VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
(An Autonomous Institution)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

B.Tech(CSE) IV Year / I Semester (R22)

Lecture Notes

Name of the Faculty	**KISHORE K**
Department	**CSE(Data Science)**
Year & Semester	**B.Tech-IV & I Sem**
Subject Name	**FULL STACK DEVELOPMENT (Professional Elective - III)**

DEPARTMENT OF CSE (Data Science)

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
An Autonomous Institution
AZIZ NAGAR, C B POST, HYDERABAD-500075
2025-2026

Building Blocks of Full Stack Development PART-1

Introduction, Front-End Technologies, Back-End Technologies, MVC, Web Services, Communication between Front-End and Back-End,

***JSON** - Syntax, Parsing and Serialization.*

Full Stack Development refers to the process of developing both the **Front-End (Client-Side)** and **Back-End (Server-Side)** of a Web Application.

A Full-Stack Developer is proficient in both **Front-End** (Client-Side) and **Back-End** (Server-Side) Technologies. This holistic understanding allows them to:

- **Design and Implement** User Interfaces **(UI).**
- Develop **Server-Side Logic** and APIs.
- **Manage Databases** for data storage and retrieval.
- **Understand and Optimize** the flow of data between the client and server.
- **Participate in all Phases** of the Software Development Lifecycle **(SDLC),** from conceptual to deployment.

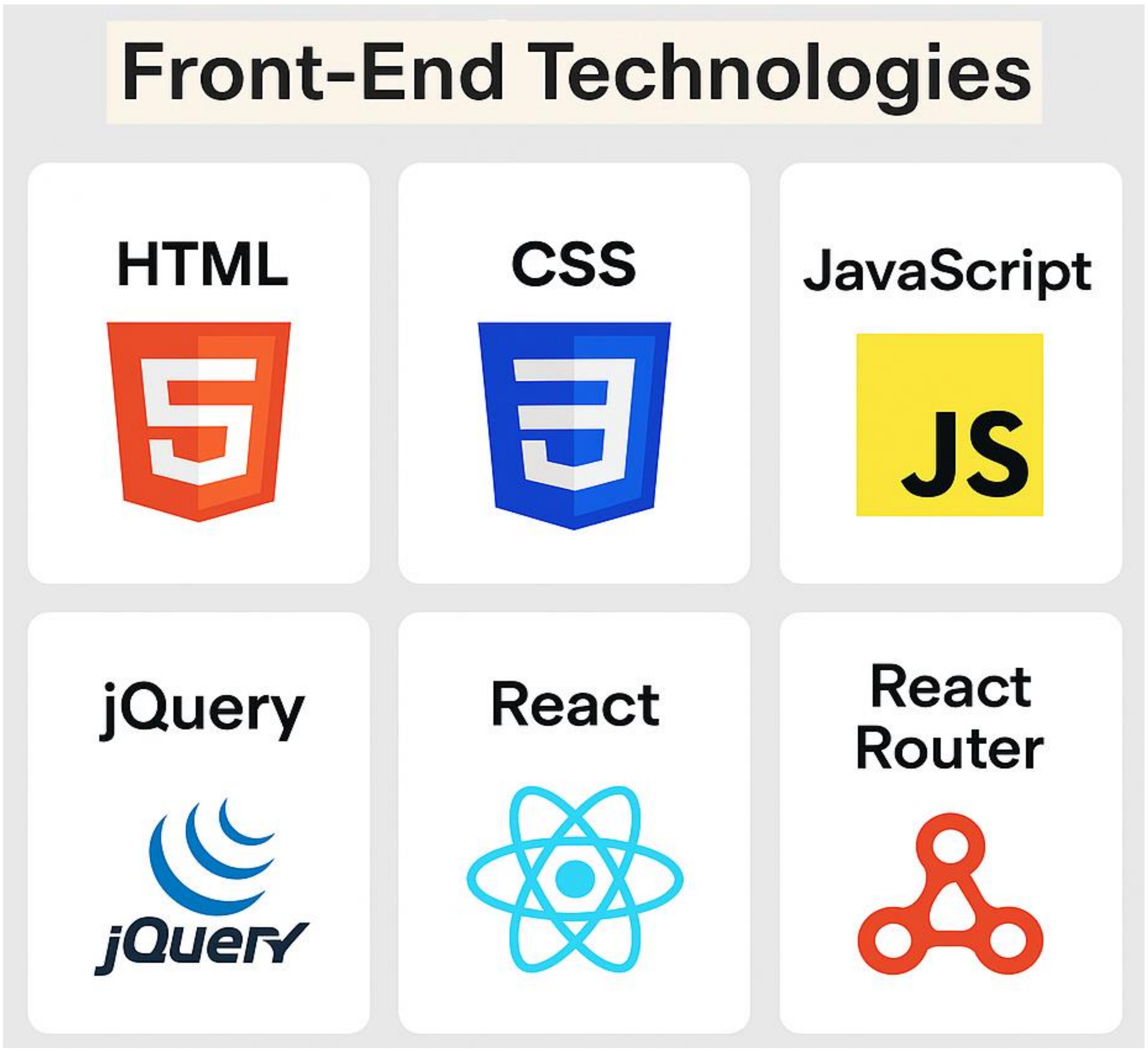
The term **Stack** refers to a set of technologies used together to build a complete application.

