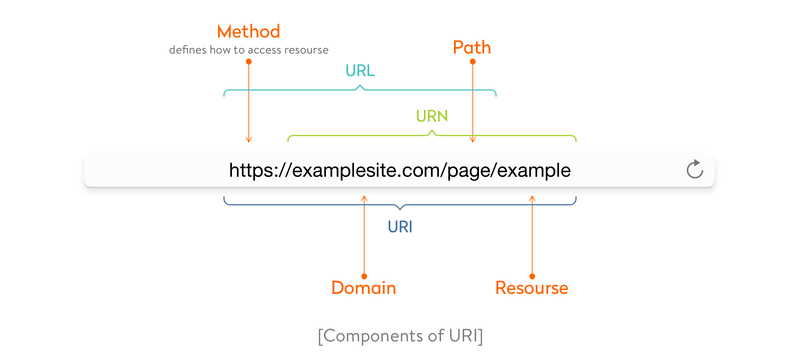
|  |
| --- |
| **REST and SOAP** |
| REST, SOAP are architectures /protocals ,the way to communicate between two systems. |
| **REST** |
| Light weight due to payload format |
| You can implement each HTTP method independently |
| They are faster than SOAP |
| Less security than SOAP |
| **Parameter Types** |
| **Template Parameter:** Parameter would be part of the URL and would be enclosed in curly braces. There can be more than one template parameters. |
| Ex: [URL: http://dummy.restapiexample.com/api/v1/employee/](http://test.com/api/vi){id}/{deptid} |
| **Query Paramater:** The Parameter would be appended with ?(question mark) and each parameter is separated by &(ampersand) |
| Ex: URL: <http://dummy.restapiexample.com/api/v1/employee?id=>[value]&deptid=[value] |

****

[**https://www.javadevjournal.com/spring/restful-methods/**](https://www.javadevjournal.com/spring/restful-methods/)

## 1. GET Method

***HTTP GET*** method used to retrieve information from the REST API.We should not use this method to change any information or the resource.GET should be idempotent, meaning regardless of how many times it repeats with the same parameters, the results are the same.

* Multiple requests should return same data for the GET request until some other processes change it (POST or PUT request).
* ***HTTP GET*** request should return *200(OK)* Response if it finds data.

Let’s take some example of the GET request URI

* GET http://www.javadevjournal.com/customers   (retrieve all customers)
* GET http://www.javadevjournal.com/customers/1   (retrieve customer with ID 1)

## 2. POST Method

This method in the REST API should only be used to create a new resource.In some cases, POST method is used to update entity but this operation is not very frequent.POST API call is not idempotent nor safe and should be used with care.Let’s cover some extra points while using POST method in our REST API.

* HTTPS POST method calls are not cachable.We need to include Cache-Control or Expires header if we want to use [cache](https://www.javadevjournal.com/spring/spring-caching/) for POST method.
* HTTPS POST must cause a cache to invalidate an entity.

Let’s take some example of POST method URI

* HTTPS POST http://www.javadevjournal.com/customers
* HTTPS POST http://www.javadevjournal.com/customers/1/addresses

HTTP/1.1 POST http://www.javadevjournal.com/customers/123

{

//customer data

}

## 3. PUT Method

If we want to update an existing resource, *PUT* method should be used for this operation.PUT method has some features as compare to the POST method and we should keep those in mind

* PUT method is idempotent.This means the client can send the same request multiple time and as per HTTP spec, this has exactly the same effect as sending once. (This is for us to make sure PUT behaves correctly every time)
* Think of PUT method as putting a resource – This means completely replacing whatever is available at the given URL with the given data/ information.
* If a new resource created by PUT request, we should tell client by sending a correct response (<em>201</em> Created code).

idempotent is the main difference between the expectations of PUT versus a POST request.We need to follow this property while designing our REST API.

