**Spring Web**

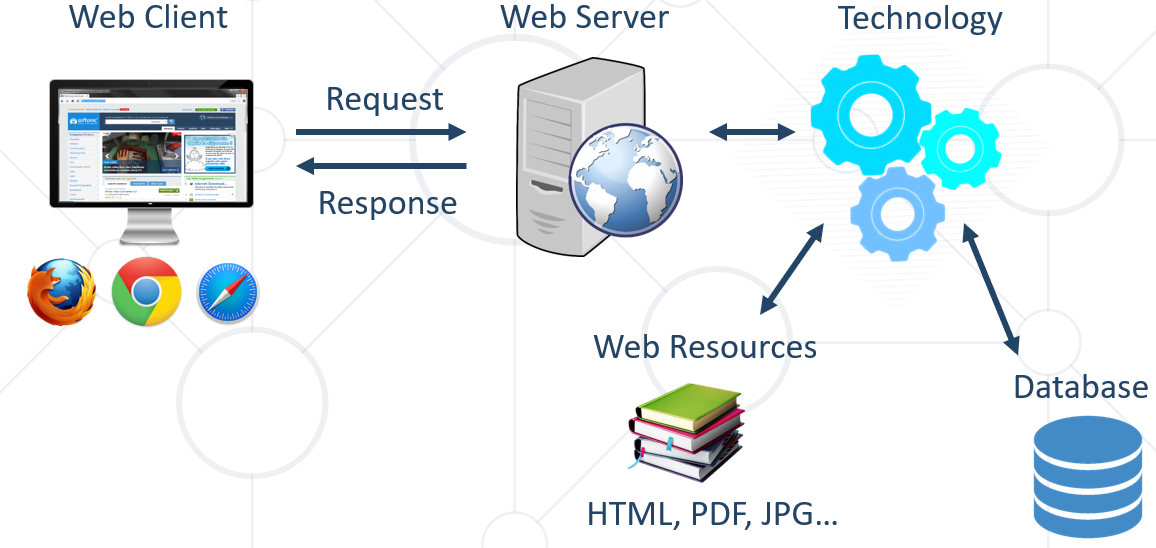
# 1. Internet Explained

## 1.1. Internet – History

* Begins with the development of electronic computers in the 1950s.
* **Packet** switching networks were developed in the late 1960s
* The **internet protocol** was developed in the 1970s
* In the 1980s at CERN Tim Berners-Lee created the [**World Wide Web**](http://info.cern.ch/hypertext/WWW/TheProject.html) – the first website, linking hypertext documents into an information system, accessible from any node on the network

## 1.2. How Does the Internet Work?

### Web Server Work Model

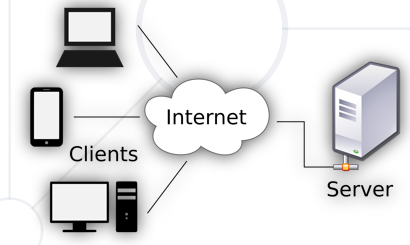


### Important Definitions

* + To understand how **the Internet works**, first we need   
    to get acquainted with a few definitions
  + **What is?**
    - Server and Client
    - Network Protocol
    - Packets
    - TCP vs UDP

### Servers and Clients

* + All of the machines on the Internet are either **servers** or **clients**
  + **Servers** are the machines that provide services to other machines
  + **Clients** are the machines that are used to connect to those services



### Network Protocol

* + **Network Protocol** – a set of rules and standards, that allow communication between network devices
  + Network protocols include **mechanisms** for devices to identify and make **connections** with each other
  + Example for standard network protocols:
    - TCP, UDP, IP, ARP
    - HTTP, FTP, TFTP, SMPT, SSH

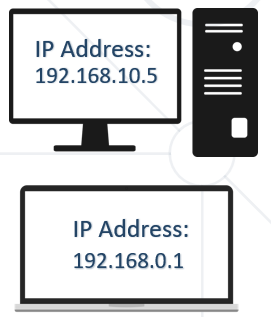
## 1.3. Packets (late 1960s)

* Everything that is created on a computer is translated into digital information using **bits**
* Bits need to have a way to be transmitted over the internet
* Every message, file or stream of information is broken down into small chunks called **packets**
* When packets are sent on the internet, they usually travel the network together
* But they might have to take a different route to get to the destination
* Each packet contains some **important information** inside of it called **the header**:
  + Where it came from
  + Where is it going
  + How long it is
    - This is how the packet is known to be complete
    - All the packets in the message are the same size
  + How many packets there are in the overall message

## 1.4. Internet Protocol (1970s) - IPv4, IPv6 and DNS

### Internet Protocol

* + One of the most important protocols used in Internet communication is the **Internet Protocol** (**IP**)
  + All the devices on the Internet have **addresses**
  + Theyarecalled **IP Addresses**
  + The IP address is **unique** to each computer or a device at the edge of the network

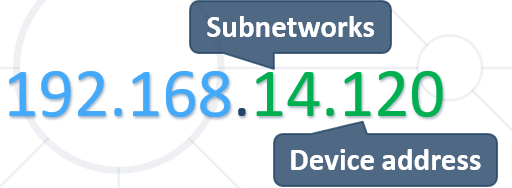


### IPv4

* + **IPv4** is a sequence of four, three-digit numbers separated by a period
    - Each number can be a number from 0 to 255
    - **IPv4** is not enough for all network devices connected to the internet
  + In 1995, a new version of the internet protocol was created, it's called **IPv6**

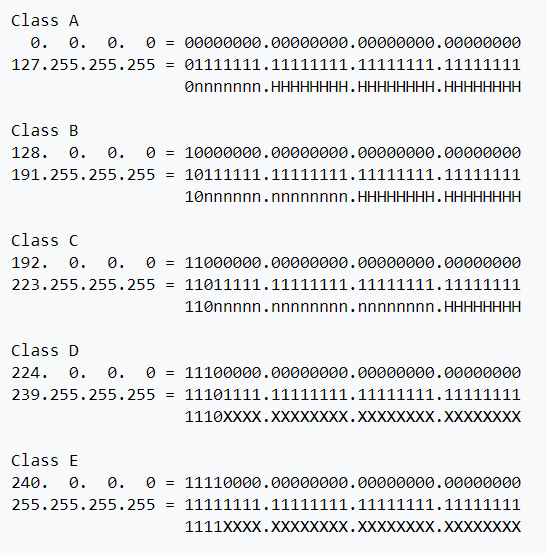
### IP Address

* + An **IP Address** has many parts, organized in a hierarchy



* + This version of IP Addressing is called **IPv4** 
    - Provides more than 4 billion **32 bits** unique addresses

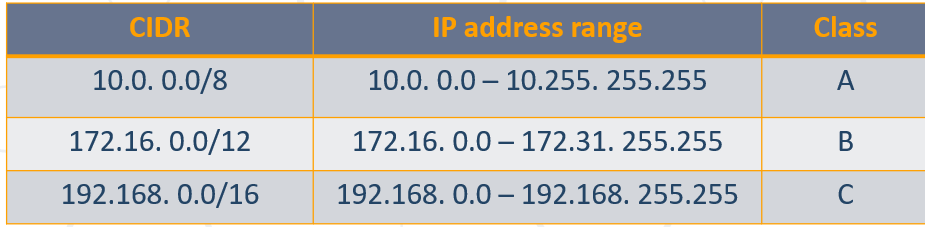
### IP address classes



### What Is CIDR (Classless Inter-Domain Routing)

* Classless Inter-Domain Routing, is an IP addressing scheme that **improves the allocation of IP addresses**.
* **It replaces the old system based on classes A, B, and C**.
* This scheme also helped greatly **extend the life of IPv4** as well as slow the growth of routing tables