It typically includes:

Layer	Role	Example Technologies
Front End	What the user sees & interacts with	HTML, CSS, JavaScript, JQuery, React, React Router
Back End	Handles logic, API requests, processing	Java (Servlets, JSP, Spring), Node.js, Python (Django), XML, JSON & REST
Database	Stores and retrieves data	MySQL, MongoDB, PostgreSQL
Versioning & Deployment	Collaboration & hosting	Git, GitHub, Docker, AWS

Front-End Technologies:

The Front-End is the **Client-Side** of the application – what the user sees and directly interacts with in their web browser.



1. **HTML (HyperText Markup Language):**
   - **Purpose:** Provides the **structure and content** of web pages. It's the skeleton upon which everything else is built.
   - **Role:** Defines headings, paragraphs, images, links, forms, and the overall layout structure using tags like <div>, <p>, <a>, <img>, etc.

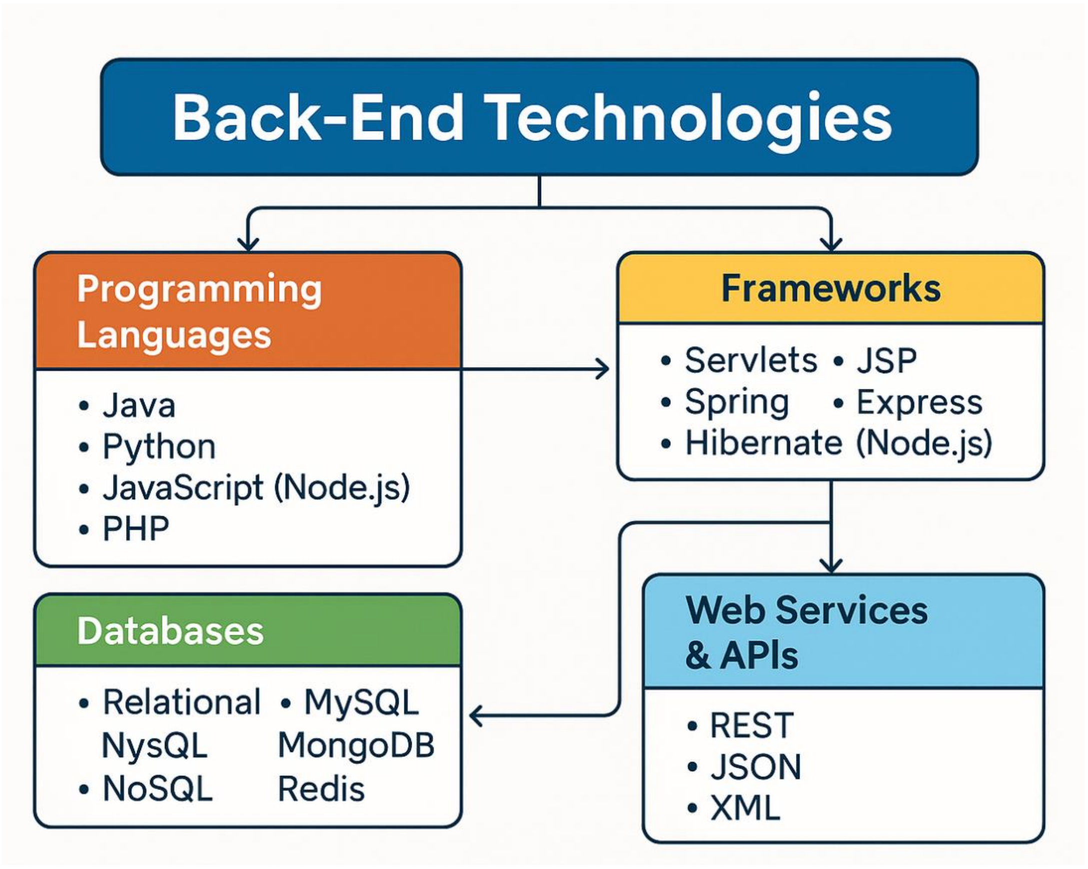
2. **CSS (Cascading Style Sheets):**
    - **Purpose:** Controls the **visual presentation and styling** of HTML elements. It brings the web page to life.
    - **Role:** Defines colors, fonts, spacing, layout (using Flexbox or Grid), animations, and responsive design for different screen sizes.

3. **JavaScript:**
    - **Purpose:** Adds **interactivity and dynamic behavior** to web pages. It's the Client-Side Scripting Language of the web browser.
    - **Role:**
        - **DOM Manipulation:** Modifying the HTML structure and content dynamically.
        - **Event Handling:** Responding to user actions (clicks, key presses, form submissions).
        - **Asynchronous Operations:** Making requests to the server (APIs) without reloading the entire page (AJAX, Fetch API).
        - **Client-Side Logic:** Performing validations, calculations, and managing UI state.

4. **Front-End Frameworks/Libraries (jQuery, React, React Router):**
    - **Purpose:** Provide structured ways and reusable components to build complex, maintainable, and scalable user interfaces. They abstract away much of the raw DOM manipulation and simplify state management.
    - **Role:**
        - **jQuery:** (Historically very popular) A JavaScript library that simplifies DOM traversal, manipulation, event handling, and AJAX. It smooths over cross-browser inconsistencies.
        - **React:** A declarative, component-based library for building user interfaces. It efficiently updates and renders components when data changes, making it ideal for Single Page Applications (SPAs).
        - **React Router:** A standard library for declarative routing in React applications. It allows you to define different views (components) that correspond to different URLs, enabling client-side navigation without full page reloads.

Back-End Technologies:

The **Back-End** (or Server-Side) handles the ***"behind-the-scenes"*** functionality, including business logic, data storage, security, and serving data to the front-end.