Let’s take some example of PUT method URI

* HTTPS PUT http://www.javadevjournal.com/customers/1/addresses/123 (Modify the address with an ID of 123)
* HTTPS PUT http://www.javadevjournal.com/customers/123

<span class="mceItemHidden">HTTP/1.1 PUT http://www.javadevjournal.com/customers/123

{

“givenName”: “Java”,

“surname”: Dev Journal

}</span>

## 4. DELETE Method

DELETE method should be used to remove a given resource.Similar to PUT method, DELETE operation is idempotent.If a resource is deleted any later request will change anything in the system and it should return 404 response.DELETE can be a long-running or asynchronous request.Let’s take some example of DELETE method URI

* HTTPS DELETE http://www.javadevjournal.com/customers/1
* HTTPS DELETE http://www.javadevjournal.com/customers/1/addresses/123

## 5. PATCH Method

Partial update to a resource should happen through PATCH method.To differentiate between PATCH and PUT method, PATCH request used to partially change a given resource while PUT used to replace existing resource.There are few things to keep in mind while using PATCH method.

* Support for PATCH in browsers, servers and web applications are not universal, this can post some challenges.
* Request payload for Patch is not simple and a client is required to send information to differentiate between the new and original documents.

Let’s have a quick look at HTTP Method Summary for RESTful API

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Operation** | **Comment** |
| GET | Read Operation only | Uses only for the read operation.GET should be idempotent |
| POST | Create new resource | Should only be used to create a new resource |
| PUT | Update / Replace Resource | Update an existing resource.Think of PUT method as putting a resource |
| DELETE | Delete Resource | To remove a given resource.DELETE operation is idempotent |
| PATCH | Partial Update / Modify | Partial update to a resource should happen through PATCH |

## 6. HTTP Status Codes

REST API uses the Status-Line part of the HTTP response to inform API client of their request’s overarching result. While designing our REST API, it’s is very important to send a correct response back to the customer.Response code help client in

* Understanding status of the given request.
* It helps a client to take a correct step in the process.

As part of the RESTful design, we should clearly inform the client if they can move to next step or they need to take any corrective steps.HTTP defines over 40 standard status codes used to convey status back to the client.

A good status code also allows the developers to get their way out of the failed call.

All these status codes grouped into 5 different categories

|  |  |  |
| --- | --- | --- |
| **Status Code Category** | **Description** | **Example** |
| <span class="hiddenSpellError">1XX</span> - Informational | Informational indicates a provisional response | 100 (Continue ) , 101 |
| <span class="hiddenSpellError">2XX</span> - Successful | This class of status code indicates that the client’s request was successfully received, understood, and accepted. | 200 (OK), 201(Created), 202 (Accepted) |
| <span class="hiddenSpellError">3XX</span> - Redirection | This class of status code indicates that further action required by the user agent to fulfill the request | 301 (Moved Permanently), 302, 304 |
| <span class="hiddenSpellError">4XX</span> - Client Error | The 4xx class of status code is intended for cases where the client seems to have erred | 400 ( Bad Request), 401, 403, 404, 405 |
| <span class="hiddenSpellError">5XX</span> - Server Error | Response status codes beginning with the digit “5” tell cases where the server is aware that it has erred or is incapable of performing the request | 500 (Internal Server Error), 502, 503, 505 |

Before finishing this post, let’s cover some of the most ***commonly used*** ***HTTP statuses in the REST API***.

#### 6.1  200 (OK)

This status indicates the REST API successfully processes the request.200 response code include response body

* For GET request, 200 should contain requested resource.
* POST request should contain the result of the action performed (e.g new user-created).

#### 6.2  201 (Created)

REST API should response with 201 response code when a new resource created in the collection.

#### 6.3  202 (Accepted)

For the long-running process, REST API should return 202 as a valid response. 202 inform the client that request is accepted by the API but not yet completed.

#### 6.4  301 (Moved Permanently)

301 status tell that request REST API resource has been moved to a new URI. As part of the 301 response, REST API should also include the new URI for the customer.

#### 6.5  304 (Not Modified)

This indicates REST client that requested content has not been modified and if applicable, they can use cache copy of the content.Using this saves bandwidth and reprocessing on both the server and client.This is achieved by using added headers like If-Modified-Since or If-None-Match.

#### 6.6  400 (Bad Request)

This is a generic error which indicates REST client that request does not meet the required criteria.The client should not repeat the request without changes.

#### 6.7  401 (Unauthorized)

A 401 error response indicates that the client tried to run on a protected resource without providing the proper authorization.

#### 6.8 403 (Forbidden)

A 403 error response indicates that the client’s request formed correctly, but the REST API refuses to honor it i.e. the user does not have the necessary permissions for the resource

#### 6.9 404 (Not Found)

**https://www.dariawan.com/tutorials/rest/http-methods-spring-restful-services/**

**Representational state transfer** (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services (or simply **RESTful services**).