### IPv4 Private Address Space and Filtering



### IPv6

* + **IPV6** uses **128 bits** - 340 undecillion unique addresses
    - That's more than the atoms on the surface of the Earth
  + These **128** bits are organized into eight 16 bit sections
  + Each 16 bit block is converted to hexadecimal and it's separated with a colon
  + This is a full IPV6 address:
    - **3FFE:F200:0234:AB00:0123:4567:8901:ABCD**
  + The **leading zeros** in **IPv6** can usually be left out



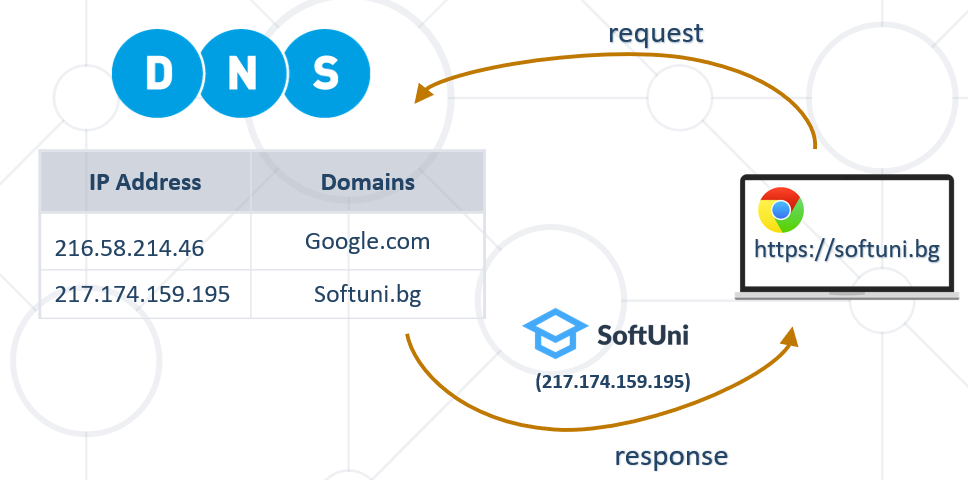
### What is a DNServer (**Domain Name Server**)?

* + The **domain name** is a human way to access IP addresses for devices and websites around the world



* + It is a sequence of phrases that **map** to a giant **Internet-wide database** of **IP addresses**
  + When a domain name is entered in the browser, a request is made to something called a **DNS** (**Domain Name Server**)
  + This server holds a **cache of tons of domain names, and their matching IP addresses**

### DNS (Domain Name Server) Example



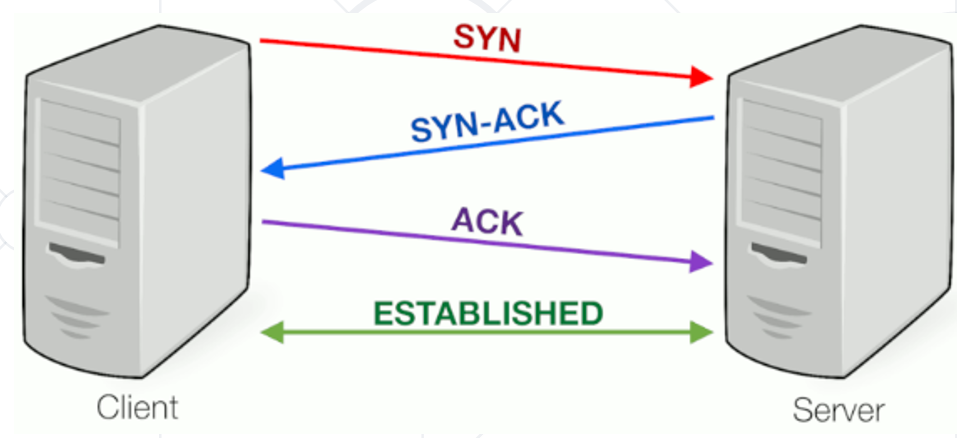
## 1.5. Reliability and TCP (Transmission Control Protocol)

### Reliability (надеждност/сигурност)

* When packets are transmitted from one location to another, they can take different paths
* When they get to the destination, they are unorganized and sometimes not complete
* So the message needs to be audited and reviewed in order to put it together in the right way
* The **Transmission Control Protocol** or **TCP** does exactly that

### TCP - Transmission Control Protocol

* **TCP** uses a process, where it looks at all the packets in a message and checks them
* Using the **header information** in **each packet** it knows:
  + How many there are
  + How large they should be
  + In which order the packets should be in
* Using this checklist, **it is able to rearrange the packets**
* If it finds that a packet doesn't match the expected characteristic, it is discarded (захвърлен)
* **TCP** has to **verify** that all the packets are:
  + In the right order
  + Free of any issues
* After that it **certifies** **the data** and the packets are **merged** together to recreate the **original** file that was on the sender's device



### TCP vs UDP

* + **TCP** places **reliability** in a higher priority than speed or latency
  + For instances where reliability isn't as important, but **speed** is, there is another protocol called **UDP** or **User Datagram Protocol**
  + UDP doesn't do excessive reliability checks, but it can send information at a faster rate
  + TCP is the foundation of how a majority of data is transmitted over networks



### UDP (User Datagram Protocol)

* + UDP does not establish a session and it does not guarantee data delivery
  + It is known as the "**fire-and-forget**" **protocol**
    - It sends data and it doesn't really care if the data is received at the other end



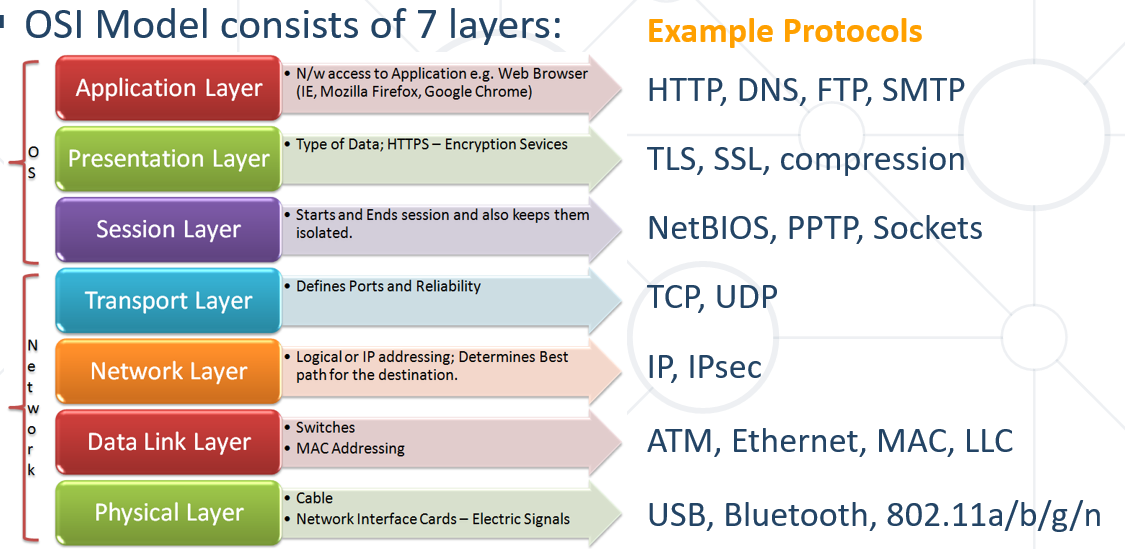
## 1.6. The OSI Model (Open System Interconnect)

