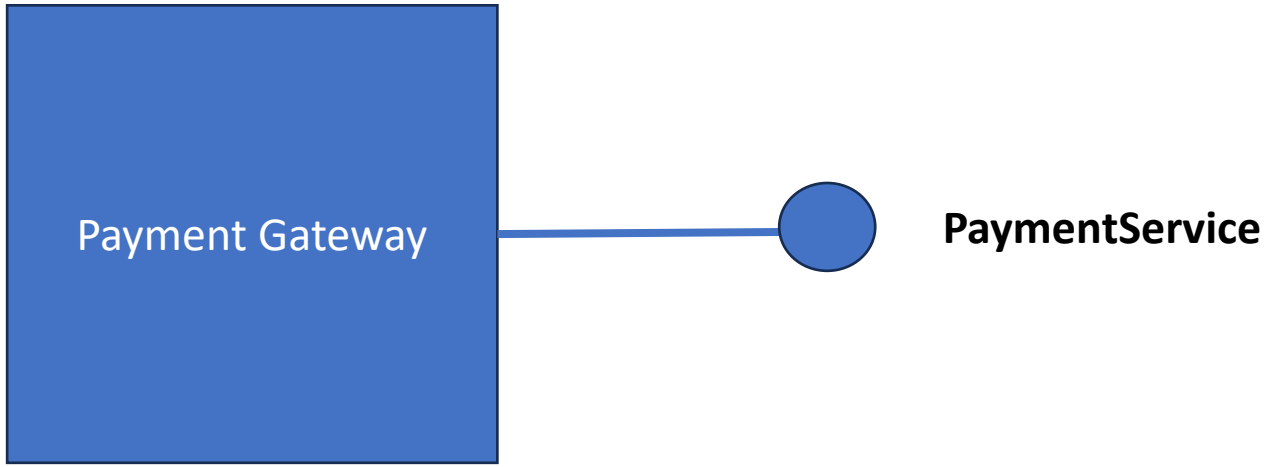


Required Interfaces

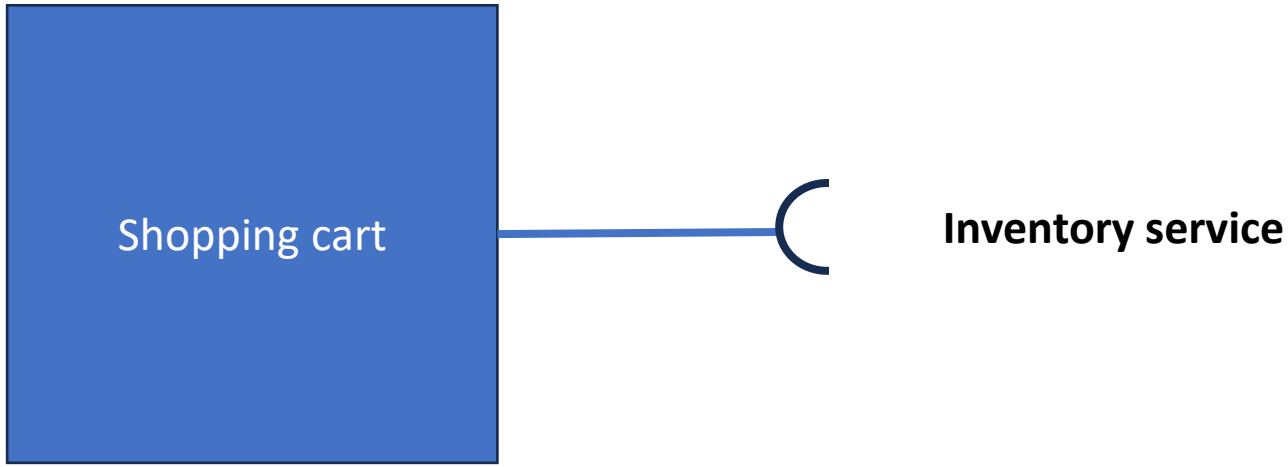
- A required interface represents a service or functionality that a component needs from other components in order to work.
- It is depicted as a half-circle (or "socket") symbol.
- The component with the required interface **depends** on some other component to provide the service.



Imagine a **Payment Gateway** component in an online store system. It might offer a **Payment Processing Service**, which other components (like the Checkout system) need. In this case, the **Payment Gateway** component has a **provided interface** representing its ability to process payments.