**RESTful** services enable us to develop any kind of application involving all possible CRUD (create, retrieve, update, delete) operations. We should utilize the different HTTP verbs which correspond to CRUD operations. The primary or most-commonly-used HTTP methods are GET, POST, PUT, PATCH, and DELETE. In performing these operations in **RESTful** services, there are guidelines or principles that suggest using a specific HTTP method on a specific type of call made to the server.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **HTTP Verb** | **CRUD** | **Sample** | **Description** | **Result** |
| **GET** | Read | /books | list all books. Pagination, sorting and filtering is used for big lists. | **200 (OK)** |
| /books/{id} | get specific book by id | **200 (OK)** **404 (Not Found)** - id not found or invalid |
| **POST** | Create | /books | create new book | **201 (Created)** - return a Location header with a link to the newly-created resource /books/{id}. **409 (Conflict)** - indicated that resource already exists |
| /books/id | Return **405 (Method Not Allowed)**, avoid using POST on single resource | |
| **PUT** | Update/Replace | /books | Return **405 (Method Not Allowed)**, unless you want to **replace every resource** in the entire collection of resource - use with caution. | |
| /books/{id} | Update a book | **200 (OK)** or **204 (No Content)** - indicate successful completion of the request **201 (Created)** - if new resource is created and creating new resource is allowed **404 (Not Found)** - if id not found or invalid and creating new resource is not allowed. |
| **PATCH** | Partial Update | /books | Return **405 (Method Not Allowed)**, unless you want to modify the collection itself.. | |
| /books/{id} | Partial update a book | **200 (OK)** or **204 (No Content)** - indicate successful completion of the request **404 (Not Found)** - if id not found or invalid |
| **DELETE** | Delete | /books | Return **405 (Method Not Allowed)**, unless you want to **delete the whole collection** - use with caution | |
| /books/{id} | Delete a book | **200 (OK)** or **204 (No Content)** or **202 (Accepted)** **404 (Not Found)** - if id not found or invalid |

## GET

The **GET** method used to retrieve information from the REST API and not to modify it in any way. GET should be a safe and idempotent method. Safe (and idempotent) methods can be cached.

* If the resource is found on the server then it must return HTTP response code 200 (OK) – along with response body (can be XML, JSON or (possible) RAW)
* In case resource is NOT found on server then it must return 404 (Not Found).
* In case that GET request itself is not correctly formed then server will return 400 (Bad Request)..

## POST

**POST** should only be used to create a new resource. In some cases, POST method is used to update entity but we have more proper HTTP Method to do this (PUT). POST API call is not idempotent nor safe and should be used with care. Invoking two identical POST requests will result in creation of two different resources with same information but different resource id

* If a resource is created on the server, the response ideally should be HTTP response code 201 (Created), a Location header (URI) and contain an entity which describes the status of the request and refer to the new resource
* In case of successful creation but the resource can't be identified by a URI, then use either 200 (OK) or 204 (No Content).
* If the resource (after certain validation) is deemed as already exists, then return 409 (Conflict)
* Is suggested to use POST method for collection and to return 405 (Method Not Allowed) on single resource

Result for this method is not cacheable. We need to include Cache-Control or Expires header if we want to use cache for POST method. In relation with GET with cached result, POST (and PUT, PATCH, and DELETE) should invalidate the cache.

## PUT

The **PUT** method is used to update existing resource. PUT method is idempotent, client can send the same request multiple time, and it will have same result as sending once.

* In case of successful update then use HTTP response code 200 (OK) or 204 (No Content). 204 (No Content) normally is use if the action has been performed but the response does not include an entity (applied too for PATCH and DELETE). Personally, I choose 200 (OK) indicating a successful update.
* In case that the resource doesn't exist, then either create a new resource or reject the PUT request by return 404 (Not Found). Personally I choose to return 404 (Not Found) since more proper HTTP Method POST is used to create new resource - try to make it consistent.
* If a new resource created by PUT request, the response should be 201 (Created)

If PUT try to modify cached resources, then the cache must be invalidated. Responses to this method are **not cacheable**.