### What is the OSI Model?

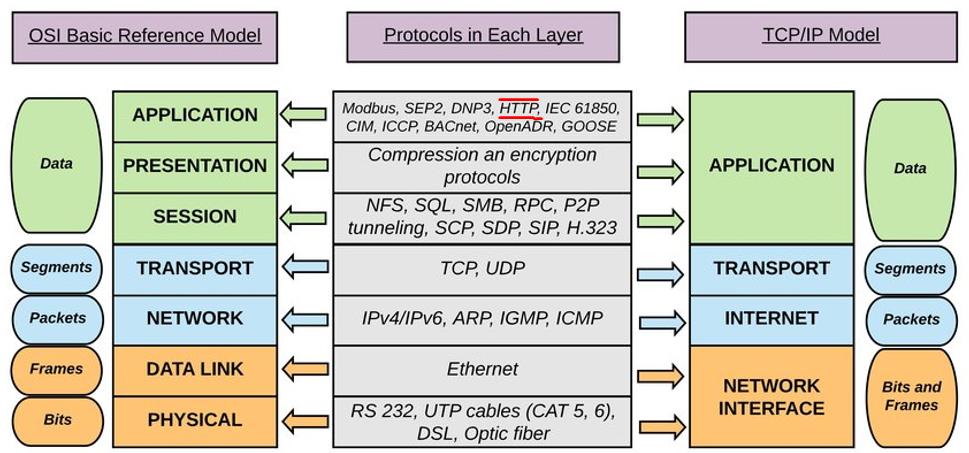
* **OSI** model stands for **O**pen **S**ystem **I**nterconnect
* It consists of 7 layers
  + Each layer serves the layer above it and in return, is served by the layer below it
* Understanding each layer of the model helps us with:
  + Troubleshooting
  + Communicating better with technical and non-technical individuals about any system

### OSI Layers

* OSI Model consists of 7 layers:

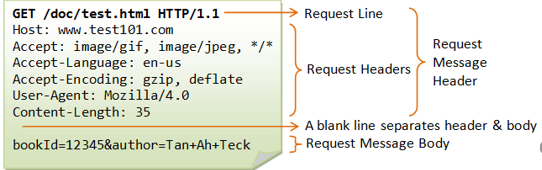


### TCP/IP model mapping to OSI



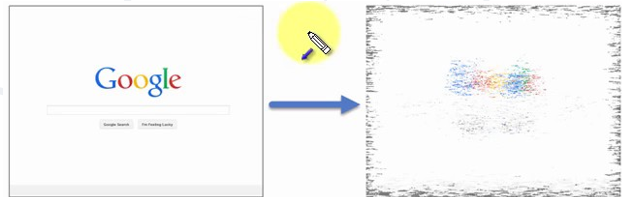
### Application Layer – level 7

* Enables different applications like the browser to use the network and **present it to the End User**
* Protocol examples:
  + **Domain Name System (DNSystem)**
  + **File Transfer Protocol (FTP)**
  + **HyperText Transfer Protocol (HTTP)**
  + **Simple Mail Transfer Protocol (SMTP)**



### Presentation Layer – level 6

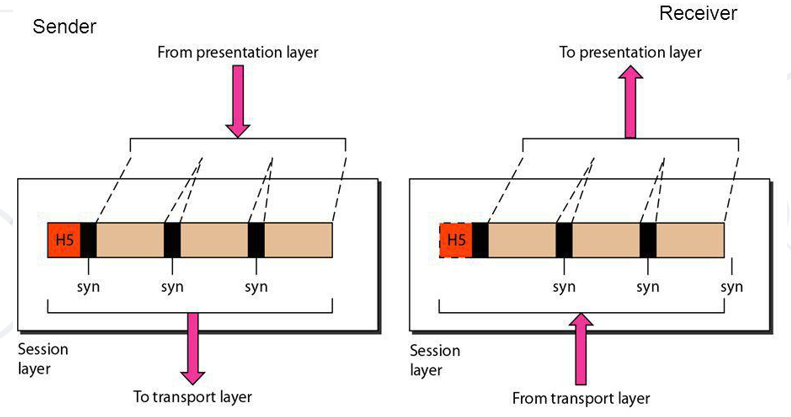
* This layer is a part of an operating system (OS)
* **Converts** incoming and outgoing **data** from **one presentation format to another**
* Example:
  + - From clear text to **encrypted** (or compressed) text
    - Back to clear text



### Session Layer – level 5

* This layer sets up coordinates and terminates conversations
* Its services include authentication and reconnection after an interruption

e.g.: Sockets

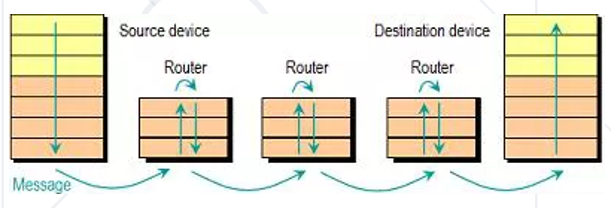


### Transport Layer – level 4

* Responsible for end-to-end communication over a network
* Provides logical communication between application processes
* Responsible for the management of error correction, providing quality and reliability to the end user
* Protocol examples:
  + **Transmission Control Protocol (TCP)**
  + **User Datagram Protocol (UDP)**

### Network Layer – level 3

* Provides the functional and procedural means of **transferring packets from one node to another**
* Responds to service requests from the transport layer and issues service requests to the data link layer
* Protocol examples:
  + **Internet Protocol (IP)**
  + **IPSec (IP + Auth)**



### Data Link Layer – level 2

* Provides node-to-node data transfer
* It **detects** and possibly **corrects** errors that may occur in the **physical layer**
* Divides into two sublayers:
  + **Medium access control (MAC)** layer - controlling how devices in a network gain access to a medium and permission to transmit data
  + **Logical link control (LLC)** layer – identifying and encapsulating network layer protocols, controls error checking and frame synchronization
* Protocol examples:
  + **Asynchronous Transfer Mode (ATM)**
  + **Ethernet**
  + **MAC**



### Physical Layer – level 1

* The things you can actually physically touch
* Converts the **binary** from the upper layers into **signals**, **transmits** them over local media (electrical, light, or radio signals)
* Examples:
  + **Ethernet**
  + **USB**
  + **Bluetooth**
  + **802.11a/b/g/n**



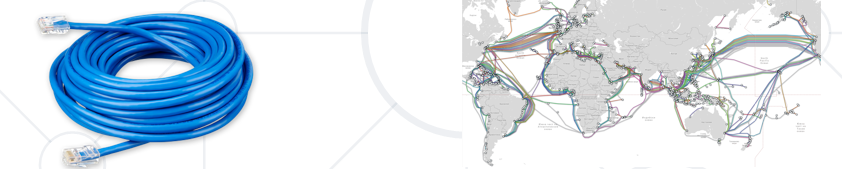
## 1.7. Network Hardware

### Network Hardware

* Basic Hardware Components
  + Cables
  + Routers
  + Repeaters, Hubs and Switches
  + Bridges
  + Gateways
  + Network Interface Cards

