



# Lesson 24

29.02.2024

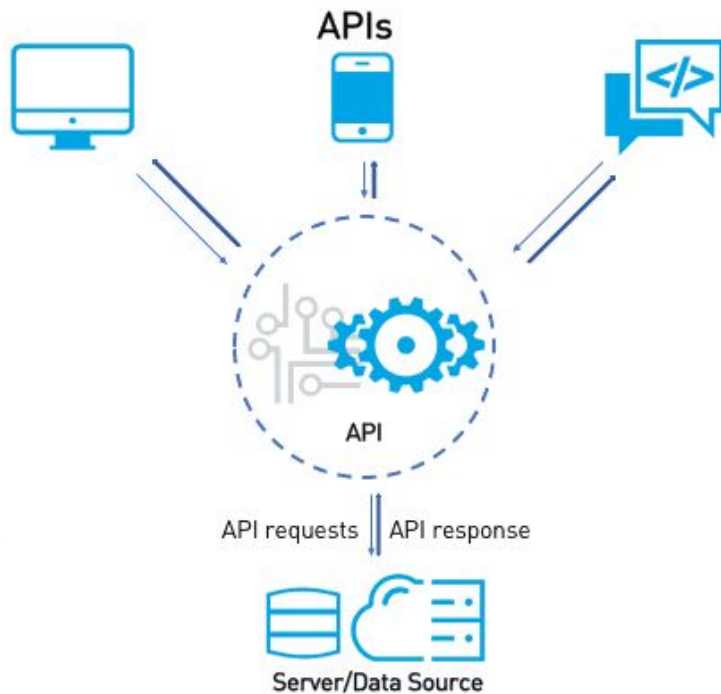
```
public class Ex1 {  
    public static void main(String[] args) {  
        boolean flag1 = "Java" == "Java"  
            .replace( 'J' , 'J' );  
        boolean flag2 = "Java" == "Java"  
            .replace( 'J' , 'J' );  
        System.out.println(flag1);  
        System.out.println(flag2);  
        System.out.println(flag1 && flag2);  
    }  
}
```

```
public class Ex2 {  
    public static void main(String[] args) {  
        String s1 = new String( "Java" );  
        String s2 = "JaVa" ;  
        String s3 = "JaVa" ;  
        String s4 = "Java" ;  
        String s5 = "Java" ;  
        int i = 1 ;  
        Integer j = 122 ;  
        Integer k = 129 ;  
        Integer m = 128 ;  
    }  
}
```

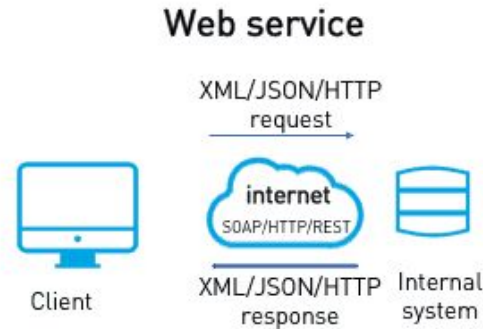
```
public class Ex3 {  
    public static void main(String[] args) {  
        int sum = 0;  
        for (int i = 0; i < 7; i++) {  
            if (i == 4)  
                break;  
            else  
                continue;  
            sum += i;  
        }  
        System.out.println(sum);  
    }  
}
```

```
public class Ex4 {  
    public static void main(String[] args) {  
        String str = "BEVERAGE" ;  
        String [] arr = str.split( "E" , 3 );  
        System.out.println(String.join ( "." , arr));  
    }  
}
```

```
public class Ex5 {  
    public static void main(String[] args) {  
        Boolean [] arr = new Boolean[ 2 ];  
        List<Boolean> list = new ArrayList<>();  
        list.add(arr[ 0 ]);  
        list.add(arr[ 1 ]);  
        if (list.remove( 0 )) {  
            list.remove( 1 );  
        }  
        System.out.println(list);  
    }  
}
```