## PATCH

**PATCH** requests are used to make partial update on a resource. A PATCH request is one of the lesser-known HTTP methods. Support for PATCH in browsers, servers and web applications are not universal, this can post some challenges. Request payload for PATCH is can be challenging and, but the common approach is to send a delta (diff) rather than the entire resource.

* In case of successful update then use HTTP response code 200 (OK) or 204 (No Content). Same as PUT, I choose 200 (OK) indicating a successful update.
* In case that the resource doesn't exist, return 404 (Not Found). Avoid creation of new resource based on partial data/information.

If PATCH try to modify cached resources, then the cache must be invalidated. Responses to this method are **not cacheable**. In my example below, I only using String as my payload to update title.

## DELETE

**DELETE** method are used to delete a resource identified by the request URI.

* A successful deletion should return HTTP response code 200 (OK) or 204 (No Content). DELETE can be a long-running or asynchronous request, so return 202 (Accepted) if the action has been queued.
* In case that the resource doesn't exist, return 404 (Not Found)

DELETE operations are **idempotent**. If you DELETE a resource, it’s removed from the collection. Although calling DELETE on a resource on subsequent time will return a 404 (Not Found), the resource still (or already) removed from collection.

If DELETE try to delete a cached resource, then the cache must be invalidated. Responses to this method are **not cacheable**.

**---------------------------------------------------------------------------------------------**

**Explain what is REST and RESTFUL?**

REST represents REpresentational State Transfer; it is a relatively new aspect of writing web API.

RESTFUL is referred for web services written by applying REST architectural concept are called RESTful services, it focuses on system resources and how state of resource should be transported over HTTP protocol to different clients written in different language. In RESTFUL web service HTTP methods like GET, POST, PUT and DELETE can be used to perform CRUD operations.

**Mention what are the HTTP methods supported by REST?**

HTTP methods supported by REST are:

* **GET:** It retrives the data. It does’t have a RequestBody
* **POST:**It creates a resource. It has a RequestBody
* **PUT:**At the request URL it update the resource
* **DELETE:**At the request URL it removes the resource
* **OPTIONS:**It indicates which techniques are supported
* **HEAD:**About the request URL it returns meta information

**Mention whether you can use GET request instead of PUT to create a resource?**

* No, you are not supposed to use PUT for GET. GET operations should only have view rights, while PUT resource is used for updating a data.

**Mention what are resources in a REST architecture?**

* Resources are identified by logical URLs; it is the key element of a RESTful design. Unlike, SOAP web services in REST, you view the product data as a resource and this resource should contain all the required information.
* A resource in REST is **a similar Object in Object Oriented Programming or is like an Entity in a Database**.

**Mention some key characteristics of REST?**

1. **It is stateless**
2. **It supports JSON and XML**
3. **It is simple than SOAP**
4. **Documentation**
5. **Error messages**
6. Lightweight
7. Independent.

Some key characteristics of REST includes

* REST is stateless, therefore the SERVER has no state (or session data)
* With a well-applied REST API, the server could be restarted between two calls as every data is passed to the server
* Web service mostly uses POST method to make operations, whereas REST uses GET to access resources

**Mention what is the difference between PUT and POST?**

* “PUT” puts a file or resource at a particular URI and exactly at that URI. If there is already a file or resource at that URI, PUT changes that file or resource. If there is no resource or file there, PUT makes one
* POST sends data to a particular URI and expects the resource at that URI to deal with the request. The web server at this point can decide what to do with the data in the context of specified resource
* PUT is idempotent meaning, invoking it any number of times will not have an impact on resources.
* However, POST is not idempotent, meaning if you invoke POST multiple times it keeps creating more resources

**Mention what is the difference between SOAP and REST?**

* SOAP stands for Simple Object Access Protocol whereas REST stands for Representational State Transfer.
* SOAP is a protocol whereas REST is an architectural pattern.
* SOAP uses service interfaces to expose its functionality to client applications while REST uses Uniform Service locators to access to the components on the hardware device.
* SOAP needs more bandwidth for its usage whereas REST doesn’t need much bandwidth.
* Comparing SOAP vs REST API, SOAP only works with XML formats whereas REST work with plain text, XML, HTML and JSON.
* SOAP cannot make use of REST whereas REST can make use of SOAP.