### Cables and Routers

* Network Cables – the **transmission** **media** to transfer data from one device to another



* Router – **connecting** **device** that transfers data packets between different computer networks (operates on level 3 of OSI)



### Repeaters, Hubs and Switches

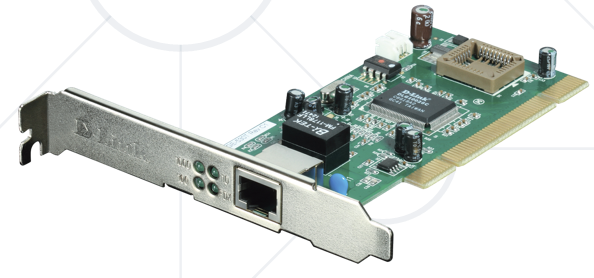
* **Repeaters**, **hubs** and **switches** **connect** network **devices** together so that they can function as a single segment
  + Repeater – **receives** a **signal** and regenerates it before re-transmitting, so that it can travel longer distances
  + Hub – multiport **repeater** (operates on level 1 of the OSI model)
  + Switch – **receives** **data** from a port, uses packet switching to resolve the destination device and forwards the data to the particular destination (operates on level 2 of the OSI model)

### Bridges and Gateways

* **Bridge**
  + Connects two separate but **similar** Ethernet network segments
  + Forwards packets from the source network to the destined network (operates on level 2 of OSI)
* **Gateway**
  + **Connects networks** that work upon **different** protocols
  + The entry and the exit point of a network (**controls the access to other networks**) Level 4, 5, 6 or 7 of the OSI model (same as Firewalls)

### Network Interface Cards – NIC

* **NIC** – a computer component that connects it to the network
* There are two types of network cards:
  + Internal
  + External



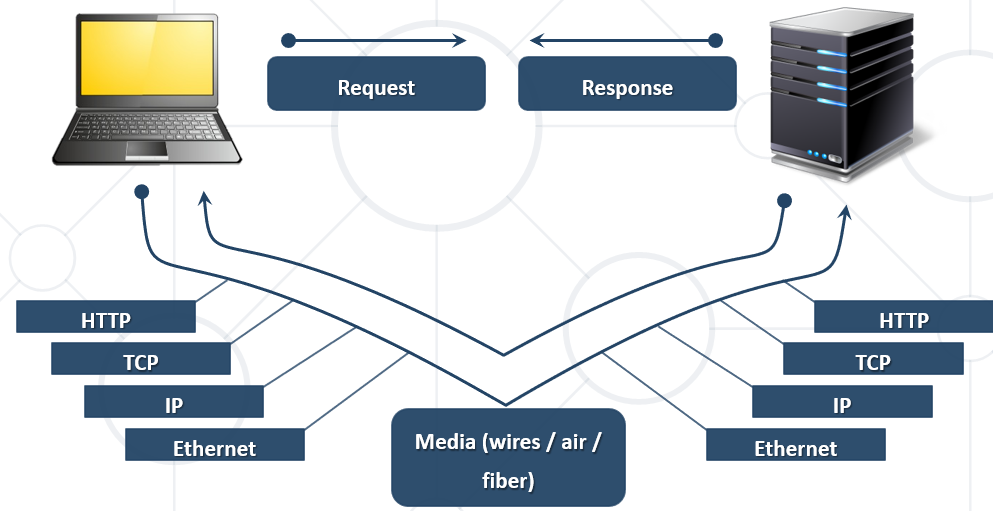
## 1.8. The Future of the Internet

* The **"Internet of Things"** will expand
  + Healthcare, agriculture, wearables, manufacturing
  + Smart homes, cars and cities (pollution, parking, energy)
  + In 2030 there will be **50 billion devices** connected to the Internet of Things

# 2. HTTP Protocol

## 2.1. HTTP Basics

### Hyper Text Transfer Protocol

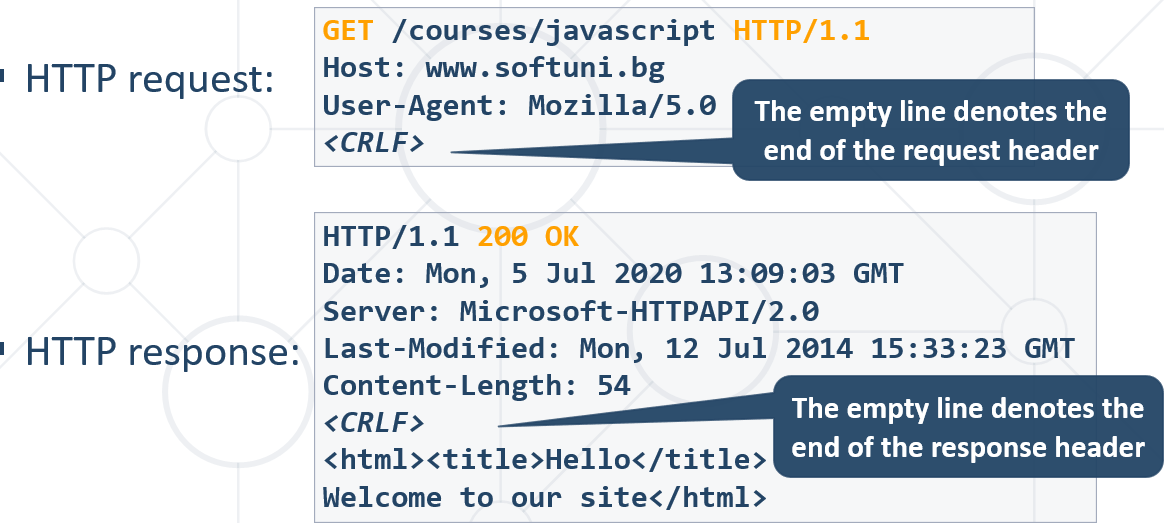


### HTTP Request Methods

* **HTTP** defines **methods** to indicate the desired action to be performed on the identified resource

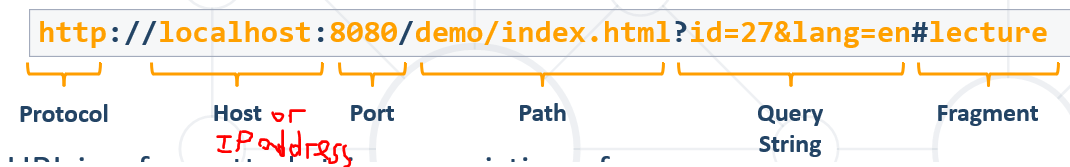
|  |  |
| --- | --- |
| Method | Description |
| GET | Retrieve / load a resource |
| POST | Create / store a resource |
| PUT | Update a resource by replacing it with a new one |
| PATCH | Update partial parts of the item |
| DELETE | Remove a resource |

### HTTP Conversation: Example



## 2.2. URL - Uniform Resource Locator

### Uniform Resource Locator (URL)



* URL is a formatted string, consisting of:
* Protocol for communicating (**http**, **ftp**, **https,** ...) – HTTP in most cases
* **Host(Domain name Server = DNServer)** or **IP** address (**www.softuni.bg**, **gmail.com**, **127.0.0.1**, **web**)
* Port (the default port is **80**) – a number in range [0…65535]
* Path (**/forum**, **/path/index.html**)
* Query string (**?id=27&lang=en**) – не се използва за парола 😊
* Fragment (**#lectures**) – used on the client to navigate to some section – за позициониране в даден документ. Не пътува по мрежата – за user friendliness!!!

