UNIT-5

System Integration

Overview of System Integration

- **System integration** refers to the process of bringing together different subsystems or components into a single, cohesive system, ensuring that they function as one. The goal is to enable the system to work seamlessly and efficiently, enhancing functionality and improving overall performance. It is essential in various industries like IT, telecommunications, manufacturing, and business services, where complex systems often require the integration of different software, hardware, and networks.

- In web application development, **system integration** refers to the process of combining the **front end** (client side) and **back end** (server side) to create a cohesive system where data is exchanged and functionalities are properly synchronized.

Key aspects for Web application system Integartion

- 1. **Front End (Client Side)**
- 2. **Back End (Server Side)**
- 3. **System Integration Approach**

1. Front End (Client Side)

- The front end is the part of the application that users interact with directly. It includes:
- **Technologies:** HTML, CSS, JavaScript, and frontend frameworks/libraries like **Angular**, **React**, or **Vue.js**.
- **User Interface (UI):** Responsible for presenting information and capturing user inputs.
- **Client-Side Logic:** Handles user interactions, data validation, and rendering UI updates.
- **APIs and Services:** Makes asynchronous requests to the back end to retrieve or send data.

2. Back End (Server Side)

- The back end handles the core logic of the application, data management, and system operations. It includes:
- **Technologies:** Python (Django, Flask), Java (Spring Boot), Node.js (Express), Ruby (Rails), etc.
- **Database:** Handles data storage and retrieval (e.g., MySQL, PostgreSQL, MongoDB, or H2 for local development).
- **Server-Side Logic:** Manages business logic, authentication, authorization, and communicates with the front end via APIs.
- **APIs (REST, GraphQL):** Provide endpoints for front-end applications to interact with. REST APIs are widely used due to their simplicity and flexibility.

3. System Integration Approach

The integration between front end and back end can be categorized into a few patterns:

a. RESTful API-Based Integration

- **Structure:** Front end communicates with the back end using HTTP requests (GET, POST, PUT, DELETE) to RESTful API endpoints.
- **Data Exchange:** Typically done in JSON or XML format.
- **Tools:**
  - Front end uses **HTTP clients** like Axios, Fetch, or Angular’s HttpClient.
  - Back end uses **API frameworks** like Django (DRF), Flask, or Spring Boot to serve requests.
- **Example:** The front end might request a list of products from /api/products, and the back end would respond with JSON data.

b. **GraphQL based Integration**

- It is a query language for APIs and a runtime for executing those queries against your data. Developed by Facebook in 2012 and released publicly in 2015, GraphQL provides a more flexible and efficient way to interact with APIs compared to traditional REST approaches.

Key Components of System Integration

- a. Subsystems
- b. Middleware
- c. API (Application Programming Interface)
- d. Data Integration Tools

Subsystems

- These are the individual components, tools, modules, or applications that perform specific functions within an organization.
- Example: Softwares like Angular JS, Spring boot, H2 DB, PostGreDB, A Customer Relationship Management (CRM) system, an Enterprise Resource Planning (ERP) system, or a Human Resource Management (HRM) system.

Middleware

- Middleware is the software layer that facilitates communication and data management between subsystems. It acts as a "glue" to connect diverse applications.
- Example: Spring boot application layer, Hibernate (ORM Middleware).

API (Application Programming Interface)

- APIs allow subsystems to communicate with each other by defining how they can interact, enabling data exchange and functional integration.
- Example: REST APIs that let a frontend application communicate with the backend server or SOAP-based web services.

Data Integration Tools

- These tools allow data to be shared and synchronized between systems, ensuring consistency and eliminating redundancy.
- Example: ETL (Extract, Transform, Load) tools like **hibernate, Spring Data JPA** for managing data flow.
    - **Hibernate:** Acts as the data integration tool by managing the data flow between the object-oriented model in Java and the relational database schema in H2. It automatically handles the conversion of Java objects to database rows (and vice versa), ensuring that data remains consistent and synchronized.
    - **Spring Data JPA:** Built on top of Hibernate, Spring Data JPA provides repositories and data access layers that simplify querying and manipulating the H2 or other database. This tool abstracts database operations, allowing easy integration between the data source and business logic.

System Integration Process

- a. Requirements Gathering
- b. System Design and Planning
- c. Implementation
- d. Testing and Validation
- e. Deployment and Monitoring
- f. Maintenance and Support

Web Services

- Web services are a method of communication between two electronic devices or applications over a network, typically using the internet. They allow different systems or applications to interact with each other, exchanging data and performing functions remotely. Web services typically follow standard protocols like HTTP/HTTPS and use specific formats such as XML or JSON for data transmission.