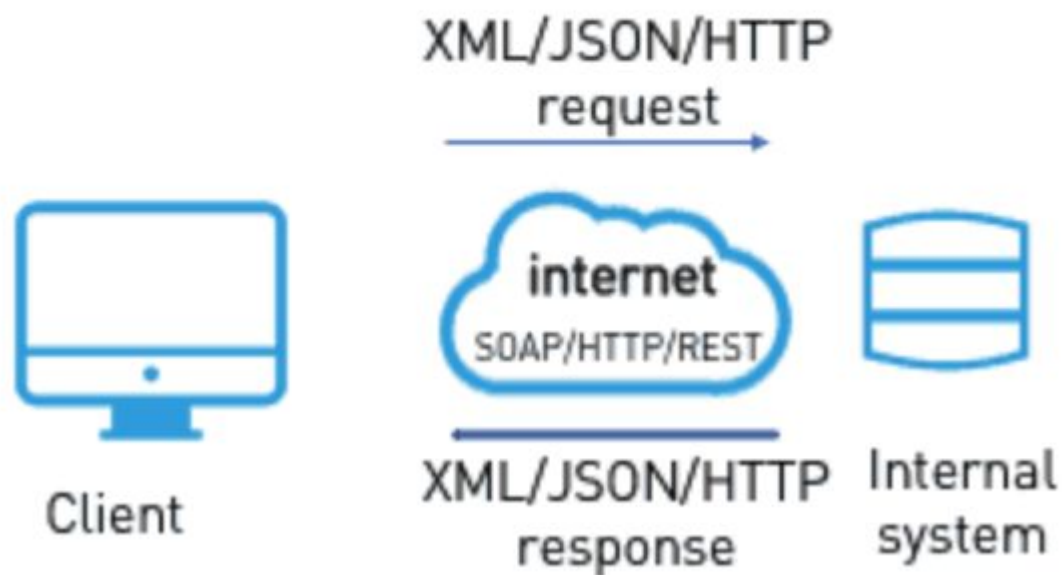


**API** - це аббревіатура від інтерфейсу прикладного програмування. Це програмний інтерфейс, який дозволяє двом програмам взаємодіяти одна з одною без втручання користувача. API надає продукт або послугу для зв'язку з іншими продуктами та послугами, не знаючи, як вони реалізовані.



Веб-служба — це набір відкритих протоколів і стандартів, які широко використовуються для обміну даними між системами або програмами. Програмні програми написані з використанням різних мов програмування та працюють на кількох платформах. Він дозволяє використовувати веб-сервіси для обміну даними через комп'ютерні мережі