**Mention which markup language can be used in restful web api?**

JSON and XML are the two markup language that can be used in restful web api

**What is URI?**

* A Uniform Resource Identifier (URI) is a string of characters used to identify a name or a resource on the Internet.

A URI identifies a resource either by location, or a name, or both. **A URI has two specializations known as URL and URN.**

* A Uniform Resource Locator (URL) is a subset of the Uniform Resource Identifier (URI) that specifies where an identified resource is available and the mechanism for retrieving it. A URL defines how the resource can be obtained. It does not have to be a HTTP URL (http://), a URL can also start with ftp:// or smb://, specifying the protocol that's used to get the resource.
* A Uniform Resource Name (URN) is a Uniform Resource Identifier (URI) that uses the URN scheme, and **does not imply availability of the identified resource**. Both URNs (names) and URLs (locators) are URIs, and a particular URI may be both a name and a locator at the same time.

### What are the features of RESTful Web Services?

Every RESTful web service has the following features:

* The service is based on the Client-Server model.
* The service uses HTTP Protocol for fetching data/resources, query execution, or any other functions.
* The medium of communication between the client and server is called “Messaging”.
* Resources are accessible to the service by means of URIs.
* It follows the statelessness concept where the client request and response are not dependent on others and thereby provides total assurance of getting the required data.
* These services also use the concept of caching to minimize the server calls for the same type of repeated requests.
* These services can also use SOAP services as implementation protocol to REST architectural pattern

### What is the concept of statelessness in REST?

The REST architecture is designed in such a way that the client state is not maintained on the server. This is known as statelessness. The context is provided by the client to the server using which the server processes the client’s request. The session on the server is identified by the session identifier sent by the client.

### What are HTTP Status codes?

These are the standard codes that refer to the predefined status of the task at the server. Following are the status codes formats available:

* 1xx - represents informational responses
* 2xx - represents successful responses
* 3xx - represents redirects
* 4xx - represents client errors
* 5xx - represents server errors

Most commonly used status codes are:

* 200 - success/OK
* 201 - CREATED - used in POST or PUT methods.
* 304 - NOT MODIFIED - used in conditional GET requests to reduce the bandwidth use of the network. Here, the body of the response sent should be empty.
* 400 - BAD REQUEST - This can be due to validation errors or missing input data.
* 401- UNAUTHORIZED - This is returned when there is no valid authentication credentials sent along with the request.
* 403 - FORBIDDEN - sent when the user does not have access (or is forbidden) to the resource.
* 404 - NOT FOUND - Resource method is not available.
* 500 - INTERNAL SERVER ERROR - server threw some exceptions while running the method.
* 502 - BAD GATEWAY - Server was not able to get the response from another upstream server

### What are the HTTP Methods?

HTTP Methods are also known as HTTP Verbs. They form a major portion of uniform interface restriction followed by the REST that specifies what action has to be followed to get the requested resource. Below are some examples of HTTP Methods:

* GET: This is used for fetching details from the server and is basically a read-only operation.
* POST: This method is used for the creation of new resources on the server.
* PUT: This method is used to update the old/existing resource on the server or to replace the resource.
* DELETE: This method is used to delete the resource on the server.
* PATCH: This is used for modifying the resource on the server.
* OPTIONS: This fetches the list of supported options of resources present on the server.

The POST, GET, PUT, DELETE corresponds to the create, read, update, delete operations which are most commonly called **CRUD**

GET, HEAD, OPTIONS are safe and idempotent methods whereas PUT and DELETE methods are only idempotent. POST and PATCH methods are neither safe nor idempotent.

### Can you tell the disadvantages of RESTful web services?

The disadvantages are:

* As the services follow the idea of statelessness, it is not possible to maintain sessions. (Session simulation responsibility lies on the client-side to pass the session id)
* REST does not impose security restrictions inherently. It inherits the security measures of the protocols implementing it. Hence, care must be chosen to implement security measures like integrating SSL/TLS based authentications, etc.

### Define Messaging in terms of RESTful web services.

The technique of sending a message from the REST client to the REST server in the form of an HTTP request and the server responding back with the response as HTTP Response is called Messaging. The messages contained constitute the data and the metadata about the message.