There are two main types of web services:

1. SOAP (Simple Object Access Protocol) Web Services:

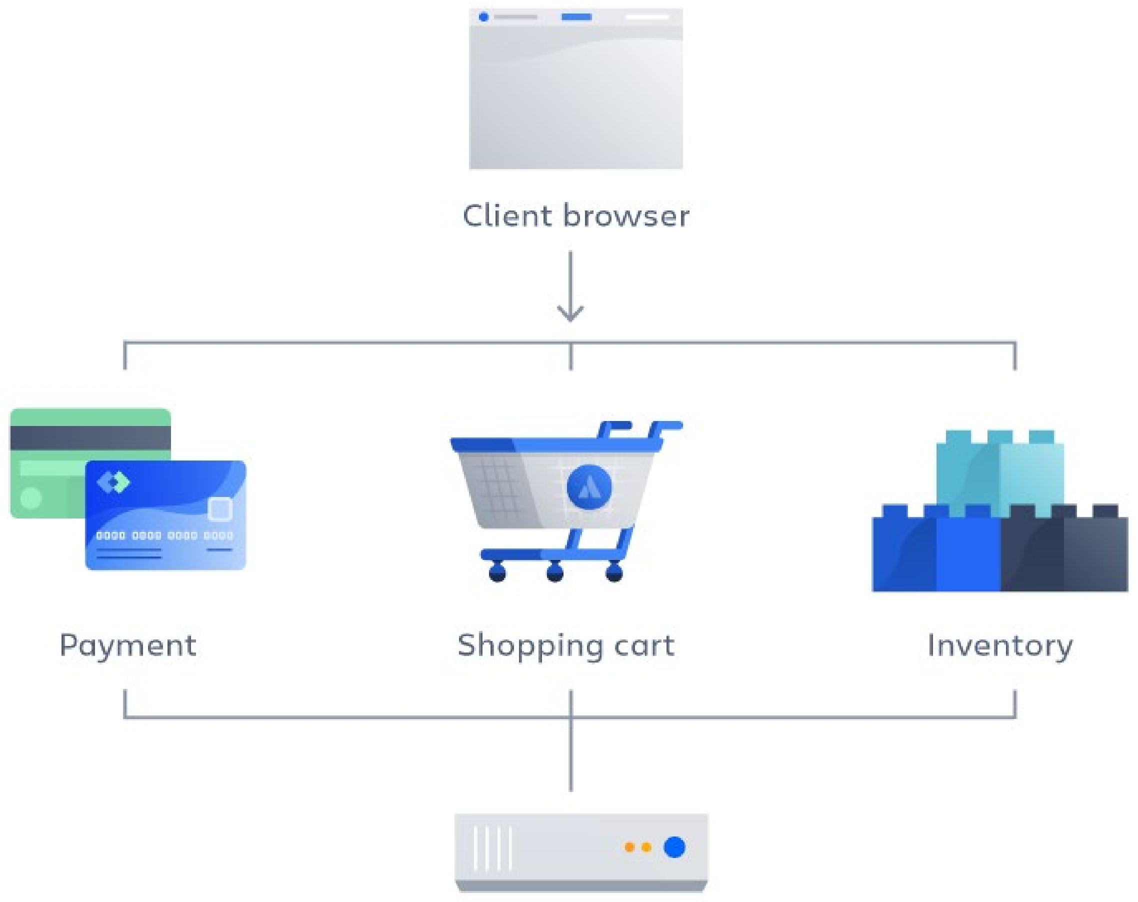
- **Protocol-based**: SOAP web services rely on a well-defined protocol that provides a set of rules and structure for requests and responses.
- **XML-based**: SOAP messages are formatted using XML.
- **Platform and Language Independent**: SOAP services can be consumed on any platform or language that supports the protocol.
- **WS-Security**: SOAP supports advanced security features like encryption and digital signatures.
- **Heavier and More Complex**: SOAP services often have more overhead due to the protocol structure and XML processing.

- 2. **REST (Representational State Transfer) Web Services**:
- Resource-based: REST services treat everything as a resource, each with a unique URL (e.g., /users, /products)
- .Uses HTTP Methods: REST services leverage standard HTTP methods like GET, POST, PUT, DELETE for communication
- .Lightweight and Easy to Use: REST uses simple formats like JSON (or XML) for data transfer and is less resource-intensive
- Stateless: Each request from the client to the server must contain all the information needed to understand and process the request.
- Popular for Web APIs: REST is widely used for building modern web APIs due to its simplicity and flexibility.

Monolithic Web Services

- A **monolithic** architecture means that the entire application is built as a single, unified unit where all the components (UI, business logic, data access, etc.) are tightly coupled and run as a single service.