# Web service

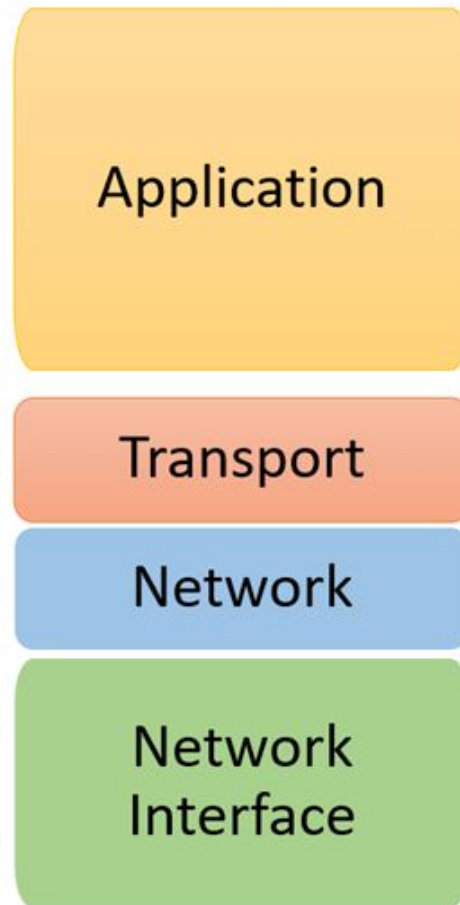


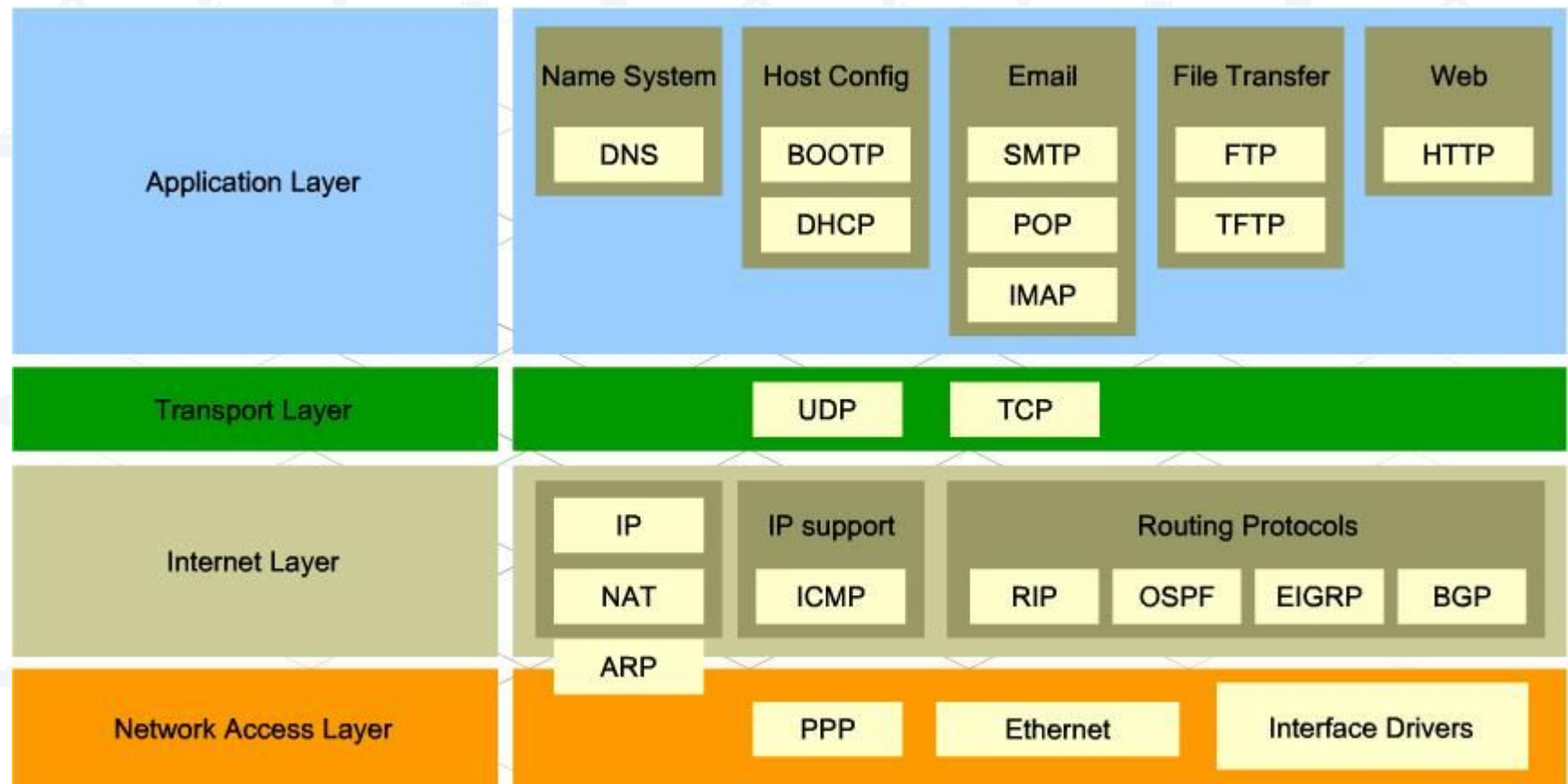


## OSI Reference Model

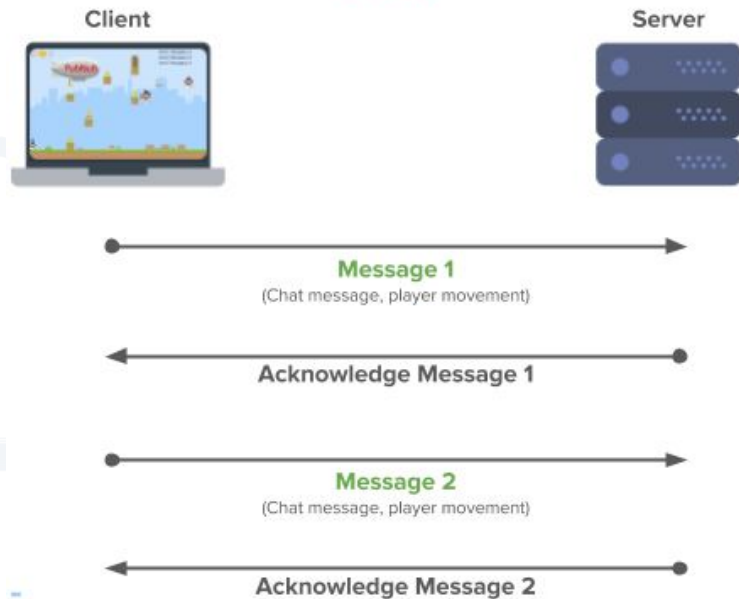


## TCP/IP Conceptual Layers

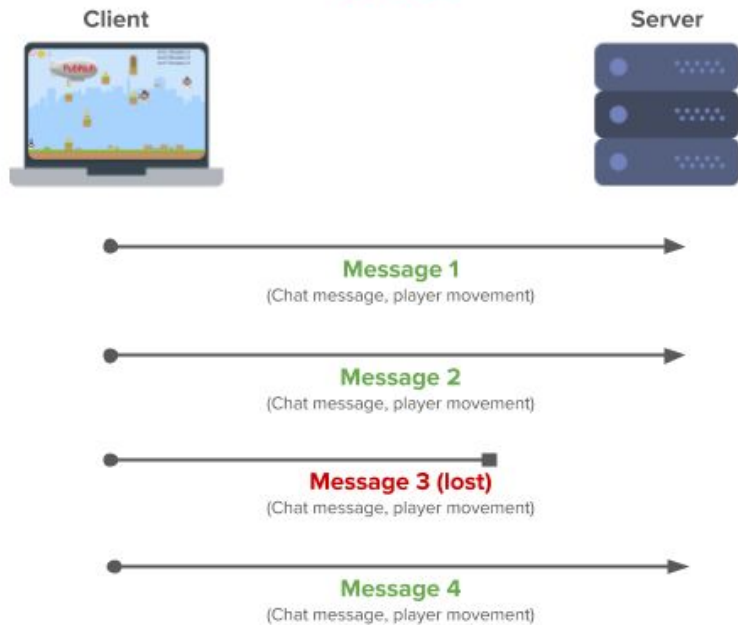




# TCP

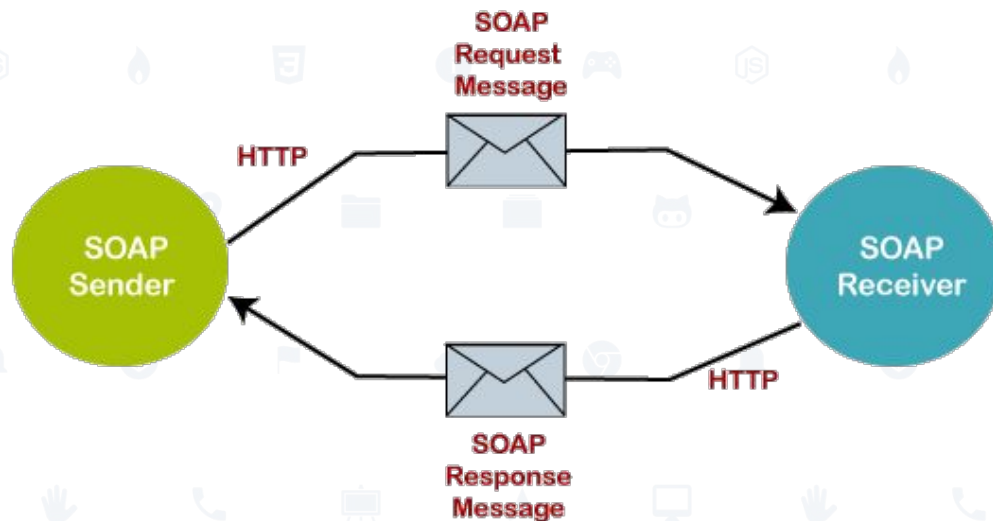


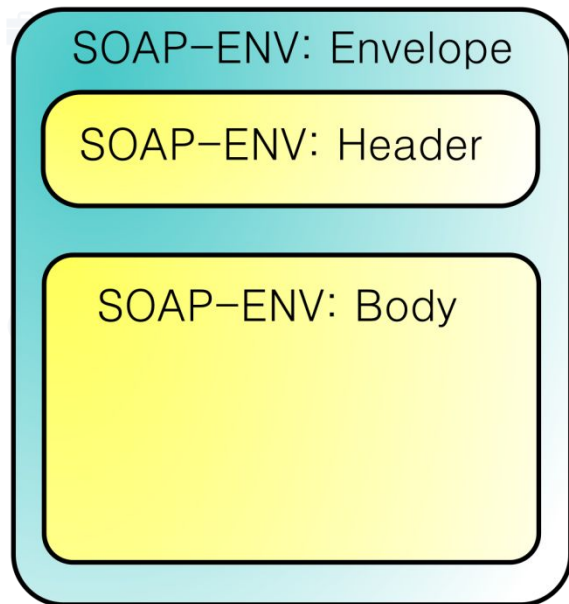
# UDP



SOAP (від англ. Simple Object Access Protocol - простий протокол доступу до об'єктів) - протокол обміну структурованими