### While creating URI for web services, what are the best practices that needs to be followed?

### **URI = scheme "://" authority "/" path [ "?" query ] [ "#" fragment ]**

###### A trailing forward slash (/) should not be included in URIs

###### Forward slash separator (/) must be used to indicate a hierarchical relationship

###### Hyphens (-) should be used to improve the readability of URIs

###### Underscores (\_) should not be used in URIs

###### Lowercase letters should be preferred in URI paths

###### File extensions should not be included in URIs

###### While defining resources, use plural nouns. Example: To identify user resource, use the name “users” for that resource.

###### Use appropriate HTTP methods like GET, PUT, DELETE, PATCH, etc. It is not needed or recommended to use these method names in the URI. Example: To get user details of a particular ID, use /users/{id} instead of /getUser

### What are the best practices to develop RESTful web services?

* Rest API must accept and Respond with JSON
* Always go with error status codes
* Don’t use verbs in urls

**Don't use** -> POST: /articles/createNewArticle/

**Do use** -> POST: /articles/

* Use Plural Nouns to name a collection

GET /cars/123

POST /cars

GET /cars

* Well compiled documentation
* Return error details in the response body.

{

"error": "Invalid payoad.",

"detail": {

"surname": "This field is required."

}

}

* User Resource nesting

/users // list all users

/users/123 // specific user

/users/123/orders // list of orders that belong to a specific user

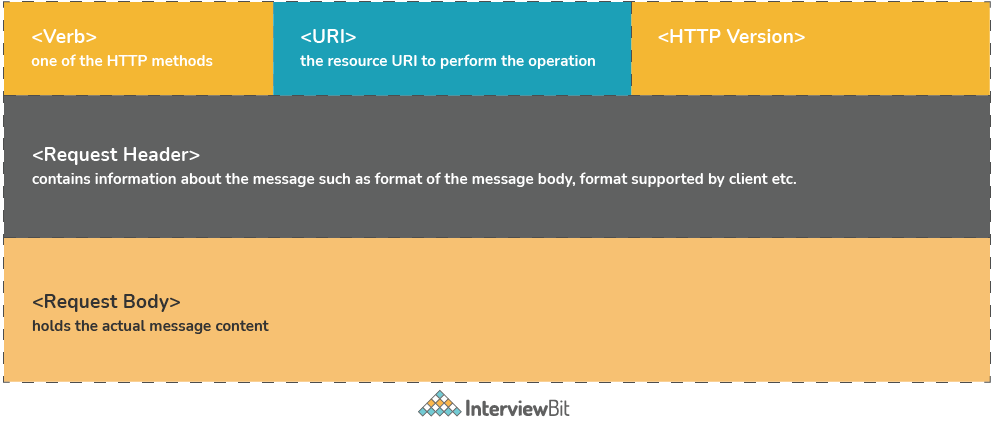
/users/123/orders/0001 // specific order of a specific users order list

* Use SSL/TLS
* Secure the APIs

### Can you tell what constitutes the core components of HTTP Request?

In REST, any HTTP Request has 5 main components, they are:

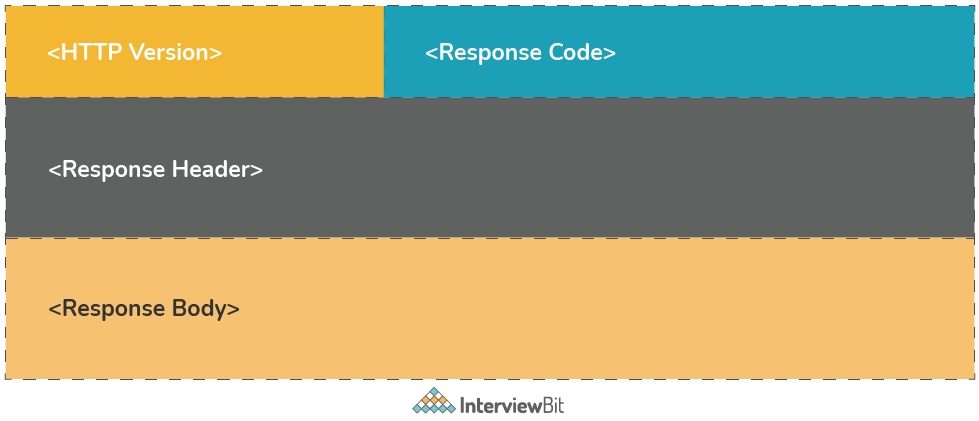
* Method/Verb − This part tells what methods the request operation represents. Methods like GET, PUT, POST, DELETE, etc are some examples.
* URI − This part is used for uniquely identifying the resources on the server.
* HTTP Version − This part indicates what version of HTTP protocol you are using. An example can be HTTP v1.1.
* Request Header − This part has the details of the request metadata such as client type, the content format supported, message format, cache settings, etc.
* Request Body − This part represents the actual message content to be sent to the server.