Monolithic architecture



Features of monolithic web services:

- **Single Codebase**: The entire application is built and deployed as a single unit.
- **Tightly Coupled Components**: Different modules (user management, billing, order processing, etc.) are part of the same process and share resources like memory, databases, etc.
- **Shared Database**: Usually, a single database is used to store the application's data.
- **Centralized Deployment**: The application is deployed and scaled as a whole. Scaling requires deploying another instance of the entire application.
- **Easier Development Initially**: Development is straightforward as there is only one service to manage, making local testing and deployment simpler in the early stages.

Advantages of **monolithic web services:**

- Simple to Develop: Easier to set up and get started since everything is contained in a single unit.
- Easier Testing and Debugging: Since the application is one unit, testing and debugging can be done within the single deployment.
- Centralized Management: Easier to manage in small applications, as everything is in one place

Advantages of **monolithic web services:**

- **Difficult to Scale**: Scaling means replicating the entire application, which can be inefficient if only one part of the application needs more resources.
- **Tight Coupling**: Changes in one module may affect others, leading to potential downtime or bugs during deployment.
- **Longer Deployments**: As the application grows, deployments can become slower, and even a small change requires redeploying the entire application.
- **Limited Agility**: Difficult to adopt new technologies in specific parts of the application, as all components share the same tech stack.

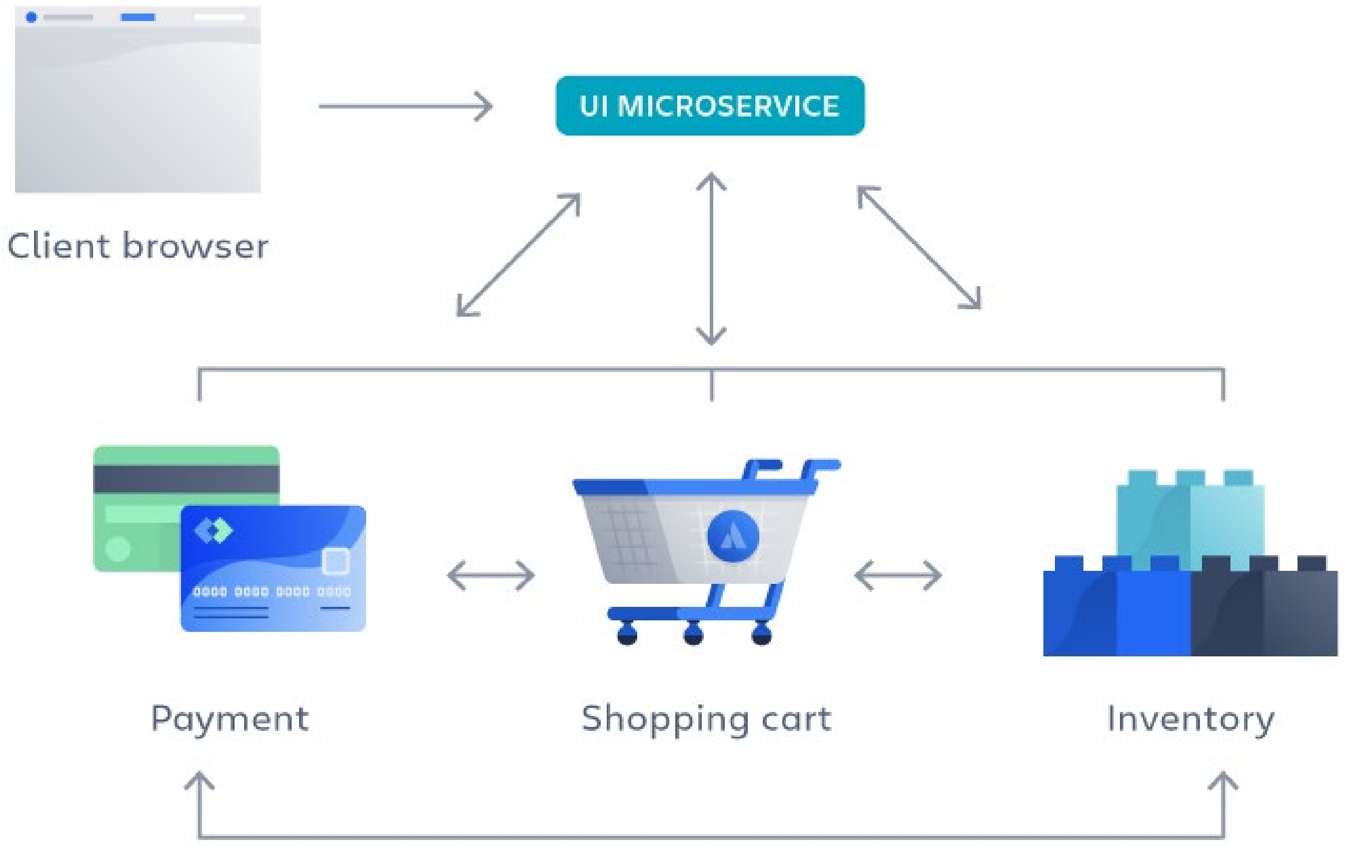
Example of monolithic web services :

- A large e-commerce application where all features (user management, product catalog, payment, etc.) are bundled together and run as one service.