повідомленнями в розподіленому обчислювальному середовищі. Спочатку SOAP призначався переважно реалізації віддаленого виклику процедур (RPC). Наразі протокол використовується для обміну довільними повідомленнями у форматі XML, а не лише для виклику процедур. Офіційна специфікація останньої версії 1.2 протоколу не розшифровує назву SOAP. SOAP є розширенням протоколу XML-RPC.

SOAP може використовуватися з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTP, HTTPS та ін. Однак його взаємодія з кожним із цих протоколів має свої особливості, які мають бути визначені окремо. Найчастіше SOAP використовується поверх HTTP.





**Envelope** – Кореневий елемент, який визначає повідомлення та простір імен, використаний у документі.

**Header** – Містить атрибути повідомлення, наприклад: інформація про безпеку або мережеву маршрутизацію.

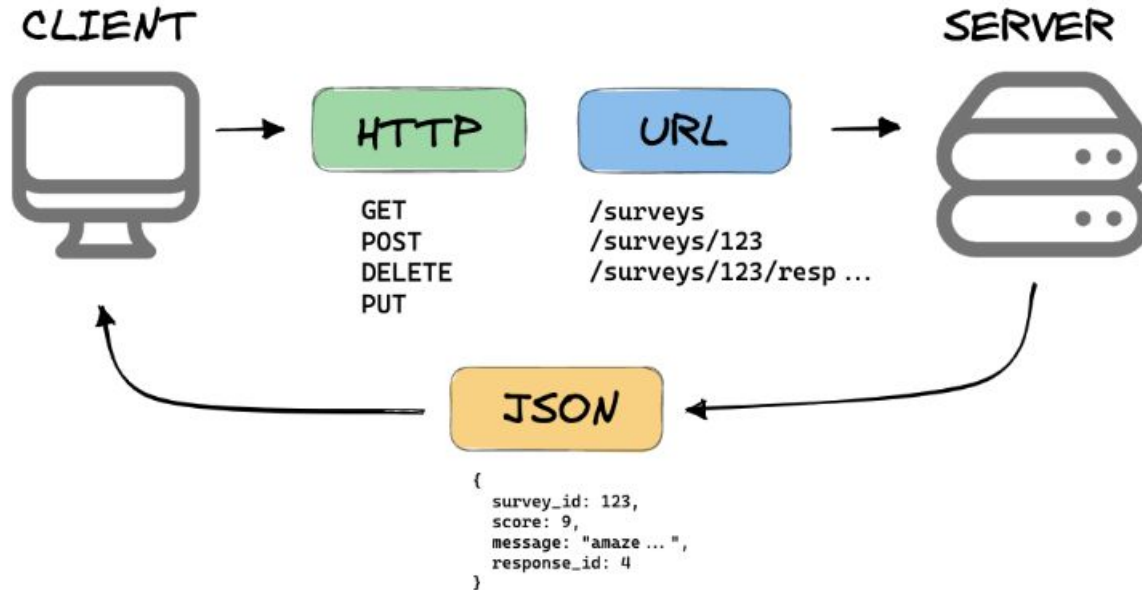
**Body** – Містить повідомлення, яким обмінюються програми.