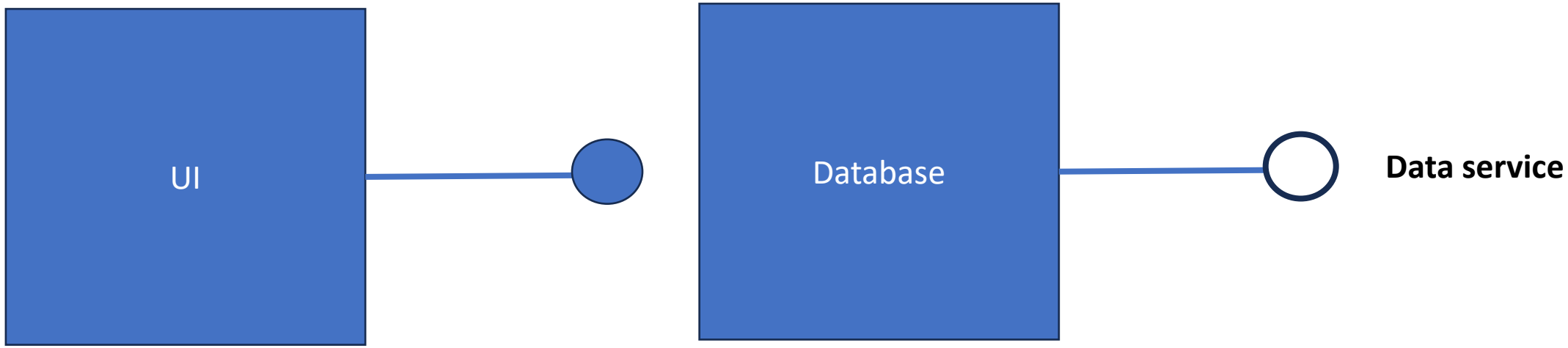


A **Shopping Cart** component might need access to an **Inventory Service** to check if items are available. The **Shopping Cart** does not provide this service itself but requires it from an external component (like an **Inventory** system).



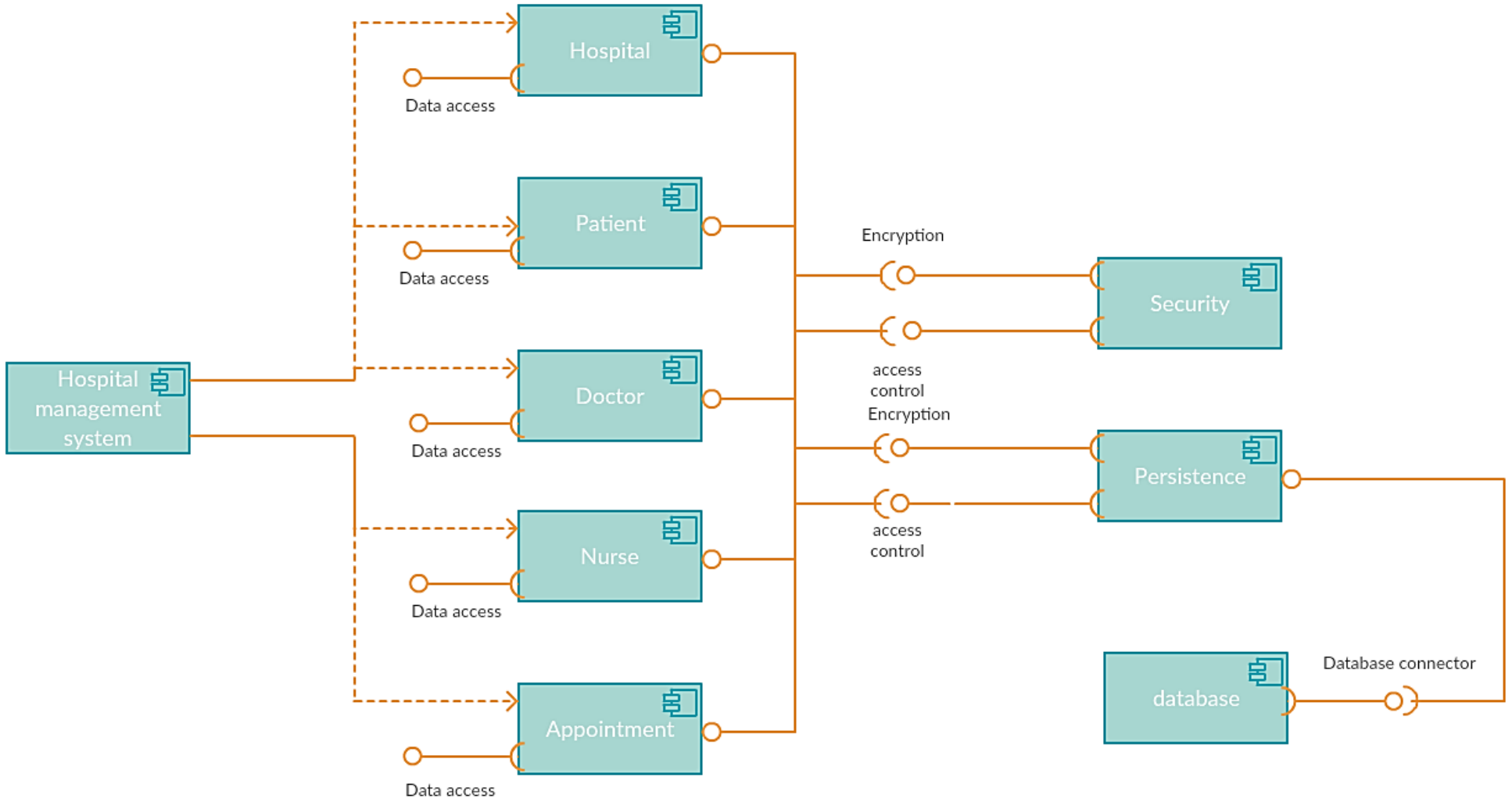
Interaction Between Provided and Required Interfaces:

- The interaction between components happens when the **required interface** of one component is connected to the **provided interface** of another component.
- This shows that one component uses the functionality provided by another.