### URI and URL

URL е подмножество на URI.

Механизъм за вземане на ресурс.

### URL Encoding

* URLs are encoded according RFC 1738:
  + Safe URL characters: **[0-9a-zA-Z]**, **$**, **-**, **\_**, **.** , **+**, **\***, **'**, **(**, **)**, **,**, **!**
* All other characters are escaped by:

**%[character hex code]**

|  |  |
| --- | --- |
| Char | URL Encoding |
| space | %20 |
| щ | %D1%89 |
| " | %22 |
| # | %23 |
| $ | %24 |
| % | %25 |
| & | %26 |

* Space is encoded as "**+**" or "**%20**"

Когато е в дясно от въпросчето, може да се encode-не с +

* URL-encoded string:

%D0%9D%D0%B0%D0%BA%D0%BE%D0%B2-%E7%88%B1-SoftUni

<https://www.punycoder.com/>

Punycode is a special encoding used to convert Unicode characters to ASCII, which is a smaller, restricted character set. Punycode is used to encode internationalized domain names (IDN).

### Valid and Invalid URLs – Examples

* Some valid URLs:

http://www.google.bg/search?sourceid=navclient&ie=UTF-8&rlz=1T4GGLL\_enBG369BG369&q=http+get+vs+post

http://bg.wikipedia.org/wiki/%D0%A1%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80%D0%BD%D0%B0\_%D0%B0%D0%BA%D0%B0%D0%B4%D0%B5%D0%BC%D0%B8%D1%8F

* Some invalid URLs:

http://www.google.bg/search?&q=C# .NET 4.0

<http://www.google.bg/search?&q=бира> написана е на български бира, а трябва да е с %-та

## 2.3. HTTP Request

### HTTP Request Message

* Request message sent by a client consists of:
  + HTTP **request line**
    - Request method (**GET** / **POST** / **PUT** / **DELETE** / …)
    - Resource URI (**URL**)
    - Protocol version
  + HTTP **request headers**
    - Additional parameters
  + HTTP **request body** – optional data, e.g. posted form fields

**<method> <resource> HTTP/<version>**

**<headers>**

***(empty line)***

**<body>**

### HTTP GET Request – Example

**GET /index.html HTTP/1.1** HTTP request line

**Host: localhost** HTTP request headers

***<CRLF>*** The request body is empty

### HTTP POST Request – Example

**POST /login.html HTTP/1.1** HTTP request line

**Host: localhost** HTTP request headers

**Content-Length: 59**

**Content-Type: application/x-www-form-urlencoded** който пътува в бодито

***<CRLF>***

**username=testUser&password=topSecret** The request body holds the submitted form data

***<CRLF>***

## 2.4. HTTP Response

### HTTP Response Message

* The **response message** sent by the HTTP server consists of:
  + HTTP response **status line**
    - Protocol version
    - Status code
    - Status phrase
  + Response **headers**
    - Provide meta data about the returned resource
  + Response **body**
    - The content of the HTTP response (data)

**HTTP/<version> <status code> <status text>**

**<headers>**

***(empty line)***

**<response body – the requested resource>**

### HTTP Response – Example

**HTTP/1.1 200 OK** HTTP response status line

**Date: Fri, 17 Jul 2020 16:09:18 GMT+2**

**Server: Apache/2.2.14 (Linux)** HTTP response headers

**Accept-Ranges: bytes**

**Content-Length: 84**

**Content-Type: text/html**

***<CRLF>***

**<html>**

**<head><title>Test</title></head>** HTTP response body

**<body>Test HTML page.</body>**

**</html>**

### HTTP Response Codes

* HTTP response code classes
  + **1xx**: informational (e.g., "**100** Continue")
  + **2xx**: successful (e.g., "**200** OK", "**201** Created")
  + **3xx**: redirection (e.g., "**304** Not Modified", "**301** Moved Permanently", "**302** Found")
  + **4xx**: client error (e.g., "**400** Bad Request", "**404** Not Found", "**401** Unauthorized", "**409** Conflict")
  + **5xx**: server error (e.g., "**500** Internal Server Error", "**503** Service Unavailable")

### HTTP Error Response – Example

**HTTP/1.1 404 Not Found** HTTP response status line

**Date: Fri, 17 Nov 2020 16:09:18 GMT+2**

**Server: Apache/2.2.14 (Linux)** HTTP response headers

**Connection: close**

**Content-Type: text/html**

***<CRLF>***

**<html><head><title>404 Not Found</title></head>**

**<body>** The HTTP response body

**<h1>Not Found</h1>**

**<p>The requested URL /img/logo.gif was not found on this server.</p> <hr><address>Apache/2.2.14 Server at Port 80</address>**

**</body></html>**

### Browser Redirection

### Content-Type and Disposition

The **Content-Type** response header the server specifies how the output should be processed

**Content-Type: text/html; charset=utf-8**

**Content-Type: application/pdf**

**Content-Disposition: attachment; filename="Report-April-2020.pdf"** This will download a PDF file named   
Report-April-2020.pdf

**Content-Type: application/json**

## 2.5. MIME (Multi-Purpose Internet Mail Extensions) and Media Types

Multi-Purpose Internet Mail Extensions

### What is MIME?

* [**MIME**](http://en.wikipedia.org/wiki/MIME)== **M**ulti-Purpose **I**nternet **M**ail **E**xtensions
  + Internet standard for encoding resources
  + Originally developed for email attachments
  + Used in many Internet protocols like HTTP and SMTP
* MIME defines several concepts
  + **Content-Type**, e.g. **text/html**, **image/gif**, **application/pdf**
    - Content **charset**, e.g. **utf-8**, **ascii**, **windows-1251**
  + **Content-Disposition**, e.g. **attachment;** **filename=logo.jpg**
  + Multipart messages (multiple resources in a single document)

### Common MIME Media Types

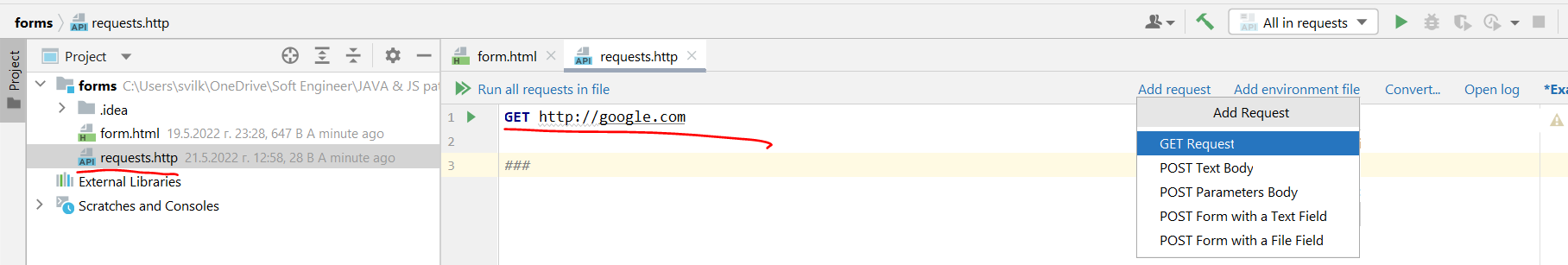
**Content-Type:**

|  |  |
| --- | --- |
| **MIME Type / Subtype** | **Description** |
| application/json | JSON data |
| image/png | PNG image |
| image/gif | GIF image |
| text/html | HTML |
| text/plain | Text |
| text/xml | XML |
| video/mp4 | MP4 video |
| application/pdf | PDF document |
| multipart/form-data; boundary |  |