Microservices- Web Services

- A **microservices** architecture involves breaking down the application into smaller, independently deployable services, where each service focuses on a specific functionality (e.g., order processing, authentication, etc.).

Microservice architecture



Features of micro services:

- **Decoupled Services**: Each service focuses on a specific functionality (like order management, inventory, user authentication) and is independently deployable.
- **Independent Databases**: Microservices often have their own databases, ensuring loose coupling.
- **Scalable Services**: Each service can be scaled independently based on its own resource needs.
- **Technology Diversity**: Each service can be developed using different technologies, frameworks, or languages depending on its specific needs.
- **Fault Isolation**: Issues in one service are less likely to impact the entire system since each service runs independently.

Advantages of Micro-Services

- Scalability: Individual services can be scaled independently, allowing better resource utilization.
- Flexibility: Different services can be built using different technologies best suited for their purpose.
- Resilience: Failures in one microservice don’t typically bring down the entire system.
- Faster Development Cycles: Teams can work independently on different services, improving development speed and agility.
- Continuous Deployment: Each microservice can be deployed independently, enabling continuous updates without affecting the whole system.

DisAdvantages of Micro-Services

- Increased Complexity: Managing many microservices can introduce operational complexity, especially in areas like communication, security, and monitoring.
- Inter-Service Communication: Services need to communicate over the network, which can introduce latency and failure points (e.g., API calls between services).
- Data Consistency: Managing distributed data across services can be challenging.
- More Difficult Testing: Integration testing in microservices can be more complex due to dependencies between services.

REST API

- A **REST API** (Representational State Transfer Application Programming Interface) is an architectural style for building web services that allows interaction with resources using HTTP methods like GET, POST, PUT, and DELETE. It is stateless, lightweight, and typically returns data in formats like JSON or XML.

- **SOAP** is a protocol with strict standards that uses XML for message formatting and supports stateful operations, while **REST** is an architectural style that is more flexible, lightweight, and typically uses JSON for data exchange, focusing on stateless operations.
- SOAP emphasizes security and transactions, while REST is preferred for performance and scalability in web services.

Key characteristics of a REST API

- Stateless: Each request from a client to a server must contain all the necessary information, and the server does not store session information.
- Client-Server Architecture: The client (frontend) and server (backend) operate independently, with the server handling the business logic.
- Cacheable: Responses can be cached to improve performance.
- Uniform Interface: Resources are identified using URIs, and operations are performed using standard HTTP methods.

Example

- We discussed the REST API example to insert students details and display student details (Refer Unit-4 ppt).

Hibernate

- **Hibernate** is an open-source Object-Relational Mapping (ORM) framework for Java that simplifies database interactions by allowing developers to work with Java objects instead of SQL queries. It provides a way to map Java classes to database tables and Java data types to SQL data types, facilitating database operations in a more object-oriented manner.

Features of Hibernate

- **Object-Relational Mapping (ORM):**
  - Hibernate maps Java objects to database tables, allowing developers to interact with the database using Java objects without dealing with the complexities of SQL.
- **Caching**:
  - Hibernate supports caching to improve performance. It has a first-level cache (session cache) and optional second-level cache (shared across sessions), reducing database access.
- **Automatic Schema Generation**:
  - Hibernate can automatically generate database schemas based on the mapping class or annotations, simplifying database setup.
- **Support for Various Databases**:
  - Hibernate can work with multiple databases (e.g., H2,MySQL, PostgreSQL, Oracle) through JDBC, providing flexibility in database choice.

Benefits of Using Hibernate

- Reduced Boilerplate Code: Developers can write less code to manage database interactions, focusing more on business logic.
- Portability: Hibernate is database-agnostic, allowing applications to switch databases with minimal changes in code.
- Improved Productivity: By abstracting database operations, Hibernate accelerates development and reduces the complexity of database programming.
- Integration: It easily integrates with other Java frameworks, such as Spring, making it a popular choice in enterprise applications.

Example

- We discussed the REST API example with hibernate to insert students details and display student details (Refer Unit-4 ppt).

Practice (for PUT request)

- Insert student details (rno,name age, dept) , display all student details and update the student details based on rno using spring boot RESTAPI with h2 database

Mini Project

- *Web application with AngularJS and Spring boot – Assignment*

    - Presentation and report

Presentation (10 Marks):

PPT presentation with demo

Report submission Hard copy(5 Marks):

Refer the given format → *https://docs.google.com/document/d/1LuXIG5300myLGWIeTqd4mDbqN1F0mYUf/edit?usp=sharing&ouid=102428071383149203176&rtpof=true&sd=true*