### What constitutes the core components of HTTP Response?

HTTP Response has 4 components:

* Response Status Code − This represents the server response status code for the requested resource. Example- 400 represents a client-side error, 200 represents a successful response.
* HTTP Version − Indicates the HTTP protocol version.
* Response Header − This part has the metadata of the response message. Data can describe what is the content length, content type, response date, what is server type, etc.
* Response Body − This part contains what is the actual resource/message returned from the server.



**Define Addressing in terms of RESTful Web Services.**

Addressing is the process of locating a single/multiple resources that are present on the server. This task is accomplished by making use of URI (Uniform Resource Identifier). The general format of URI is

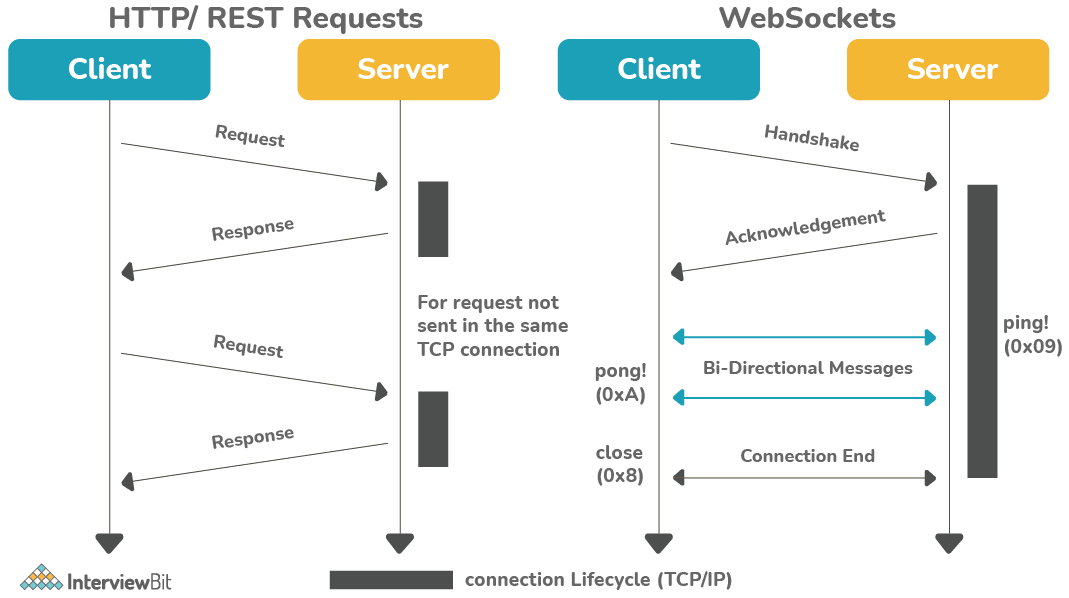
<protocol>://<application-name>/<type-of-resource>/<id-of-resource>

**What are the differences between PUT and POST in REST?**

| **PUT** | **POST** |
| --- | --- |
| PUT methods are used to request the server to store the enclosed entity in request. In case, the request does not exist, then new resource has to be created. If the resource exists, then the resource should get updated. | POST method is used to request the server to store the enclosed entity in the request as a new resource. |
| The URI should have a resource identifier. Example: PUT /users/{user-id} | The POST URI should indicate the collection of the resource. Example: POST /users |
| PUT methods are idempotent. | POST methods are not idempotent. |
| PUT is used when the client wants to modify a single resource that is part of the collection. If a part of the resource has to be updated, then PATCH needs to be used. | POST methods are used to add a new resource to the collection. |
| The responses are not cached here despite the idempotency. | Responses are not cacheable unless the response explicitly specifies Cache-Control fields in the header. |
| In general, PUT is used for UPDATE operations. | POST is used for CREATE operations. |