## 2.6. HTTP Tools

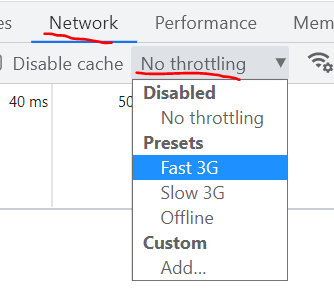
**HTTP Tools for Developers:**

* **През IntelliJ**



* **Browser – Chrome DevTools, Firefox DevTools, Microsoft Edge DevTools**

Ако искаме да видим как ни върви сайта при бавна връзка на клиента



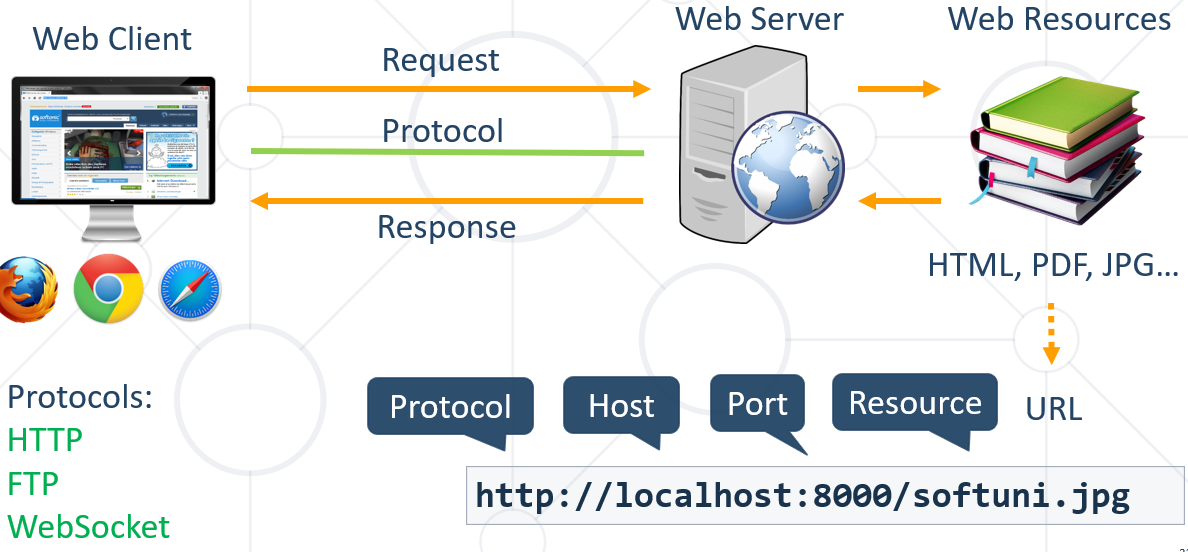
* **Telnet**
* През Ubunto (Linux) и SSL връзка
* [Insomnia Rest](https://insomnia.rest/)
* [Postman](https://www.postman.com/downloads/)
* [RESTClient](https://addons.mozilla.org/bg/firefox/addon/restclient/)

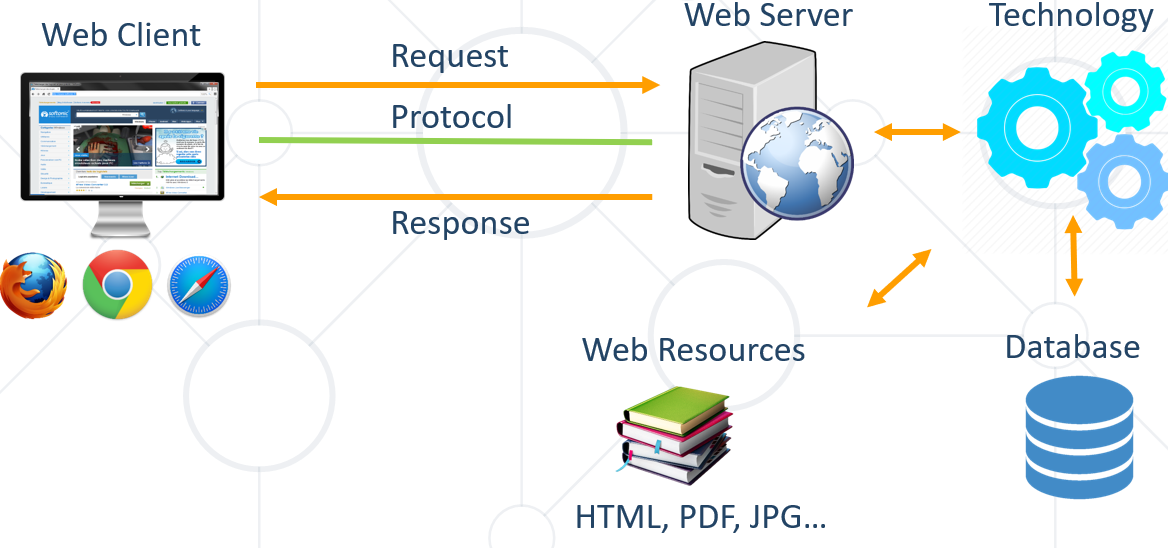
## 2.7. Web Server

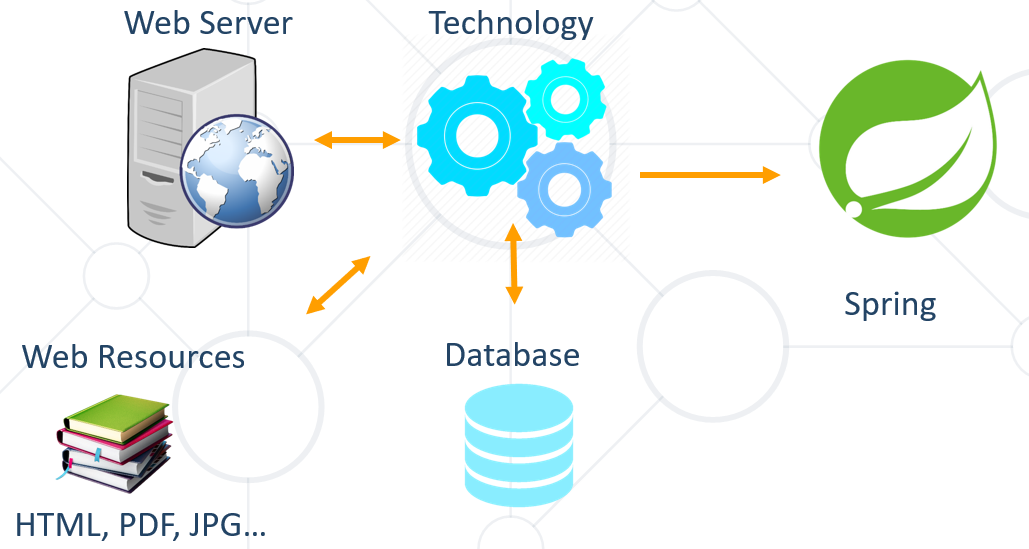
### What is a Web Server?