In a system, a **User Interface (UI)** component may require a **DataService**, which is provided by a **Database** component.

COMPONENT DIAGRAM OF HOSPITAL MANAGEMENT SYSTEM



ToDo: Create a component diagram for campus management system (CMS)

•**User Interface (UI) Component:**
•**Login/Register Module:** Handles authentication and authorization for different types of users such as students, faculty, and staff.
•**Dashboard Module:** Provides a personalized dashboard for each type of user, showing relevant features, notifications, and shortcuts.

•**Student Management Component:**
•**Student Profile Module:** Manages student records, personal details, enrollment information, and academic history.
•**Course Registration Module:** Enables students to register for courses, view available classes, and drop/add subjects.

•**Faculty Management Component:**
•**Faculty Profile Module:** Manages the profiles of faculty members, including contact details, department, and qualifications.
•**Class Management Module:** Allows faculty to manage class schedules, post assignments, and enter grades.

•**Academic Management Component:**
•**Course Management Module:** Handles the creation and management of courses, schedules, syllabus, and course content.
•**Examination & Grading Module:** Manages exam schedules, grading systems, and results publication.

•**Financial Management Component:**
•**Fee Management Module:** Manages fee payment, tracks outstanding balances, and generates receipts.
•**Scholarship & Financial Aid Module:** Manages scholarships, grants, and other financial aid programs for students.

•**Library Management Component:**
•**Book Inventory Module:** Manages the inventory of books, periodicals, and digital resources.
•**Borrowing/Lending Module:** Tracks book loans, renewals, and overdue returns.

•**Hostel & Transport Management Component:**
•**Hostel Allocation Module:** Manages hostel rooms, allocations, and fees.
•**Transport Scheduling Module:** Handles transportation schedules and route management for students and staff.

•**Notification & Messaging Component:**
•**Email/SMS Notification Module:** Sends important updates, deadlines, and announcements to students, faculty, and staff.

•**Report Generation Component:**
•**Report Module:** Generates academic reports, fee reports, attendance records, and other customized reports.

•**Database Component:**
•Stores all data, including user profiles, academic records, fee details, course schedules, and library inventories.

•**Security & Audit Component:**
•**Access Control Module:** Manages role-based access to different components based on the user type (admin, faculty, student, etc.).
•**Audit Log Module:** Keeps track of system activities for auditing purposes.

•**UI Layer**: This interacts with the users (students, faculty, staff) and sends/receives information to/from the backend components.

•**Management Components**: Each functional area (Student, Faculty, Financial, Library, etc.) is represented by separate components. These components interact with the **Database** to store/retrieve information.

•**Notification Component**: Acts as a communication bridge to keep users updated on system changes, deadlines, and general announcements.

•**Security Components**: Ensure that only authorized users can access certain modules and logs all activities for security purposes.

Provided and Required Interfaces in the Campus Management System

1. User Interface (UI) Component:

- **Provided Interface**:
  - Offers user interaction capabilities such as login, dashboard access, and navigation between modules.
  - Provides session management for authenticated users.
- **Required Interface**:
  - Needs access to authentication services (from Security Component).
  - Requests data from components like Student Management, Faculty Management, etc., to populate the dashboard.

2. Student Management Component:

- **Provided Interface**:
  - Provides student data (profile, enrollment, academic records).
  - Allows interaction with the course registration and management functions.
- **Required Interface**:
  - Requires access to database services for storing/retrieving student records.
  - Depends on the Academic Management Component for course information and schedules.

3. Faculty Management Component:

- **Provided Interface**:
  - Offers data related to faculty profiles, classes, and assigned responsibilities.
- **Required Interface**:
  - Requires access to the Academic Management Component for scheduling courses and exams.
  - Requests access from the Database Component to store and retrieve faculty records.

4. Academic Management Component:

•**Provided Interface**:

- Provides functionality for managing courses, schedules, and exams.
- Generates reports like grade sheets and academic performance.

•**Required Interface**:

- Requires student enrollment data from the Student Management Component.
- Needs access to the Examination and Grading Module for grade entry and report generation.

5. Financial Management Component:

•**Provided Interface**:

- Offers fee management services, including fee collection, scholarship disbursement, and financial reporting.

