**Servlets**

**### Basics of Web ###**

**Web applications are software programs that run on web servers and are accessed via web browsers over a network, typically the Internet. They provide functionality through a user-friendly interface and can range from simple websites to complex applications like online banking systems, e-commerce platforms, and social media sites.**

**### Key Components: ###**

**1. Client: The user interface, typically a web browser, that interacts with the web application.**

**2. Server: The backend that processes requests from the client, performs business logic, and serves responses.**

**3. Database: Stores data required by the web application.**

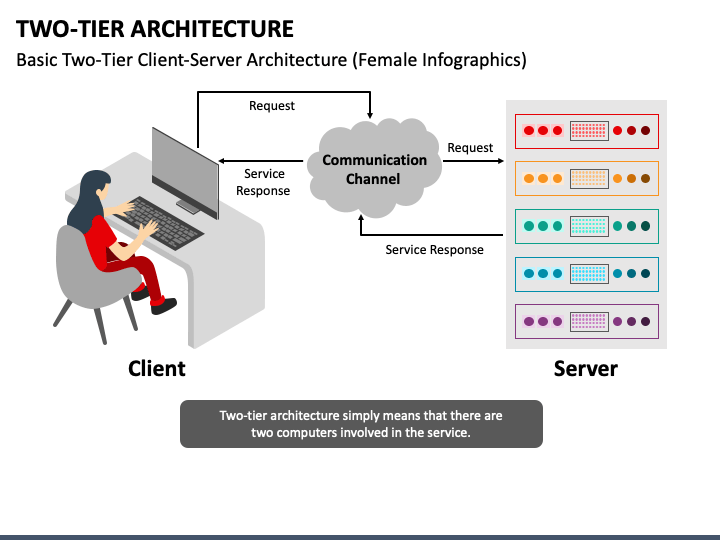
**### Client-Server Architecture ###**

**Client-server architecture is a computing model where clients (users) request services and resources from a centralized server.**

**Types of Client-Server Architecture:**

**1. Two-Tier Architecture:**

* **Client: Manages the user interface and sends requests to the server.**
* **Server: Handles requests, processes data, and returns responses.**
* **Example: A web browser (client) requesting a web page from a web server.**

****

**2. Three-Tier Architecture:**

* **Presentation Tier (Client): The user interface layer (web browser).**
* **Logic Tier (Application Server): Processes business logic and handles communication between the client and the database.**
* **Data Tier (Database Server): Manages the storage and retrieval of data.**
* **Example: An e-commerce website where the client interacts with the application server, which in turn interacts with the database server.**

****

**3. N-Tier Architecture:**

* **Extends the three-tier architecture by adding more layers (e.g., additional application servers) to improve scalability and manageability.**
* **Example: Large-scale enterprise applications with multiple application servers handling different types of business logic.**

**### HTTP Protocol ###**

***Hypertext Transfer Protocol (HTTP****)* **is an** [**application-layer**](https://en.wikipedia.org/wiki/Application_Layer) **protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical** [**client-server model**](https://en.wikipedia.org/wiki/Client%E2%80%93server_model)**, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a** [**stateless protocol**](https://en.wikipedia.org/wiki/Stateless_protocol)**, meaning that the server does not keep any data (state) between two requests.** [**https://developer.mozilla.org/en-US/docs/Web/HTTP**](https://developer.mozilla.org/en-US/docs/Web/HTTP)

**### HTTP request methods ###**

**HTTP defines a set of *request methods* to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as *HTTP verbs*. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be** [**safe**](https://developer.mozilla.org/en-US/docs/Glossary/Safe/HTTP)**,** [**idempotent**](https://developer.mozilla.org/en-US/docs/Glossary/Idempotent)**, or** [**cacheable**](https://developer.mozilla.org/en-US/docs/Glossary/Cacheable)**.**

[**https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods)

# ### HTTP response status codes ###

**HTTP response status codes indicate whether a specific** [**HTTP**](https://developer.mozilla.org/en-US/docs/Web/HTTP) **request has been successfully completed. Responses are grouped in five classes:**