**We can develop webservices using web sockets as well as REST. What are the differences between these two?**

| **REST** | **Web Socket** |
| --- | --- |
| REST follows stateless architecture, meaning it won’t store any session-based data. | Web Socket APIs follow the stateful protocol as it necessitates session-based data storage. |
| The mode of communication is uni-directional. At a time, only the server or the client will communicate. | The communication is bi-directional, communication can be done by both client or server at a time. |
| REST is based on the Request-Response Model. | Web Socket follows the full-duplex model. |
| Every request will have sections like header, title, body, URL, etc. | Web sockets do not have any overhead and hence suited for real-time communication. |
| For every HTTP request, a new TCP connection is set up. | There will be only one TCP connection and then the client and server can start communicating. |
| REST web services support both vertical and horizontal scaling. | Web socket-based services only support vertical scaling. |
| REST depends on HTTP methods to get the response. | Web Sockets depend on the IP address and port number of the system to get a response. |
| Communication is slower here. | Message transmission happens very faster than REST API. |
| Memory/Buffers are not needed to store data here. | Memory is required to store data. |



### Can we implement transport layer security (TLS) in REST?

Yes, we can. TLS does the task of encrypting the communication between the REST client and the server and provides the means to authenticate the server to the client. It is used for secure communication as it is the successor of the Secure Socket Layer (SSL). HTTPS works well with both TLS and SSL thereby making it effective while implementing RESTful web services. One point to mention here is, the REST inherits the property of the protocol it implements. So security measures are dependent on the protocol REST implements

### What is Payload in terms of RESTful web services?

Payload refers to the data passes in the request body. It is not the same as the request parameters. The payload can be sent only in POST methods as part of the request body.

### Is it possible to send payload in the GET and DELETE methods?

No, the payload is not the same as the request parameters. Hence, it is not possible to send payload data in these method

### How does HTTP Basic Authentication work?

While implementing Basic Authentication as part of APIs, the user must provide the username and password which is then concatenated by the browser in the form of “username: password” and then perform base64 encoding on it. The encoded value is then sent as the value for the “Authorization” header on every HTTP request from the browser. Since the credentials are only encoded, it is advised to use this form when requests are sent over HTTPS as they are not secure and can be intercepted by anyone if secure protocols are not used.

### What is the difference between idempotent and safe HTTP methods?

* Safe methods are those that do not change any resources internally. These methods can be cached and can be retrieved without any effects on the resource.
* Idempotent methods are those methods that do not change the responses to the resources externally. They can be called multiple times without any change in the responses.

According to [restcookbook.com](https://restcookbook.com/), the following is the table that describes what methods are idempotent and what is safe.

| **HTTP Methods** | **Idempotent** | **Safe** |
| --- | --- | --- |
| OPTIONS | yes | yes |
| GET | yes | yes |
| HEAD | yes | yes |
| PUT | yes | no |
| POST | no | no |
| DELETE | yes | no |
| PATCH | no | no |

### What is the use of @RequestMapping?

* The annotation is used for mapping requests to specific handler classes or methods.
* In spring, all the incoming web request routing is handled by Dispatcher Servlet. When it gets the request, it determines which controller is meant for processing the request by means of request handlers. The Dispatcher Servlet scans all the classes annotated with @Controller. The process of routing requests depends on @RequestMapping annotations that are declared inside the controller classes and their methods.

### 39. What are the differences between the annotations @Controller and @RestController?

| **@Controller** | **@RestController** |
| --- | --- |
| Mostly used traditional Spring MVC service. | Represents RESTful web service in Spring. |
| It is mostly used in Spring MVC service where model data needs to rendered using view. | It is used in case of RESTful web service that returns object values bound to response body. |
| If response values need to be converted through HttpMessageConverters and sent via response object, extra annotation @ResponseBody needs to be used on the class or the method handlers. | The default behavior of the @RestController needs to be written on the response body because it is the combination of @Controller and @ResponseBody. |
| @Controller provides control and flexibility over how the response needs to be sent. | @RestController annotation has no such flexibility and writes all the results to the response body. |

### 40. What does the annotation @PathVariable do?

@PathVariable annotation is used for passing the parameter with the URL that is required to get the data. Spring MVC provides support for URL customization for data retrieval using @PathVariable annotation.