•**Required Interface**:

- Needs student profile information (to calculate fees) from the Student Management Component.
- Requires communication with the Database Component to store financial transactions.

6. Library Management Component:

•**Provided Interface**:

- Provides book borrowing, lending, and inventory management services.

•**Required Interface**:

- Requires access to the Student and Faculty Management Components to authenticate users before lending.
- Needs to communicate with the Database Component for tracking inventory.

7. Hostel & Transport Management Component:

•**Provided Interface**:

- Provides services for hostel room allocation and transport scheduling.

•**Required Interface**:

- Requires student information (room assignment, hostel fees) from the Student Management Component.
- Depends on Financial Management Component for tracking hostel fees.

8. Notification & Messaging Component:

- **Provided Interface**:
  - Sends notifications (email/SMS) to students, faculty, and staff about system updates, deadlines, and announcements.
- **Required Interface**:
  - Requires access to other components to retrieve data for generating notifications (e.g., exam schedules from Academic Management, fee payment deadlines from Financial Management).

9. Report Generation Component:

- **Provided Interface**:
  - Offers report generation services for all system components (e.g., academic reports, fee reports).
- **Required Interface**:
  - Requires data access from all components (Student, Faculty, Financial, etc.) to generate accurate reports.

10. Database Component:

- **Provided Interface**:
  - Provides persistent data storage and retrieval services to all components.
- **Required Interface**:
  - None, as it serves as the backend for all components.

11. Security & Audit Component:

- **Provided Interface**:
  - Offers authentication and authorization services (e.g., login, role-based access control).
  - Provides audit logs for tracking user actions.
- **Required Interface**:
  - Requires user data from the UI component for authentication.
  - Needs access to the Database Component for storing and retrieving access logs.

Collaboration diagram

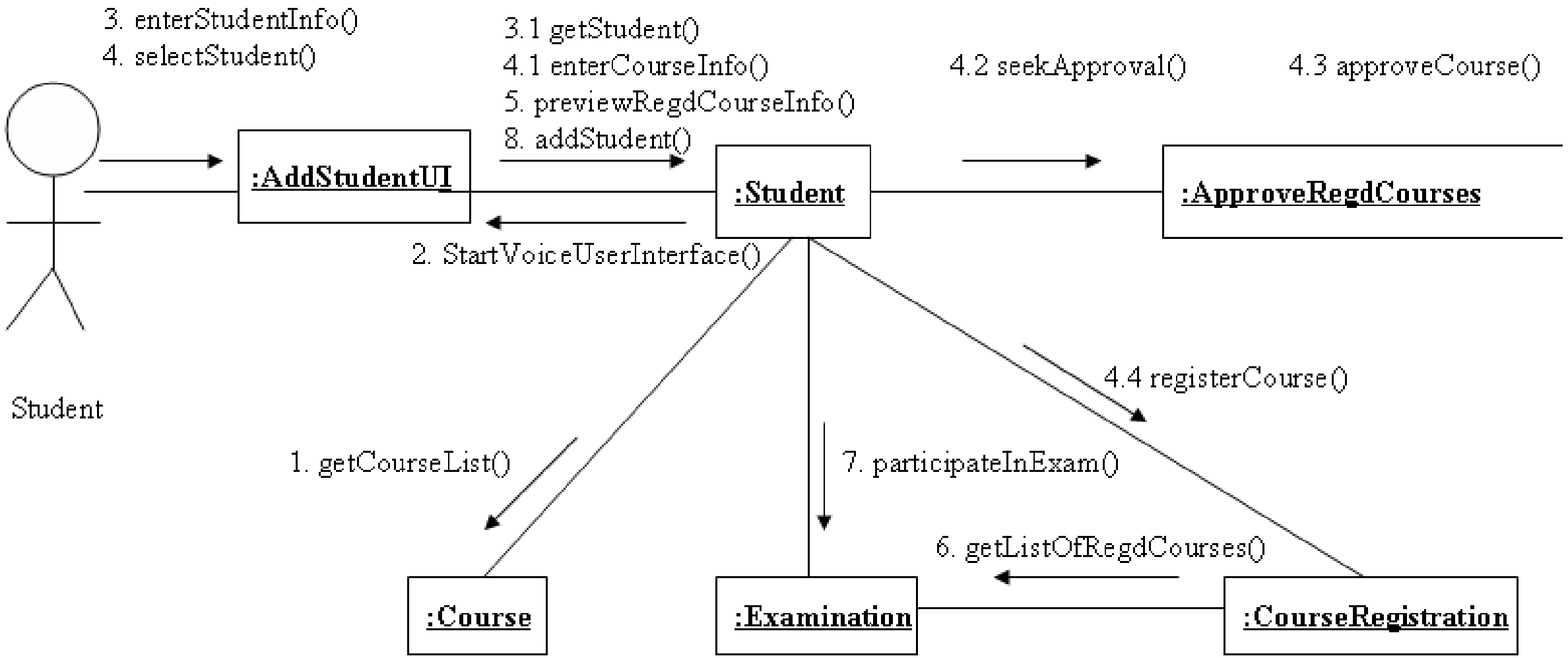
- It emphasizes the relationships and flow of messages between objects in a system, making it useful for visualizing real-time processes and interactions within a system.