1. [**Informational responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#information_responses) **(100 – 199)**
2. [**Successful responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#successful_responses) **(200 – 299)**
3. [**Redirection messages**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#redirection_messages) **(300 – 399)**
4. [**Client error responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#client_error_responses) **(400 – 499)**
5. [**Server error responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#server_error_responses) **(500 – 599)**

[**https://developer.mozilla.org/en-US/docs/Web/HTTP/Status**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

**### Web Servers and Its Types**

**A web server is software that serves web pages to clients over the HTTP protocol. It processes incoming requests, retrieves the requested resources, and sends responses back to the client.**

**### Types of Web Servers:**

**1. Static Web Servers:**

**- Serve static content like HTML, CSS, JavaScript, images, and videos.**

**- Content is pre-stored and does not change dynamically.**

**- Example: Nginx, Apache HTTP Server.**

**2. Dynamic Web Servers:**

**- Generate content dynamically based on server-side logic.**

**- Use programming languages like Java, PHP, Python, or Node.js to process requests and generate responses.**

**- Often integrate with databases to retrieve and update data.**

**- Example: Apache Tomcat, Microsoft IIS, Node.js.**

**3. Reverse Proxy Servers:**

**- Act as intermediaries between clients and backend servers.**

**- Distribute client requests to different backend servers to balance the load.**

**- Can cache content to improve performance.**

**- Example: Nginx, HAProxy.**

**4. Application Servers:**

**- Provide a platform for running web applications, offering additional features like transaction management, messaging, and security.**

**- Often include web server functionality.**

**- Example: IBM WebSphere, Oracle WebLogic, JBoss EAP.**

**By understanding these concepts, you’ll have a solid foundation for developing and managing web applications effectively. If you need more detailed explanations or examples, feel free to ask!**

**Web Services**

Web services are software systems designed to allow for interoperable communication and interaction over a network, typically the internet. They facilitate the exchange of data and functionalities between different applications or systems, making it possible for them to work together. Web services follow specific communication protocols and use standard formats like XML or JSON for data exchange.

There are several types of web services, with two fundamental categories being:

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

* SOAP is a protocol for exchanging structured information in web services.
* It uses XML for message formatting and relies on other protocols like HTTP and SMTP for message negotiation and transmission.
* It defines a set of rules for structuring messages, including headers and bodies.

2. RESTful Web Services (Representational State Transfer):

* REST, or Representational State Transfer,
* is an architectural style for designing networked applications.
* A RESTful API (Application Programming Interface) is a set of rules and conventions for building and interacting with web services.
* It typically uses standard HTTP methods like GET, POST, PUT, DELETE to perform operations on resources, and it relies on stateless communication, meaning each request from a client contains all the information needed to understand and fulfill that request.
* JSON is commonly used for data interchange in REST APIs.
* It often utilizes JSON or XML for data representation.

**### Eclipse Jakarta Project ###**

**The Eclipse Jakarta Project, also known as Jakarta EE, is an open-source project that is managed by the Eclipse Foundation. It is the successor to Java EE (Java Platform, Enterprise Edition) and provides a set of specifications for developing enterprise-level applications in Java.**

**1. History and Evolution:**

* **Originally developed by Sun Microsystems as Java EE.**
* **Acquired by Oracle, which later donated it to the Eclipse Foundation.**
* **Renamed from Java EE to Jakarta EE to reflect its new governance under the Eclipse Foundation.**

**2. Goals:**

* **To provide a set of stable, scalable, and enterprise-level specifications for Java.**
* **To promote the adoption of cloud-native and microservices architectures.**
* **To ensure compatibility and interoperability between implementations.**

**3. Specifications:**

**Jakarta EE includes a wide range of specifications, such as:**

* **Jakarta Servlet: For creating web applications.**
* **Jakarta Persistence (JPA): For managing relational data.**
* **Jakarta Contexts and Dependency Injection (CDI): For managing dependencies.**
* **Jakarta RESTful Web Services (JAX-RS): For building RESTful web services.**
* **Jakarta Faces (JSF): For building component-based user interfaces.**
* **Jakarta Transactions (JTA): For managing transactions.**
* **Jakarta Messaging (JMS): For sending messages between applications.**

**4. Governance and Community:**

* **Managed by the Eclipse Foundation, which ensures open governance, vendor neutrality, and community-driven development.**
* **Supported by a large community of developers, organizations, and vendors.**