1. Servlets & JSPs (Java-specific):

- **Purpose:** Traditional Java technologies for building Dynamic Web Applications.
- **Role:**
  - **Servlets:** Java classes that extend server capabilities to handle HTTP requests and generate responses. They act as controllers.
  - **JSPs (Java Server Pages):** Allow Java code to be embedded directly within HTML-like pages, primarily for generating dynamic HTML content on the server side. In modern architectures with a React front-end, JSPs are less common for UI rendering but might be used for initial page setup or specific server-side templating.

2. **Server-Side Languages/Runtimes (e.g., Java with Spring, Node.js, Python):**

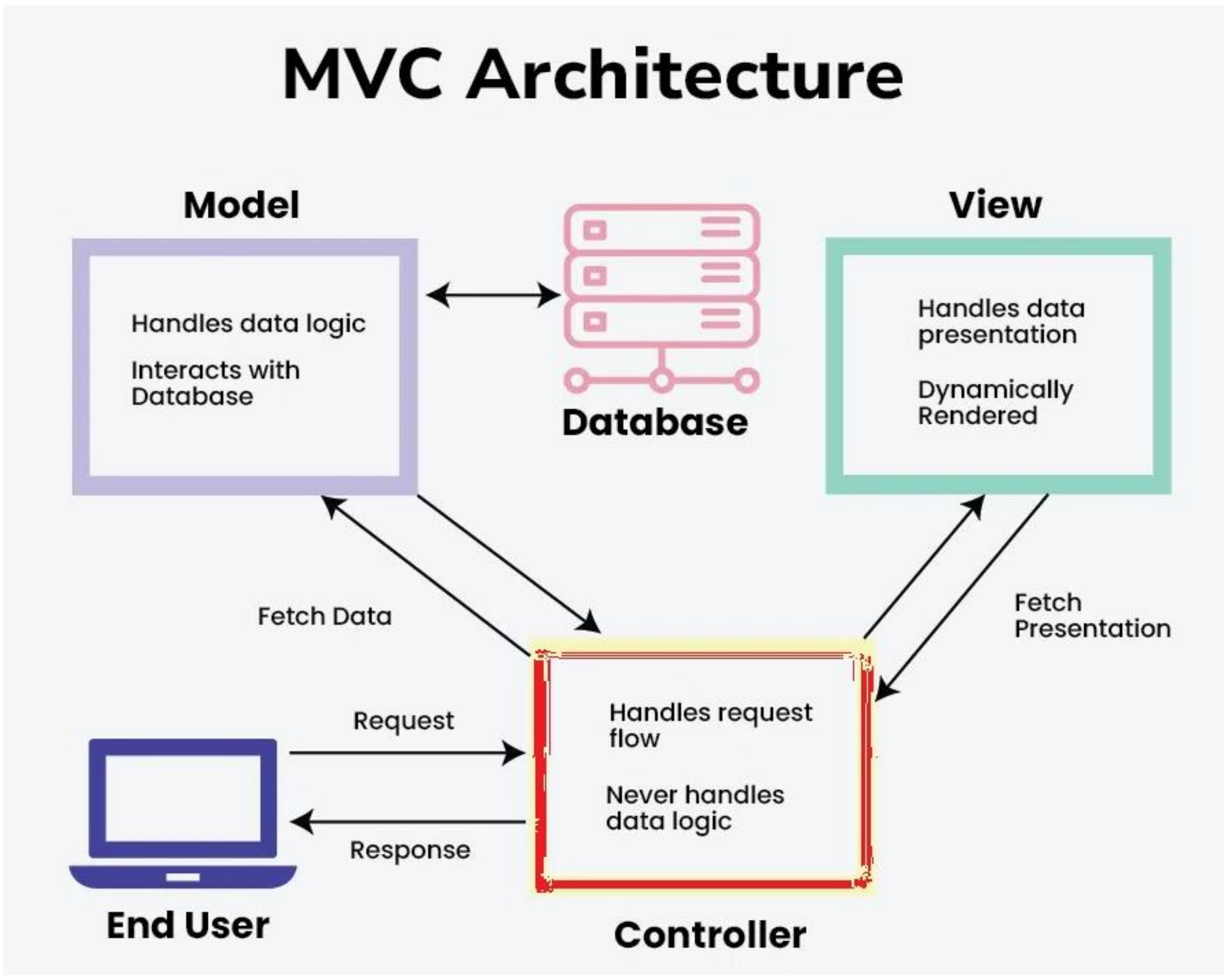
- **Purpose:** Execute business logic, handle HTTP requests, interact with databases, manage user sessions, and expose APIs for the front-end.
- **Role:**
    - **Java with Spring Framework:** A robust and widely used choice for enterprise applications. Spring provides a comprehensive ecosystem for building web applications (Spring MVC for REST APIs), handling data (Spring Data JPA), security (Spring Security), and more.
    - **Node.js (JavaScript):** Allows developers to use JavaScript on the server-side, enabling a unified language across the full stack. Excellent for building fast, scalable network applications.
    - **Python (Django, Flask):** Known for its readability and extensive libraries, popular for web development, data science, and AI.

3. **ORM / Persistence Frameworks (e.g., Hibernate):**

- **Purpose:** Bridge the gap between object-oriented programming languages (like Java) and relational databases.
- **Role:** Allows developers to interact with database tables using familiar objects and methods of their programming language instead of writing raw SQL. This simplifies data access and makes the code more maintainable.
    - **Hibernate:** A leading ORM (Object Relational Mapping) for Java. It maps Java classes to database tables and Java data types to SQL data types, managing the object-relational impedance mismatch. Often used with Spring Data JPA (Java Persistence API) for even easier data access in Spring applications.