Key elements of a Collaboration Diagram:

1.Objects/Actors: Represented as rectangles, each object plays a role in the interaction.

2.Links/Associations: Lines connecting the objects, showing their relationships or dependencies.

3.Messages: Arrows labeled with the message passed from one object to another, often numbered to indicate the sequence.


3. enterStudentInfo()
4. selectStudent()
3.1 getStudent()
4.1 enterCourseInfo()
5. previewRegdCourseInfo()
8. addStudent()
4.2 seekApproval()
4.3 approveCourse()
:AddStudentUI
:Student
:ApproveRegdCourses
2. StartVoiceUserInterface()
Student
4.4 registerCourse()
1. getCourseList()
7. participateInExam()
6. getListOfRegdCourses()
:Course
:Examination
:CourseRegistration

Collaboration Diagram for an E-commerce Purchase

Collaboration Diagram Elements:

1. **Customer** interacts with the system.
2. **Order System** handles customer orders.
3. **Payment Service** validates and processes payments.
4. **Inventory System** checks product availability.

collaboration diagram for weather forecast system

Key Components of the Weather Forecast System:

1. **User**: The person or system requesting the weather forecast.
2. **Forecasting System**: The core system that processes data and generates the forecast.
3. **Weather Data Provider**: External systems, such as satellite systems or weather data APIs, that provide raw weather data.
4. **Database**: Stores historical weather data and forecasts.
5. **Notification Service**: Optional, for sending alerts or forecasts to users

Steps and Interactions:

1. **User** sends a request for the weather forecast to the **Forecasting System**.
2. The **Forecasting System** sends a request to the **Weather Data Provider** (e.g., APIs, satellite systems) for real-time weather data.
3. The **Weather Data Provider** sends the real-time data back to the **Forecasting System**.
4. The **Forecasting System** processes the data using models, perhaps referencing the **Database** for historical data.
5. The **Forecasting System** sends the generated forecast to the **User**.
6. (Optional) If needed, the **Notification Service** can send weather alerts to the **User**.

Diagram Type	Focus/Emphasis	Best for	Key Elements
Sequence	Order of messages over time	Understanding the **sequence** of interactions	Lifelines, Messages, Time progression
Activity	Control flow of activities	Representing **workflows** and processes	Actions, Decisions, Control flow
Collaboration	Message exchange & structure	Visualizing **interactions** and communication patterns	Objects, Messages, Links

Deployment diagram

- It represents the architecture of a system in terms of its hardware components (nodes) and their relationships
- it’s useful for visualizing how software components are distributed across various hardware platforms.

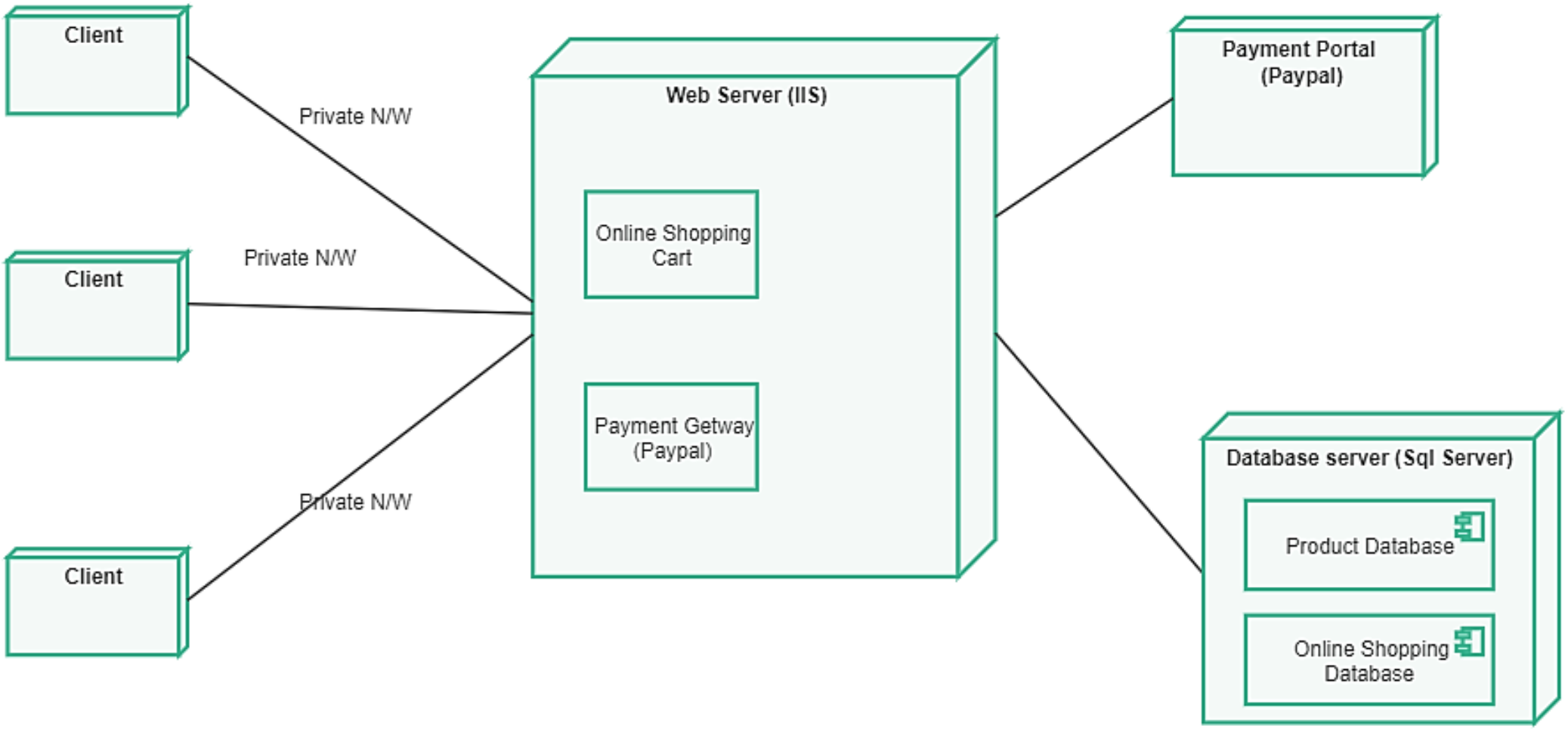
Key Elements of a Deployment Diagram:

1. **Nodes**: These represent the physical hardware or virtual machines. Nodes can be:
    1. **Device nodes**: Such as servers, computers, or devices.
    2. **Execution environments**: Such as a Java Virtual Machine (JVM) or application server.
2. **Artifacts**: These are the physical files or components that are deployed on the nodes. Artifacts can include:
    1. Executables
    2. Libraries
    3. Configuration files

3. Connections: These show the communication paths between nodes. They can represent:

1. Network connections (like TCP/IP)
2. Other forms of communication (like shared memory)

Deployment Diagram for Online Shopping



In some deployment diagrams, **software components** are represented as nodes to highlight how the **software environments** or **platforms** themselves function as critical elements of the deployment. This approach is commonly used when the software (or middleware) plays a key role in executing or hosting other components.

Reasons for Representing Software Components as Nodes

Software as an Execution Environment:

•Some software components (e.g., **virtual machines**, **containers**, **application servers**, **web servers**) provide the environment where the actual business logic runs. These are often abstracted as nodes because they behave like virtualized hardware for other components.

•For example, a **Java Virtual Machine (JVM)**, a **Docker container**, or a **Tomcat server** can be treated as nodes, since they host and execute other deployed artifacts.

Clarity in Complex Systems:

•In complex systems, treating major software platforms or middleware as nodes clarifies the deployment architecture. It helps to distinguish between:

- **Execution environments** (e.g., JVM, Python environment, etc.)
- **Physical hardware nodes** (e.g., servers or devices)
- **Artifacts** (e.g., applications, services, libraries)

•This is particularly useful when different parts of the system depend on specific versions or configurations of software environments.