**Fault** – Необов'язковий елемент, який надає інформацію про помилки, що сталися під час обробки повідомлень

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetails xmlns="http://warehouse.example.com/ws">
      <productID>12345</productID>
    </getProductDetails>
  </soap:Body>
</soap:Envelope>
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
      <getProductDetailsResult>
        <productID>12345</productID>
        <productName>Стакан граненый</productName>
        <description>Стакан граненый. 250 мл.</description>
        <price>9.95</price>
        <currency>
          <code>840</code>
          <alpha3>USD</alpha3>
          <sign>$</sign>
          <name>US dollar</name>
          <accuracy>2</accuracy>
        </currency>
        <inStock>true</inStock>
      </getProductDetailsResult>
    </getProductDetailsResponse>
  </soap:Body>
</soap:Envelope>
```


# WHAT IS A REST API?



## REST

(REpresentational State Transfer) – це архітектура, тобто. принципи побудови розподілених гіпермедіа систем, що називається World Wide Web, включаючи універсальні методи обробки та передачі станів ресурсів по HTTP

Автор ідеї та терміна Рой Філдінг 2000р.



## RESTful критерії:

**Client-Server.** Система має бути поділена на клієнтів та на серверів. Поділ інтерфейсів означає, що, наприклад, клієнти не пов'язані зі зберіганням даних, яке залишається всередині кожного сервера, тому мобільність коду клієнта покращується. Сервери не пов'язані з інтерфейсом користувача або станом, так що сервери можуть бути простішими і масштабованішими. Сервери та клієнти можуть бути замінені та розроблятися незалежно, доки інтерфейс не змінюється.

**Stateless.** Сервер не повинен зберігати будь-яку інформацію про клієнтів. У запиті повинна зберігатись вся необхідна інформація для обробки запиту і, якщо необхідно, ідентифікації клієнта.

**Cache.** Кожна відповідь має бути позначена, чи вона кешується чи ні, для запобігання повторному використанню клієнтами застарілих або некоректних даних у відповідь на подальші запити.

**Uniform Interface.** Єдиний інтерфейс визначає інтерфейс між клієнтами та серверами. Це спрощує та відокремлює архітектуру, яка дозволяє кожній частині розвиватися самостійно.

**Layered System.** У REST допускається розділити систему на ієрархію шарів але з умовою, кожен компонент може бачити компоненти лише безпосередньо наступного шару. Наприклад:





Presentation Layer

Component

Component

Component

Business Layer

Component

Component

Component

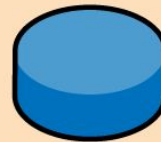
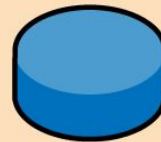
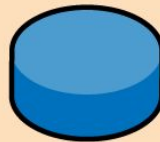
Persistence Layer

Component

Component

Component

Database Layer



## SAFE METHODS

NO ACTION ON SERVER

MESSAGE WITH

BODY

SEND DATA TO SERVER

GET

HEAD

PUT

POST

PATCH

TRACE

OPTIONS

DELETE

HTTP/1.1 MUST IMPLEMENT THIS METHOD

INSPECT RESOURCE HEADERS

DEPOSIT DATA ON SERVER — INVERSE OF GET

SEND INPUT DATA FOR PROCESSING

PARTIALLY MODIFY A RESOURCE

ECHO BACK RECEIVED MESSAGE

SERVER CAPABILITIES

DELETE A RESOURCE — NOT GUARANTEED

# HTTP STATUS CODES



## HTTP Status Codes

javaconceptoftheday.com

1xx : Informational Purpose		4xx : Client Errors		5xx : Server Errors	
100	Continue	400	Bad Request	500	Internal Server Error
101	Switching Protocols	401	Unauthorized	501	Not Implemented
102	Processing	402	Payment Required	502	Bad Gateway
103	Early Hints	403	Forbidden	503	Service Unavailable
2xx : Success		404	Not Found	504	Gateway Timeout
200	Ok	405	Method Not Allowed	505	HTTP Version Not Supported
201	Created	406	Not Acceptable	507	Insufficient Storage
202	Accepted	407	Proxy Authentication Is Required	508	Loop Detected
203	Non-Authoritative Information	408	Request Time Out	510	Not Extended
204	No Content	409	Conflict	511	Network Authentication Required
205	Reset Content	410	Gone		
206	Partial Content	411	Length Required		
207	Multi Status	412	Precondition Failed		
208	Already Reported	413	Payload Too Large		
226	IM Used	414	URI Too Long		