MVC Architecture

The **Model-View-Controller (MVC)** framework is an architectural/design pattern that separates an application into three main logical components Model, View, and Controller. Each architectural component is built to handle specific development aspects of an application. It isolates the business logic and presentation layer from each other. Nowadays, MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects.



Components of MVC

The MVC framework includes the following 3 components:

- *Controller*
- *Model*
- *View*

Controller:

The controller is the component that enables the interconnection between the views and the model so it acts as an intermediary. The controller doesn’t have to worry about handling data logic, it just tells the model what to do. It processes all the business logic and incoming requests, manipulates data using the **Model** component, and interacts with the **View** to render the final output.

Responsibilities:

- Receiving user input and interpreting it.
- Updating the Model based on user actions.
- Selecting and displaying the appropriate View.

Example: In a bookstore application, the Controller would handle actions such as searching for a book, adding a book to the cart, or checking out.

View:

The **View** component is used for all the UI logic of the application. It generates a user interface for the user. Views are created by the data which is collected by the model component but these data aren’t taken directly but through the controller. It only interacts with the controller.

Responsibilities:

- Rendering data to the user in a specific format.
- Displaying the user interface elements.
- Updating the display when the Model changes.

Example: In a bookstore application, the View would display the list of books, book details, and provide input fields for searching or filtering books.

Model:

The **Model** component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the **View** and **Controller** components or any other business logic-related data. It can add or retrieve data from the database. It responds to the controller's request because the controller can't interact with the database by itself. The model interacts with the database and gives the required data back to the controller.

Responsibilities:

- Managing data: CRUD (Create, Read, Update, Delete) operations.
- Enforcing business rules.
- Notifying the View and Controller of state changes.

Benefits of MVC Architecture

MVC architecture offers several advantages:

- **Enhanced Organization:** MVC is useful in splitting the application into different parts thus improving on the way the codebase is structured. This also helps in easy identification of errors that exist in a program, implement changes to the existing program and even have better understanding of the overall structure of the created application.
- **Parallel Development:** It means that two or more developers can work on Model, View, and Controller form at the same time without attempting or interfering with the work of the other. This aspect of parallel development can go along way in enhancing the development process since they are being worked on simultaneously.
- **Code Reusability:** Compared with other designs, elements of MVC can be reused from part of the application to another. For example, a Model representing user data can be reused in different views and controllers and modification of one will automatically affect them because they are derived from it.
- **Improved Maintainability:** It bellows from the fact that this methodology of separation of concerns makes it easier to maintain and update the application. Refinements to one part (for example, modifying the Model by altering the database schema) affect no other part at least at the conceptual level (for example, the View should not be impacted).
- **Testability:** Compared with other architectures, MVC architecture is relatively easy to test because each component of the architecture can be tested separately. Writing unit tests can be done for the Model, View and Controller, thus assuring that each component is correct before the others.

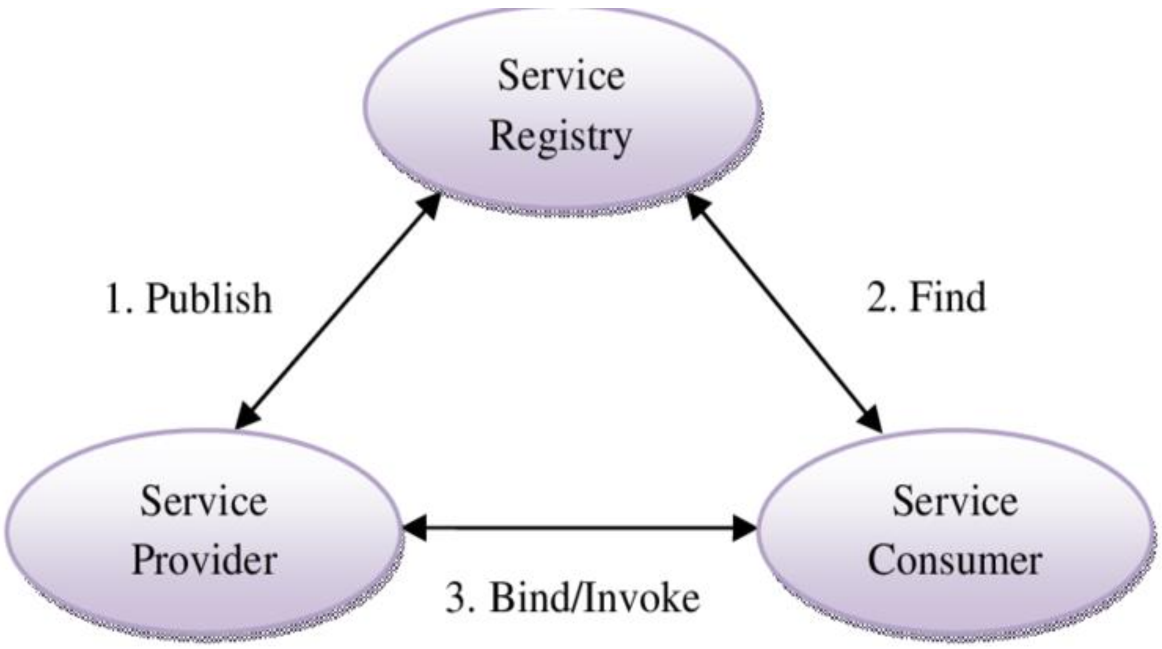
Popular MVC (Model-View-Controller) frameworks

- **Spring (Java):** A comprehensive framework for building enterprise-level Java applications.
- **Spring Boot (Java):** A streamlined version of Spring, designed for rapid application development.
- **Django (Python):** A high-level Python web framework that encourages rapid development and clean, pragmatic design.
- **ASP.NET Core (C#):** A cross-platform framework for building web, mobile, and desktop applications.