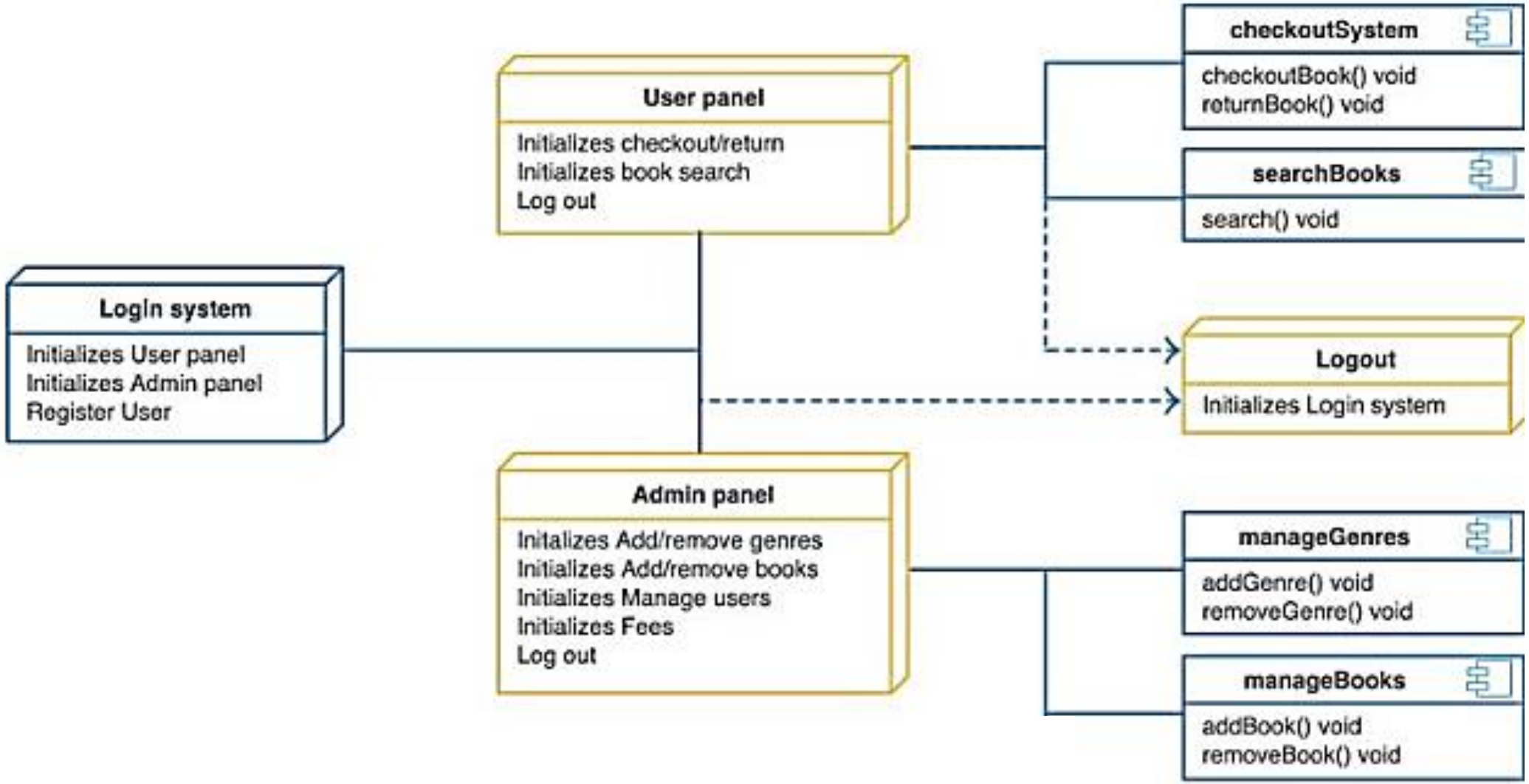
Virtualization and Cloud Systems:

•In cloud-based or containerized systems, the boundaries between hardware and software often blur. For example:

- **Virtual machines (VMs)** are software that act as virtual hardware.
- **Containers (Docker, Kubernetes)** are often treated as nodes because they create isolated software environments.
- Cloud services like **AWS Lambda** or **Google App Engine** can be abstracted as nodes, as they represent serverless platforms running your code.

Middleware or Platforms Hosting Multiple Applications:

- In cases where a single software platform or middleware hosts multiple applications (e.g., an **enterprise application server** or a **database management system**), that software platform itself becomes critical enough to be modeled as a node in the deployment diagram.
- This shows how different components interact with the platform and clarifies where specific components are deployed.


checkoutSystem
checkoutBook() void
returnBook() void
User panel
Initializes checkout/return
Initializes book search
Log out
searchBooks
search() void
Login system
Initializes User panel
Initializes Admin panel
Register User
Logout
Initializes Login system
Admin panel
Initalizes Add/remove genres
Initializes Add/remove books
Initializes Manage users
Initializes Fees
Log out
manageGenres
addGenre() void
removeGenre() void
manageBooks
addBook() void
removeBook() void

•**Admin Panel** and **Checkout Panel** often represent different functional modules or user interfaces of the system.

•By representing them as nodes, the diagram emphasizes that these are **separate interfaces** or **subsystems** with distinct responsibilities.

- **Admin Panel**: This interface is for librarians or system administrators to manage books, users, and library configurations.
- **Checkout Panel**: This interface is focused on user transactions like borrowing, returning, and checking out books.

By showing these components separately, the deployment diagram highlights that these panels may be hosted or deployed on different servers or virtual environments.

Separation of User Access and Permissions:

- In most LMS, the **admin panel** has more sensitive functions (e.g., user management, book inventory updates), while the **checkout panel** is designed for standard library users with more limited access.
- These panels may be deployed on different nodes for security purposes, ensuring that the more sensitive admin functions are isolated from regular user interactions.
- Treating them as separate nodes ensures that different security measures (firewall rules, authentication mechanisms, etc.) can be applied to each.