* Computer system that processes requests via **HTTP**, the basic network protocol

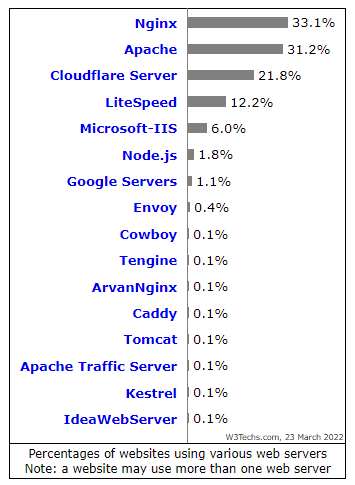
### Web Server Work Model







### Most Popular Web Servers (W3Techs)

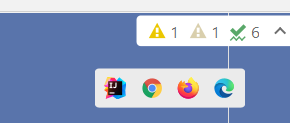


## 2.8. HTML Forms

### Form.html

<!DOCTYPE **html**>  
<**html**>  
<**body**>  
  
<**h2**>HTML Forms</**h2**>  
  
<**form action="http://postman-echo.com/post" method="post"**>  
 <**label for="fname"**>First name:</**label**><**br**>  
 <**input type="text" id="fname" name="fname" value="John"**><**br**>  
 <**label for="lname"**>Last name:</**label**><**br**>  
 <**input type="text" id="lname" name="lname" value="Doe"**><**br**><**br**>  
 <**input type="submit" value="Submit"**>  
</**form**>  
  
<**button onclick="***testMe*()**"**>CLICK ME</**button**>  
  
<**script**>  
 **function** *testMe*() {  
 **for** (**let** i = 0; i < 10; i++) {  
 ***console***.log(**"Test "** + i);  
 }  
  
 ***console***.log(**"Test "** + i);  
 }  
</**script**>  
  
</**body**>  
</**html**>

### На .html файл излизат тези иконки на браузъри, и си зареждаме оттам web страницата.



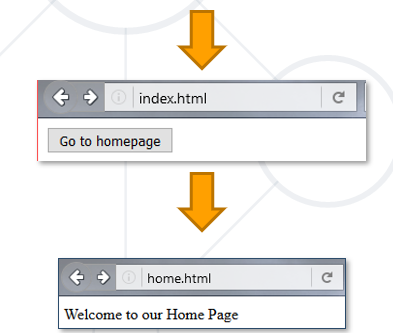
### HTML Forms – Action Attribute

* Defines **where to submit** the form data:

<form action="home.html">

  <input type="submit" value="Go to homepage"/>

</form>



### HTML Forms – Method Attribute

* Specifies the HTTP method to use when sending form-data

#### Get

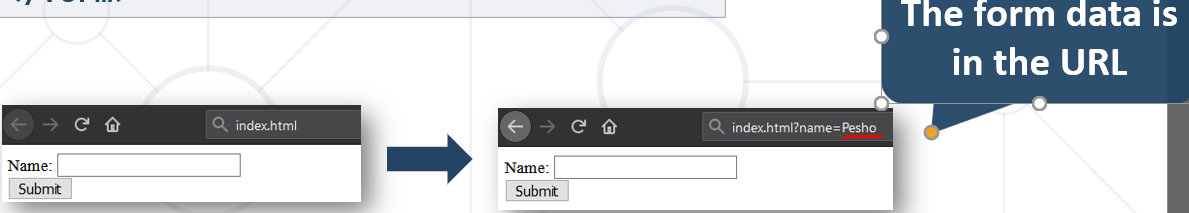
<form action="/" method="get">

    Name: <input type="text" name="name">

        <br>

  <input type="submit" value="Submit">

</form>



#### Post

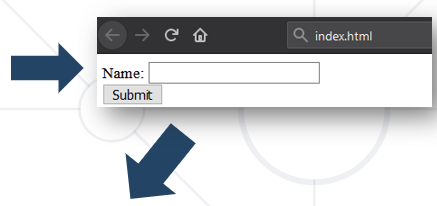
<form action="/" method="post">

    Name: <input type="text" name="name">

    <br>

    <input type="submit" value="Submit">

</form>



POST http://localhost/index.html HTTP/1.1

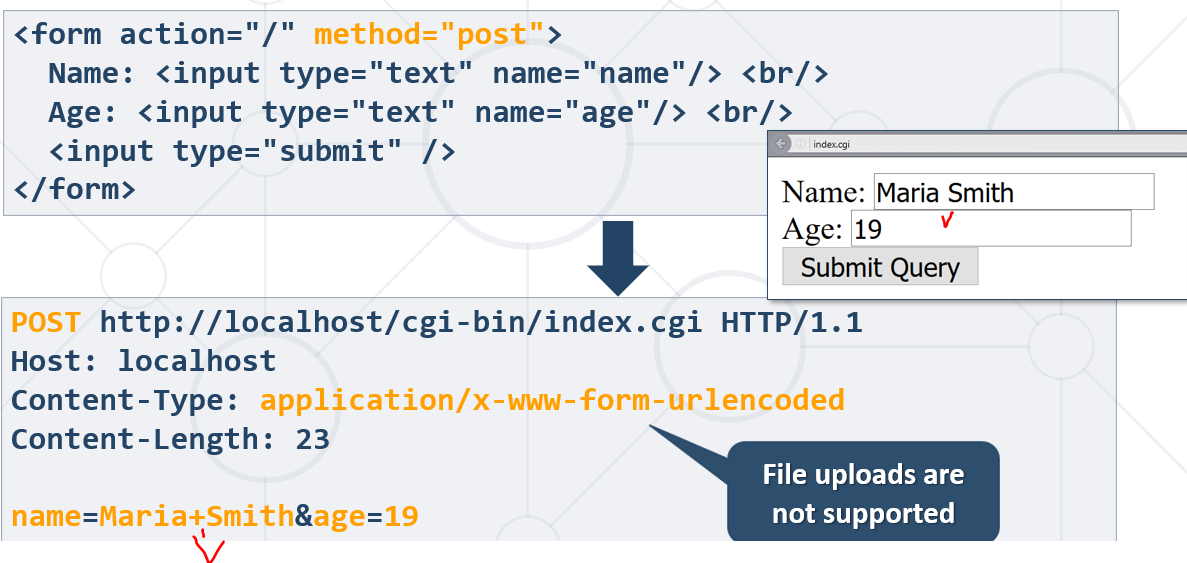
Host: localhost

Content-Type: **application/x-www-form-urlencoded**

Content-Length: 10

name=Pesho **HTTP request body holds the form data**

### URL Encoded Form Data – Example



## 2.8. HTTP/2

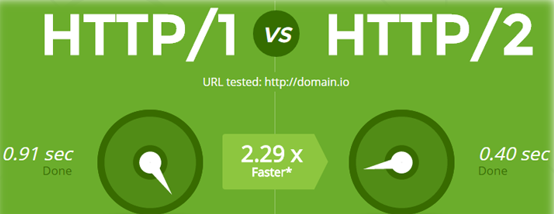
### Old HTTP/1.1

### What's HTTP/2

* Speedy е бащата/експерименталната версия на HTTP2
* **HTTP/2** (originally named **HTTP/2.0**) major revision of the **HTTP** network protocol used by the **World Wide Web**.
  + Supported by most of the popular web browsers (Chrome, Mozilla, Opera...)
  + **Fast & Optimized.** Meets modern web usage requirements.
  + Completely **Backwards-Compatible**
* As of Jan 2021, **50%** of all the websites support **HTTP/2** (W3Techs statistics).

