**RESTUL API**

**RESTful API, or simply a REST API, is a web service that follows the principles of Representational State Transfer (REST) architecture. REST is a set of architectural constraints that are applied to web services, which can be used to create highly scalable and flexible web services.**

**One of the key features of a REST API is that it is stateless, which means that the server does not store any information about the client's state. Instead, all information required to service a request is included in the request itself. This allows for greater scalability and flexibility, as the server does not need to maintain any state information about the client.**

**Another key feature of a REST API is that it uses HTTP methods, such as GET, POST, PUT and DELETE, to indicate the type of operation the client requests. For example, a GET request retrieves information from the server, while a POST request is used to submit information to the server. This allows for a simple and consistent way of interacting with the server, regardless of the type of data being exchanged.**

**REST API also uses a simple and consistent URL structure, which makes it easy for developers to understand and interact with the API. The URL structure typically includes the base URL, followed by a resource path and an optional query string. The resource path is used to specify the specific resource that the client is interacting with, while the query string is used to pass additional information to the server.**

**REST APIs also use standard HTTP response codes to indicate the status of a request. For example, a 200 OK response indicates that the request was successful, while a 400 Bad Request response indicates that the request was invalid. This allows for easy and consistent error handling, as developers can easily understand the meaning of a particular response code.**

**One of the major benefits of using a REST API is that it is platform-independent, which means that it can be used with any programming language or framework that supports HTTP. This allows for greater flexibility and ease of integration with existing systems, as developers can use the tools and technologies with which they are already familiar.**

**Another benefit of using a REST API is that it is lightweight and easy to use. It uses standard HTTP methods and response codes, which makes it easy for developers to understand and interact with the API. Additionally, because a REST API is stateless, it does not require session or cookie-based authentication, which simplifies the authentication process.­­­**

|  |  |  |
| --- | --- | --- |
| **Sl NO** | **CONTENT** | **PGNO** |
| **1** | **IMPLEMEENT RESTUL API** | **3** |
| **2** | **CREATE A CLASS IN ABOVE PACKAGE** | **4** |
| **3** | **GO WEB BROWSER AND CHECK THE OUTPUT** | **5** |

**INDEX**

**Client-Server Separation: This separates the user interface concerns (handled by the client) from the data storage and processing concerns (handled by the server), improving the flexibility of development.**

**Stateless: Each request from the client to the server must contain all the information needed to understand and process the request, without relying on stored context on the server.**

**Cacheable: Responses should be defined as cacheable or non-cacheable to improve performance and scalability by reducing redundant server-side processing.**

**Uniform Interface: By adhering to conventions like resource naming and HTTP methods (GET, POST, PUT, DELETE), REST creates a standard way for clients to interact with APIs.**

**Layered System: A client cannot tell whether it is directly connected to the end server or to an intermediary, which enhances security, scalability, and flexibility.**

**Code on Demand (optional): Servers can transfer executable code to the client, enhancing the functionality without changing the underlying architecture.**

**REST uses a resource-based approach. Each piece of data (or resource) is represented by a URI (Uniform Resource Identifier). In RESTful APIs, a resource might represent entities like users, orders, products, etc.**

**Examples:**

**GET /api/users – Retrieve a list of users.**

**POST /api/users – Create a new user.**

**GET /api/users/{id} – Retrieve a specific user based on their unique identifier.**

**DELETE /api/users/{id} – Delete the user with the specified ID.**

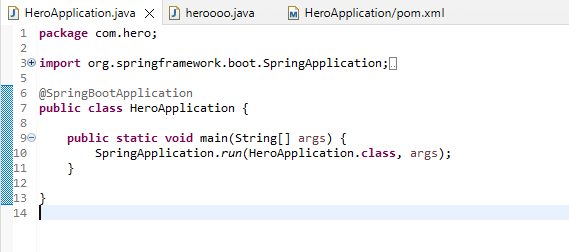
**Resource names should be nouns (e.g., /users, /orders) rather than actions (like /getUsers or /deleteUser).**