Scalability and Maintenance:

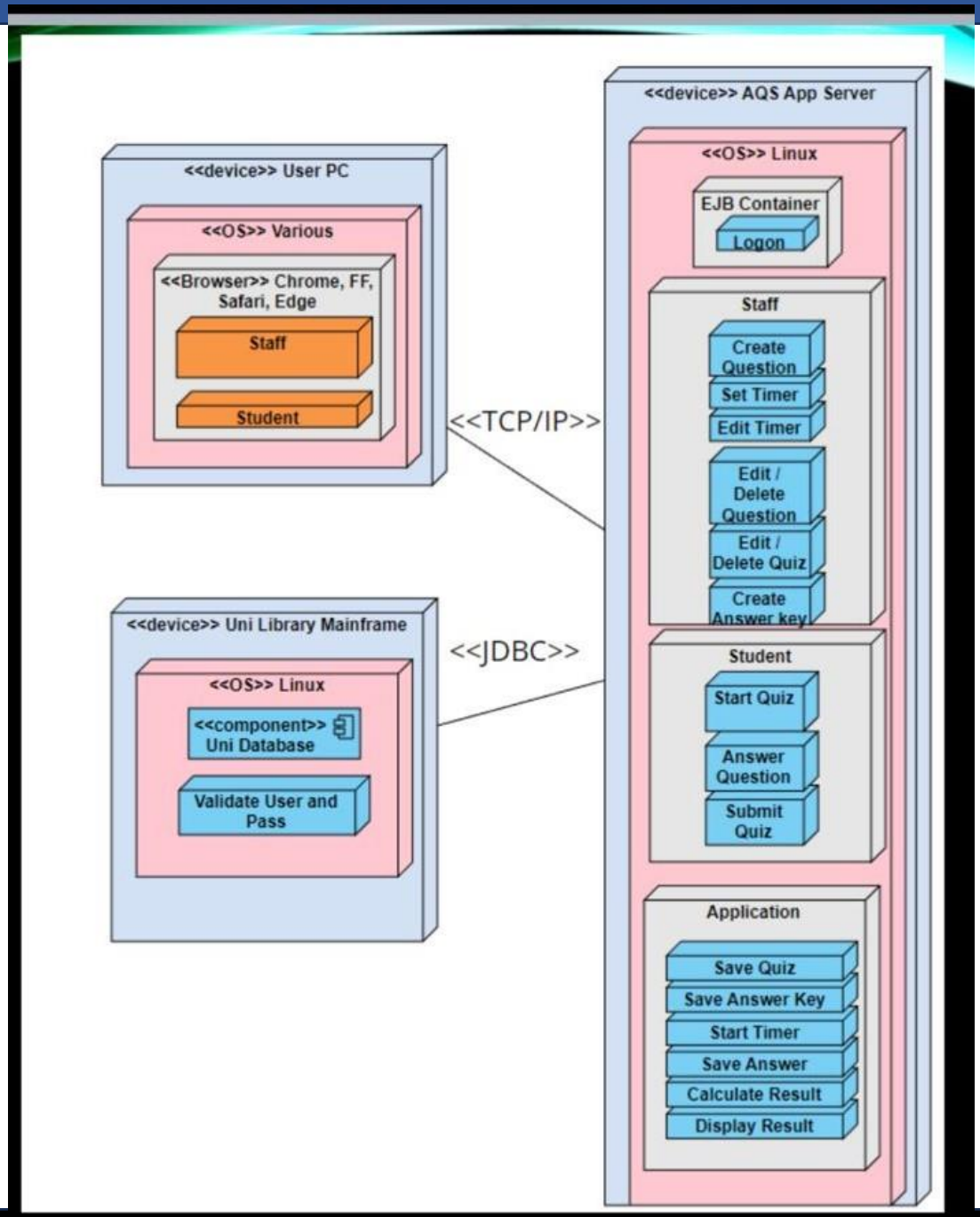
•**Scalability**: Each panel might be deployed on separate hardware or virtual machines (VMs) to handle different workloads. For example:

- **Admin Panel** could be accessed by fewer users (e.g., librarians) but may have more intensive tasks like generating reports or managing inventory.
- **Checkout Panel** might need to handle high traffic from students or library users.

•**Maintenance**: By separating the panels, each part of the system can be maintained or upgraded independently. For instance, updates to the admin panel won't affect users checking out books.

Clear Visual Representation of Subsystems:

- Representing **admin panel** and **checkout panel** as nodes in a deployment diagram provides a clear visual understanding of how the LMS is divided into subsystems or modules.
- This helps when communicating the deployment architecture to stakeholders, showing which parts of the system interact and how they are deployed.


<<device>> User PC
<<OS>> Various
<<Browser>> Chrome, FF, Safari, Edge
Staff
Student
<<TCP/IP>>
<<device>> Uni Library Mainframe
<<OS>> Linux
<<component>> Uni Database
Validate User and Pass
<<JDBC>>
<<device>> AQS App Server
<<OS>> Linux
EJB Container
Logon
Staff
Create Question
Set Timer
Edit Timer
Edit / Delete Question
Edit / Delete Quiz
Create Answer key
Student
Start Quiz
Answer Question
Submit Quiz
Application
Save Quiz
Save Answer Key
Start Timer
Save Answer
Calculate Result
Display Result

That’s it