Web Services

A **Web Service** is a standardized way for two different applications to communicate over the **internet**. It allows web-based systems to **interact, exchange data, and perform tasks** using HTTP and other protocols.

In full stack development, **web services** connect the **Front-End (Browser)** to the **Back-End (Server/Database)**.



1. Service Provider

Role:
The **Service Provider** is the **system that offers a web service**. It hosts and publishes the service so that it can be discovered and invoked by others.

Responsibilities:

- Implements the business logic of the service.
- Publishes service descriptions (usually in WSDL for SOAP or OpenAPI for REST).
- Handles incoming requests and sends responses.

Example:

A REST API that provides weather information (GET /weather/today) is a **Service Provider**.

2. **Service Registry**

Role:

The **Service Registry** is a **central directory** that allows service providers to **publish** their services and service requesters to **find** them.

Responsibilities:

- Stores metadata about services.
- Provides mechanisms to search and locate services.
- Supports publish/find/bind operations.

Standards:

- **UDDI (Universal Description, Discovery, and Integration)** is a traditional registry standard (mostly used with SOAP).
- In REST-based services, registries may be API Gateways (e.g., SwaggerHub, Postman, Kong).

Example:

- A company registers its service on a UDDI node.
- A client searches this registry and finds the service based on keywords or categories.

3. **Service Requester**

Role:

The **Service Requester** is the **consumer** of the web service. It locates, binds to, and invokes operations provided by the Service Provider.

Responsibilities:

- Searches for available services (via the registry).
- Retrieves the service description.
- Sends a request to the service.
- Processes the response from the service provider.

Example:

A mobile weather app that fetches data from the weather API is a **Service Requester**.

Types of Web Services

Web services enable machine-to-machine communication over the internet using standardized protocols. The main types of web services include:

1. SOAP Web Services (Simple Object Access Protocol)

Description:

SOAP is a **protocol-based** web service that uses **XML** to encode requests and responses. It follows strict standards and is highly structured.

Key Features:

- **Uses XML** exclusively.
- Relies on **WSDL (Web Services Description Language)** to describe services.
- Works over multiple transport protocols (HTTP, SMTP, FTP, etc.).
- Built-in error handling and security (WS-Security).

Best For:

- Enterprise applications.
- Scenarios requiring **strong security, transaction management**, and formal contracts.

Example Use Case:

- A banking system using SOAP for secure transactions.

2. RESTful Web Services (Representational State Transfer)

Description:

REST is an **architectural style**, not a protocol. It uses **HTTP methods** (GET, POST, PUT, DELETE) and typically exchanges **JSON** or XML.

Key Features:

- Lightweight and stateless.
- URL-based access to resources.
- Uses standard HTTP verbs.
- More scalable and faster than SOAP.

Best For:

- Web and mobile applications.
- Public APIs and microservices.

Example Use Case:

- A React app fetching product data from a Spring Boot REST API (GET /products).

Advantages of Web Services

1. Platform and Language Independence

- Web services use **standard protocols** like HTTP and data formats like **XML** or **JSON**.
- This makes them accessible from any platform (Windows, Linux, etc.) and any language (Java, Python, .NET, etc.).

Example: *A Java backend can provide services consumed by a JavaScript frontend.*

2. Interoperability

- Web services enable different systems to **communicate and share data**, even if they are built on different technologies or architectures.

Example: *A .NET client calling a Python REST API.*

3. Reusability

- Once developed, a web service can be **reused** across multiple applications or modules.

Example: *A payment service can be used by both mobile and web applications.*

4. Modularity and Maintainability

- Web services promote **loose coupling** between client and server. Each service is independent and can be developed, deployed, and updated independently.

Example: *Easier to maintain and scale large applications.*

5. Scalability and Flexibility

- Web services can be scaled horizontally (across servers).
- REST services are stateless, which helps in load balancing and performance optimization.

6. Improved Integration

- Helps in **integration** between legacy systems and modern applications.
- Common in B2B communication and enterprise application integration (EAI).

Example: *Integrating an old ERP system with a new web-based dashboard.*

7. Supports Standardized Protocols

- SOAP, REST, gRPC, GraphQL all follow industry standards.
- Ensures security (WS-Security, OAuth), reliability, and governance.

8. Easier Testing and Debugging

- Tools like **Postman**, **Swagger**, and **SoapUI** simplify API testing.
- RESTful services often provide interactive documentation (OpenAPI/Swagger).