3xx : Redirection		415	Unsupported Media Type		
300	Multiple Choices	416	Range Not Satisfiable		
301	Moved Permanently	417	Expectation Failed		
302	Found	421	Misdirect Request		
303	See Other	422	Unprocessable Entity		
304	Not Modified	423	Locked		
305	Use Proxy	424	Failed Dependency		
306	No Longer Used	425	Too Early		
307	Temporary Redirect	426	Upgrade Required		
308	Moved Permanently	428	Precondition Required		
		429	Too Many Requests		
		431	Request Header Fields Too Large		
		451	Unavailable For Legal Reasons		



HTTP метод **GET** використовується для отримання (або читання) даних ресурсу. У разі "вдалої" (або не містить помилок) адреси, GET повертається подання ресурсу у форматі XML або JSON у поєднанні з кодом стану HTTP 200 (OK). У разі помилок зазвичай повертається код 404 (NOT FOUND) чи 400 (BAD REQUEST).

**[GET] <http://www.example.com/api/v1.0/users>**

(Повернути список користувачів)

**[GET] <http://www.example.com/api/v1.0/users/12345>**

(Повернути дані про користувача з id 12345)

HTTP метод **PUT** зазвичай використовується для надання можливості оновлення ресурсу. Тіло запиту при надсиланні PUT-запиту до існуючого ресурсу URI має містити оновлені дані оригінального ресурсу (повністю або тільки оновлювану частину).

При успішному оновленні за допомогою виконання PUT запиту повертається код 200 PUT не безпечна операція, так як внаслідок її виконання відбувається модифікація (або створення) екземплярів ресурсу на стороні сервера, але цей метод ідемпотентний. Іншими словами, створення або оновлення ресурсу за допомогою відправки PUT запиту - ресурс не зникне, буде розташовуватися там же, де і був при першому зверненні, а також багаторазове виконання одного і того ж PUT запиту не змінить загального стану системи

**[PUT] <http://www.example.com/api/v1.0/users/12345>**

(оновити дані користувача з id 12345)

**[PUT] <http://www.example.com/api/v1.0/users/12345/orders/98765>**

(оновити дані замовлення з id 98765 для користувача з id 12345)



HTTP метод **POST** запит найчастіше використовується **для створення нових ресурсів**. Насправді він використовується до створення вкладених ресурсів. Іншими словами, при створенні нового ресурсу, POST запит відправляється до батьківського ресурсу і, таким чином, сервіс бере на себе відповідальність на встановлення зв'язку ресурсу, що створюється з батьківським ресурсом, призначення новому ресурсу ID і т.п.

При успішному створенні ресурсу повертається HTTP код 201, а також у заголовку `Location` передається адреса створеного ресурсу.

POST не є безпечним чи ідемпотентним запитом. Тому рекомендується його використання для не ідемпотентних запитів. В результаті виконання ідентичних запитів POST надаються дуже схожі, але не ідентичні дані.

**[POST] <http://www.example.com/api/v1.0/customers>**

(Створити новий ресурс у розділі customers)

**[POST] <http://www.example.com/api/v1.0/customers/12345/orders>**

(Створити замовлення для ресурсу з id 12345)



HTTP метод **DELETE** використовується **видалення ресурсу**, ідентифікованого конкретним URI (ID).

При успішному видаленні повертається 200 (OK) код HTTP, разом із тілом відповіді, що містить дані віддаленого ресурсу. Також можливе використання коду HTTP 204 (NO CONTENT) без тіла відповіді. Якщо ви виконуєте DELETE запит на ресурс, він видалається. Повторний DELETE запит до ресурсу закінчиться: ресурс видалено.

Проте, існує застереження щодо ідемпотентності DELETE. Повторний DELETE запит до ресурсу часто супроводжується 404 (NOT FOUND) кодом HTTP через те, що ресурс вже видалений (наприклад з бази даних) і більше не доступний. Це робить DELETE операцію не ідемпотентною, але це загальноприйнятий компроміс на той випадок, якщо ресурс був видалений з бази даних, а не помічений як віддалений.

**[DELETE] <http://www.example.com/api/v1.0/customers/12345>**

(Видалити з customers ресурс з id 12345)

**[DELETE] <http://www.example.com/api/v1.0/customers/12345/orders/21>**

(Видалити у ресурсу з id 12345 замовлення з id 21)

# Java Developer Career Path

---