**5. Implementations:**

* + **Several Jakarta EE-compatible application servers and frameworks are available, such as Eclipse GlassFish, Red Hat’s WildFly, IBM’s Open Liberty, and Payara Server.**

**### Benefits of Using Jakarta EE: ###**

* + **Standardization: Provides a consistent set of APIs and standards for building enterprise applications.**
  + **Portability: Applications built using Jakarta EE can be easily moved between different application servers.**
  + **Scalability: Designed to handle large-scale, distributed, and transactional applications.**
  + **Productivity: Rich set of tools and libraries that help developers build robust applications quickly.**

**### Explore Documentation and Tutorials: ###**

* **Official Jakarta EE documentation:** [**Jakarta EE Specifications | The Eclipse Foundation**](https://jakarta.ee/specifications/)
* **Tutorials and examples:**

1. [**Overview :: Jakarta EE Documentation**](https://jakarta.ee/learn/docs/jakartaee-tutorial/current/intro/overview/overview.html)
2. [**The Jakarta® EE Tutorial (eclipse-ee4j.github.io)**](https://eclipse-ee4j.github.io/jakartaee-tutorial/)

**By understanding and leveraging the Eclipse Jakarta Project, you can build powerful, scalable, and maintainable enterprise applications in Java.**

**### Tomcat Web Server ###**

**### 2. Servlet Basics**

**Introduction to Servlets: Understand what a servlet is, its purpose, and how it fits into the Java EE ecosystem.**

**Setting Up the Environment: Install necessary tools like JDK, Apache Tomcat, and an IDE (e.g., Eclipse or IntelliJ IDEA).**

**### 3. Servlet Lifecycle**

**Servlet Lifecycle Methods: Learn about the `init()`, `service()`, and `destroy()` methods.**

**Lifecycle Phases: Understand the initialization, request handling, and destruction phases of a servlet.**

**### 4. Servlets API**

**Servlet Interfaces and Classes: Explore the key interfaces (e.g., `Servlet`, `ServletRequest`, `ServletResponse`) and classes (e.g., `GenericServlet`, `HttpServlet`).**

**Servlet Packages: Study the `javax.servlet` and `javax.servlet.http` packages.**

**### 5. HTTP Servlets**

**HTTP Protocol: Understand the basics of HTTP, including methods (GET, POST, PUT, DELETE), status codes, and headers.**

**Handling HTTP Requests and Responses: Learn how to handle HTTP requests and responses in servlets.**

**### 6. Servlets Configuration**

**Deployment Descriptor (web.xml): Understand how to configure servlets using the `web.xml` file.**

**Annotations: Learn how to use annotations for servlet configuration (`@WebServlet`, `@WebInitParam`).**

**### 7. Servlet Context**

**ServletContext Interface: Understand the `ServletContext` interface and its methods.**

**Context Parameters: Learn how to configure and access context parameters.**

**### 8. Servlets Collaboration**

**RequestDispatcher: Learn about the `RequestDispatcher` interface for forwarding and including requests.**

**Servlet Communication: Understand how servlets communicate with each other.**

**### 9. Session Tracking**

**Session Management: Explore different techniques for session management, including cookies, URL rewriting, and `HttpSession` API.**

**Session Attributes: Learn how to store and retrieve session attributes.**

**### 10. CRUD Operations**

**Database Connectivity: Understand how to connect to a database using JDBC.**

**CRUD Operations: Learn to implement Create, Read, Update, and Delete operations using servlets.**

**## Learning Resources**

**### Books**

**- \*Head First Servlets and JSP\* by Bryan Basham, Kathy Sierra, and Bert Bates**

**- \*Java Servlet & JSP Cookbook\* by Bruce W. Perry**

**### Online Tutorials**

**- [Oracle's Official Servlet Documentation](https://docs.oracle.com/javaee/7/tutorial/servlets.htm)**

**- [Baeldung Servlet Tutorial](https://www.baeldung.com/java-ee-servlet)**

**Basics of Web, Servlet Lifecycle, Servlets API, HTTP Servlets, Servlets**

**Configuration, Servlets Context, Servlets Collaboration, Session**

**Tracking, CRUD operations**