9. Cost Efficiency

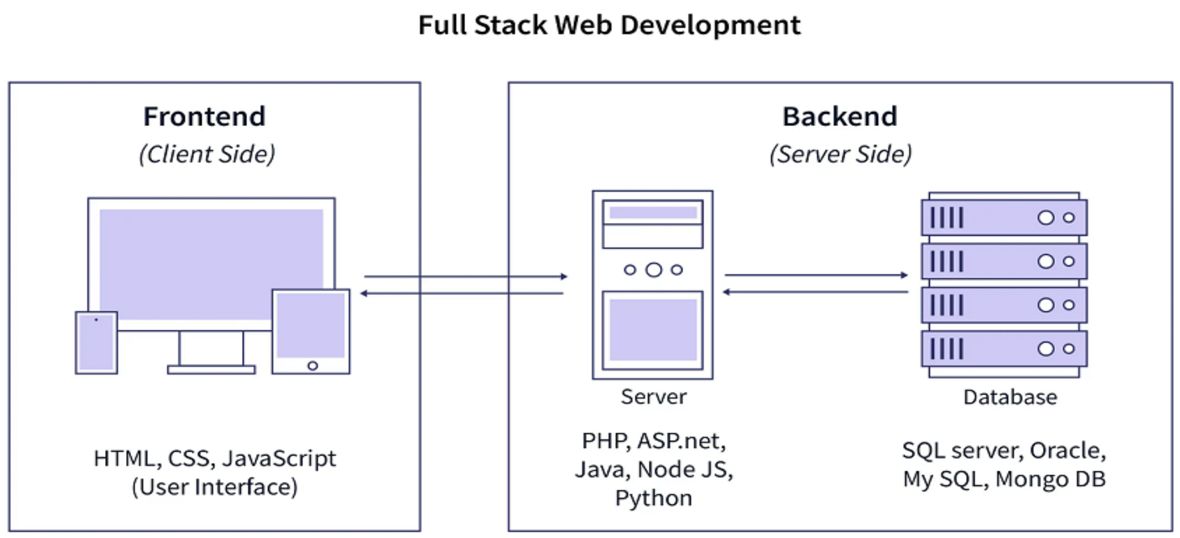
- Reduces duplication of logic.
- Centralized services lower development and maintenance cost across teams.

10. Cloud and Microservices Ready

- Web services are foundational to **cloud computing** and **microservice architectures**.
- Easy deployment to AWS, Azure, GCP, etc.

Communication between Front-End and Back-End

In Full-Stack Development, the communication between the Front-End (the Client) and the Back-End (the Server) is the fundamental process that brings a web application to life. It is the bridge that allows the user's interface to interact with the application's data and business logic.


Full Stack Web Development
Frontend
(Client Side)
HTML, CSS, JavaScript
(User Interface)
Backend
(Server Side)
Server
PHP, ASP.net, Java, Node JS, Python
Database
SQL server, Oracle, My SQL, Mongo DB

The Communication Workflow: Step-by-Step

1. **User Action (Front-End):** A user interacts with the front-end application (e.g., clicks a "Buy" button on a product page). The front-end is built with technologies like **HTML, CSS, and React**.
2. **Front-End Prepares Request (JavaScript):**
   - The **React** component's associated **JavaScript** code captures the user event.
   - It gathers the necessary data (e.g., the product ID, the quantity).
   - It uses a JavaScript API or library to make an asynchronous HTTP request.
   - **Common Tools:**
     - **Fetch API:** A modern, built-in JavaScript API for making network requests.
     - **Axios:** A popular third-party library that simplifies making HTTP requests, handles errors, and provides a more consistent interface.
3. **HTTP Request is Sent:**
   - The client's browser sends an **HTTP request** (e.g., a POST request) to a specific back-end endpoint (e.g., https://api.myapp.com/api/orders).
   - The request includes the data, usually formatted as **JSON** in the request body.

4. **Back-End Receives and Processes Request (Server-Side):**
   - The back-end server (e.g., running an application built with **Spring, Servlets, and Hibernate**) receives the incoming HTTP request.
   - A **Controller** (e.g., a **Spring REST Controller** or a **Servlet**) is mapped to the URL of the endpoint. This controller method is executed.
   - The controller reads the incoming JSON data from the request body.
   - The controller invokes the **business logic** (e.g., an order service) to process the request.
   - The business logic interacts with the **database** (via **Hibernate**) to save the new order.
5. **Back-End Generates and Sends Response:**
   - Once the operation is complete (e.g., the order is saved successfully), the back-end generates a response.
   - The response includes:
     - An **HTTP Status Code** (e.g., 201 Created for success, 400 Bad Request for an error).
     - A **response body**, typically a new JSON object containing the details of the created order, an error message, or a success message.
   - The back-end sends this HTTP response back to the client.
6. **Front-End Receives and Renders Response (JavaScript/React):**
   - The JavaScript code in the browser receives the HTTP response.
   - It checks the HTTP status code to see if the request was successful.
   - It parses the **JSON** response body to get the returned data.
   - **React** then updates the UI based on this data. For example, it might show a confirmation message, display the new order in a list, or navigate to a new page.

JSON (Java Script Object Notation)

JSON is an acronym for **JavaScript Object Notation**, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange. It is a language-independent data format. It supports almost every kind of language, framework, and library.

JSON is an open standard for exchanging data on the web. It supports data structures like objects and arrays. So, it is easy to write and read data from JSON.

JSON Data Types

Following are the most commonly used JSON data types.

Data Type	Description	Example
String	A string is always written in double-quotes. It may consist of numbers, alphanumeric and special characters.	"student", "name", "1234", "Ver_1"
Number	Number represents the numeric characters.	121, 899
Boolean	It can be either True or False.	true
Null	It is an empty value.	

JSON Objects

In JSON, objects refer to dictionaries, which are enclosed in curly brackets, i.e., { }. These objects are written in key/value pairs, where the key has to be a string and values have to be a valid JSON data type such as string, number, object, Boolean or null. Here the key and values are separated by a colon, and a comma separates each key/value pair.