**Use Consistent Resource Naming: Use plural nouns (e.g., /users instead of /user) and avoid using verbs in URIs.**

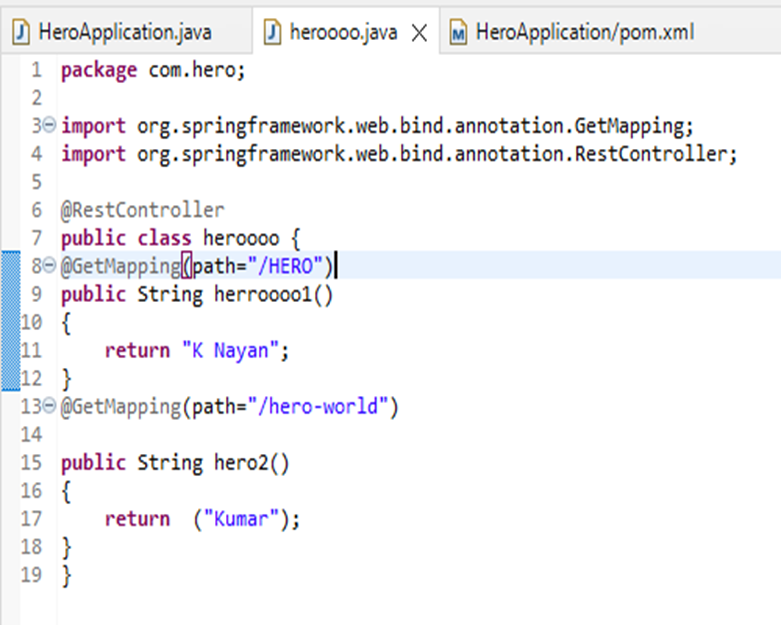
**Use Appropriate HTTP Methods: Always align your HTTP method with the action being performed (GET for reading, POST for creating, etc.).**

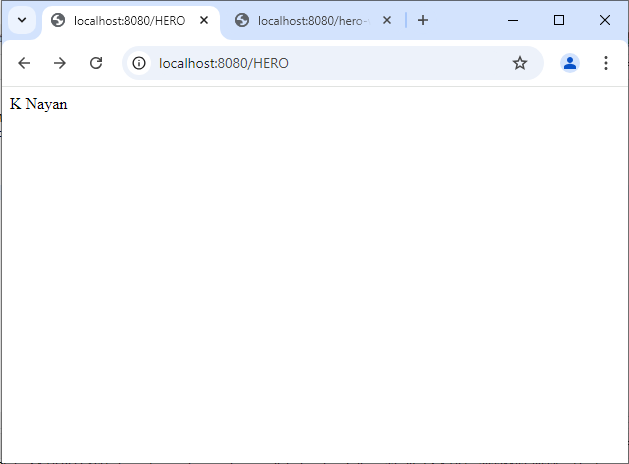
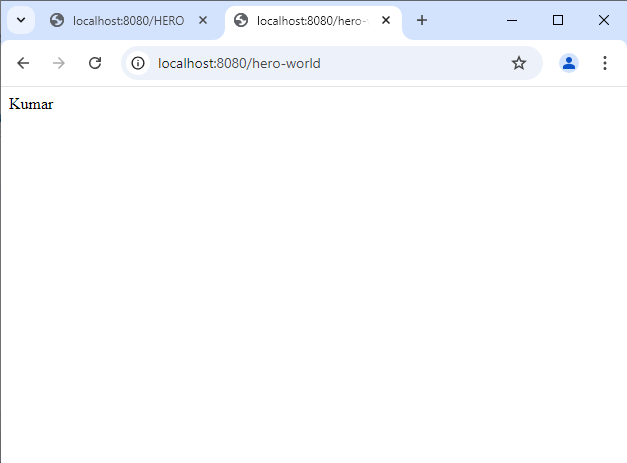
**Pagination: When dealing with large data sets, always provide pagination. A common format is GET /users?page=1&limit=20.**

**CREATE A CLASS**

****

**CREATE ONE MORE CLASS**

****

**OUTPUT:­ **

**Example 1: User Management API**

**GET /api/users: Returns a list of users.**

**GET /api/users/{id}: Returns data for a specific user.**

**POST /api/users: Creates a new user.**

**PUT /api/users/{id}: Updates the entire user object.**

**PATCH /api/users/{id}: Modifies part of the user object (e.g., only the email).**

**DELETE /api/users/{id}: Deletes a user by ID.**

**Example 2: E-Commerce API**

**Key Characteristics:**

**Stateless: Each request from a client contains all the information the server needs to fulfill the request. The server does not store any client context between requests.**

**Client-Server Architecture: The client and server operate independently. The client is responsible for the user interface, while the server handles requests and responses.**

**Cacheable: Responses must explicitly state whether they are cacheable or not. This allows clients to reduce server load by caching responses when possible.**

**Uniform Interface: A set of rules is established for interaction between the client and server.**

**-------X--------**