For example:

{"name" : "Jack", "employeeid" : 001, "present" : false}

JSON Arrays

In JSON, arrays can be understood as a list of objects, which are mainly enclosed in square brackets [ ].

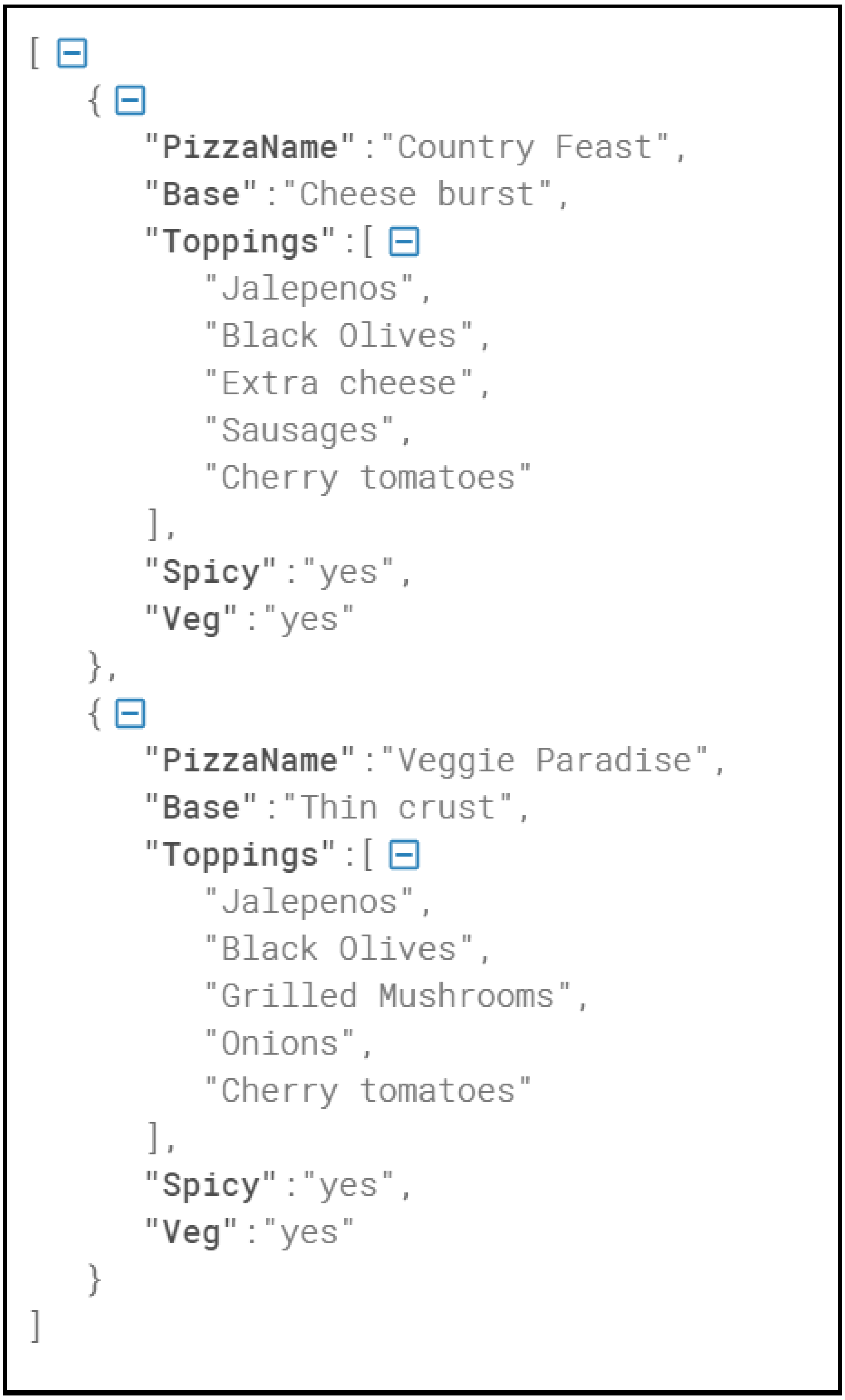
An array value can be a string, number, object, array, boolean or null.

For example:

```
 [
{
"PizzaName" : "Country Feast",
"Base" : "Cheese burst",
"Toppings" : ["Jalepenos", "Black Olives", "Extra cheese", "Sausages", "Cherry tomatoe
s"],
"Spicy" : "yes",
"Veg" : "yes"
},

{
"PizzaName" : "Veggie Paradise",
"Base" : "Thin crust",
"Toppings" : ["Jalepenos", "Black Olives", "Grilled Mushrooms", "Onions", "Cherry to
matoes"],
"Spicy" : "yes",
"Veg" : "yes"
}
]
```

In the above example, the object "Pizza" is an array. It contains five objects, i.e., PizzaName, Base, Toppings, Spicy, and Veg.



```
[
   {
      "PizzaName":"Country Feast",
      "Base":"Cheese burst",
      "Toppings":[
         "Jalepenos",
         "Black Olives",
         "Extra cheese",
         "Sausages",
         "Cherry tomatoes"
      ],
      "Spicy":"yes",
      "Veg":"yes"
   },
   {
      "PizzaName":"Veggie Paradise",
      "Base":"Thin crust",
      "Toppings":[
         "Jalepenos",
         "Black Olives",
         "Grilled Mushrooms",
         "Onions",
         "Cherry tomatoes"
      ],
      "Spicy":"yes",
      "Veg":"yes"
   }
]
```

JSON parsing and serialization

JSON parsing and serialization are fundamental processes in working with JSON data, a widely used format for data interchange.

Parsing converts a JSON string into a data structure (like a JavaScript object), while serialization does the reverse, converting a data structure into a JSON string.

These processes are essential for interacting with APIs, storing data, and transmitting information across different systems.

JSON Parsing (Deserialization):

Definition: Parsing, also known as deserialization, is the process of converting a JSON string into a usable data structure in a programming language. For example, in JavaScript, JSON.parse() is used to convert a JSON string into a JavaScript object.

Example (JavaScript):

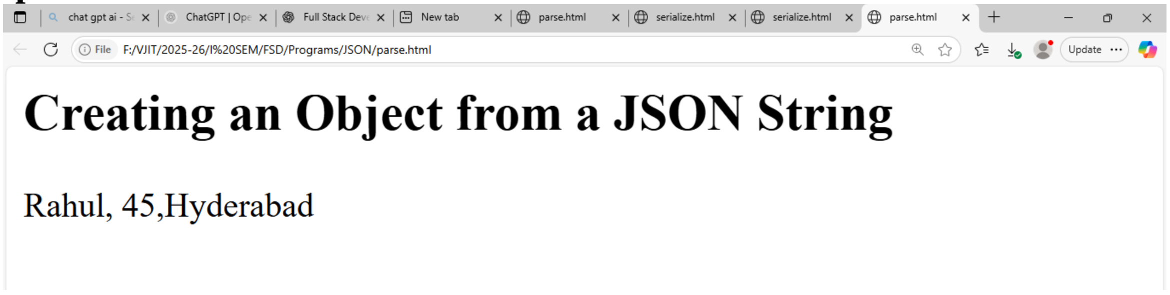
JavaScript

```
const jsonString = '{"name":"Rahul", "age":45, "city":"Hyderabad"}'
const javascriptObject = JSON.parse(jsonString);
// javascriptObject will now be: { name: "Rahul", age: 45, city:"Hyderabad }
```

Key Function: JSON parsing allows you to take data received from an external source (like an API) and make it accessible within your application.

```
<html>
<body>
<h2>Creating an Object from a JSON String</h2>
<p id="demo"></p>
<script>
const txt = '{"name":"Rahul", "age":45, "city":"Hyderabad"}'
const obj = JSON.parse(txt);
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age + ","+
obj.city;
</script>
</body>
</html>
```

Output:


chat gpt ai - S | ChatGPT | Op | Full Stack Dev | New tab | parse.html | serialize.html | serialize.html | parse.html

File F:/VJIT/2025-26/I%20SEM/FSD/Programs/JSON/parse.html

Creating an Object from a JSON String

Rahul, 45,Hyderabad

JSON Serialization:

Definition: Serialization is the process of converting a data structure (like a JavaScript object) into a JSON string.

Example (JavaScript):

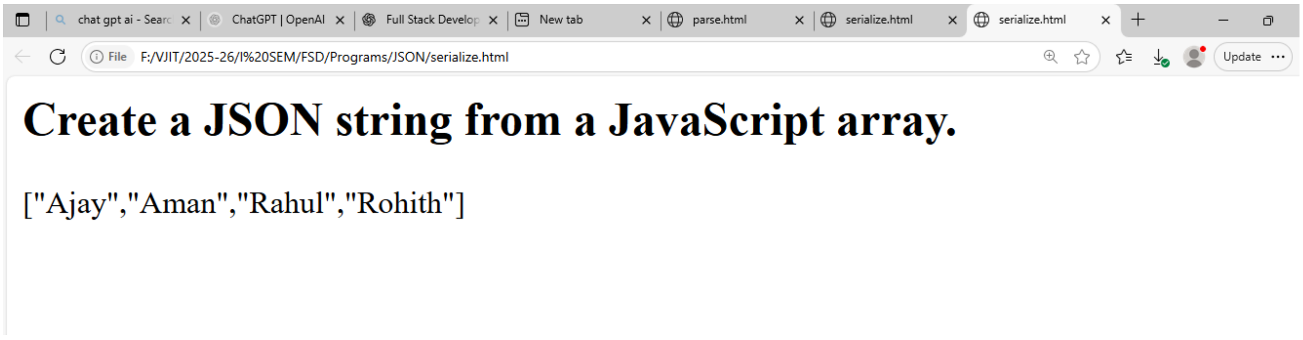
JavaScript

```
const javascriptArray = ["Ajay", "Aman", "Rahul", "Rohith"];
const jsonString = JSON.stringify(javascriptArrayt);
 // jsonString will now be: ["Ajay", "Aman", "Rahul", "Rohith"];
```

- **Key Function:** JSON serialization enables you to send data from your application to another system or store it in a format that can be easily transmitted or stored.

```
<html>
<body>
<h2>Create a JSON string from a JavaScript array.</h2>
<p id="demo"></p>
<script>
const arr = ["Ajay", "Aman", "Rahul", "Rohith"];
const myJSON = JSON.stringify(arr);
document.getElementById("demo").innerHTML = myJSON;
</script>
</body>
</html>
```

Output:


chat gpt ai - Searc | ChatGPT | OpenAI | Full Stack Develop | New tab | parse.html | serialize.html | serialize.html

File F:/VJIT/2025-26/I%20SEM/FSD/Programs/JSON/serialize.html

Create a JSON string from a JavaScript array.

["Ajay","Aman","Rahul","Rohith"]