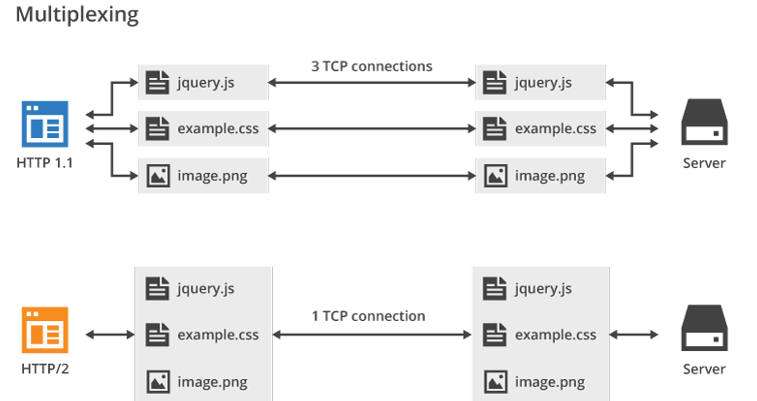
### What's New?

* **HTTP/2** is meant to erase the need of maintaining complex server infrastructures in order to perform well.
* **HTTP/2** communicates in binary data frames.
* **HTTP/2** introduces several **new important** elements
  + HTTP/2 Multiplexing
  + HTTP/2 Header Compression
  + HTTP/2 Server Push



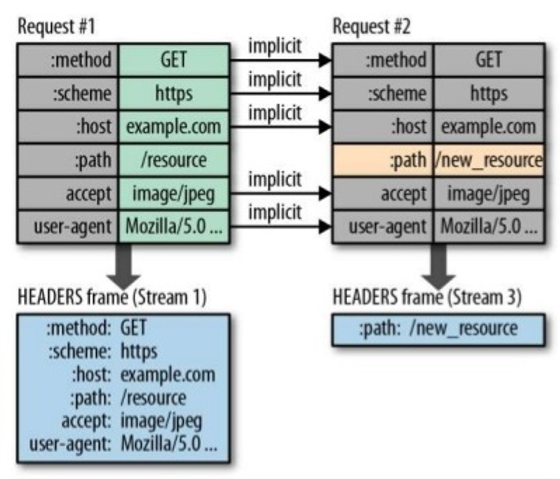
### HTTP/2 Multiplexing

* The art of handling multiple streams over a **single** TCP connection.



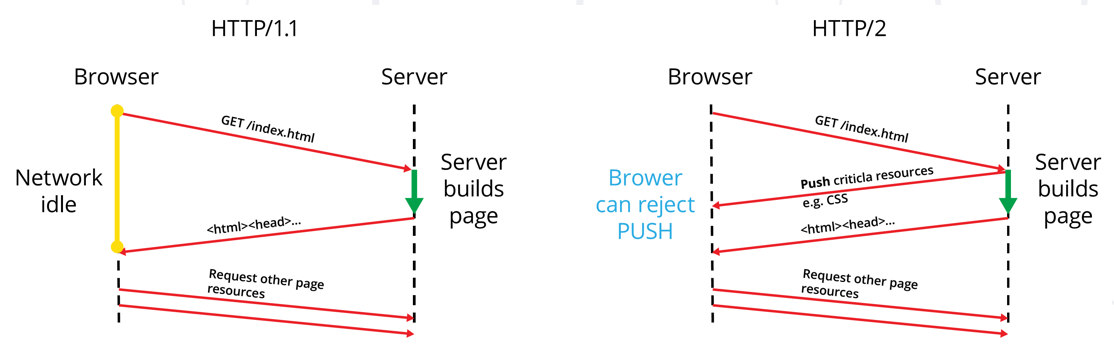
### HTTP/2 Header Compression

* **HTTP/2** maintains a **HTTP** **Header Table** across requests.
* Optimizes communication drastically.
* The process is essentially a **de-duplication**, rather than compression – не се изпращат части, които са redundant(излишни).



### HTTP/2 Server Push – TCP handshake

* **HTTP/2 Server Push** is the process of sending resources to clients, without them having to ask for it.



## 2.9. What's HTTP/3

* **HTTP/3** is a new standard in development that will affect how web browsers and servers communicate.
* Significant upgrades for user experience
* **Performance**, **Reliability**,and **Security**
* **HTTP/3** runs on **QUIC**, a new transport protocol designed for **mobile-heavy** Internet usage.

## 2.10. Два основни вида криптирани алгоритми + катинарче

**Transport Layer Security** (**TLS**) the successor of the now-deprecated Secure Sockets Layer (SSL)

is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use in securing HTTPS remains the most publicly visible.

Симетрични алгоритми за криптиране - бързи – разчитат на един и същи ключ от двете страни за осъществяване на криптирания канал – AES (Advanced Encryption Standard), 3DES, Blowfish is a symmetric-key block cipher, и много други.

Асиметрични алгоритми за криптиране - бавни

Публичен ключ на сървъра – дава се на всеки

Private ключ на сървъра – само на конкретния потребител

Ако нещо се криптира Public key, то може да се декриптира само с private ключа.

И ако нещо се криптира с Private key (моят цифров подпис), то може да се декриптира само с Public Key и всеки който иска да знае, че това съм аз.

При осъществена криптирана връзка, излиза ключето в браузъра.

SHA256 и хеширащи алгоритми за пазене на пароли….

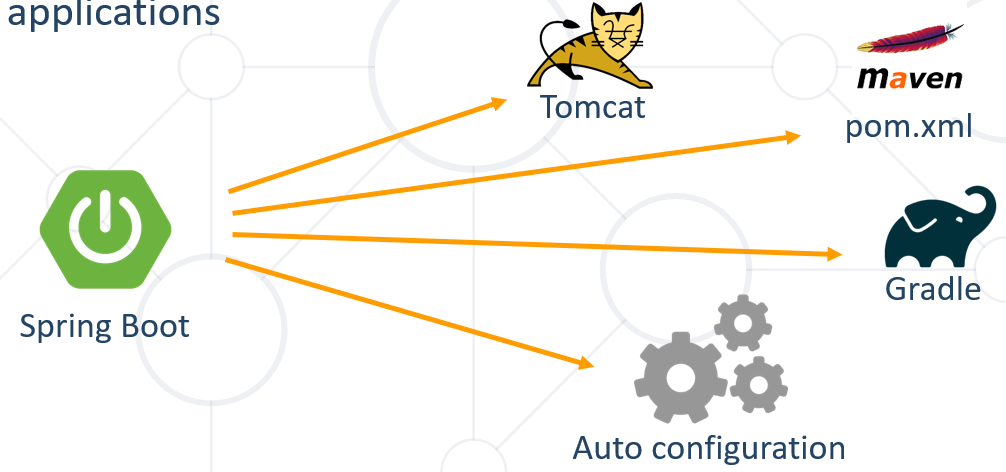
За разлика от хепирането, което е необратим процес, то крипитрането е обратим процес и може да се декриптира.

# 3. Spring Boot Introduction

## 3.1. What is Spring Boot?

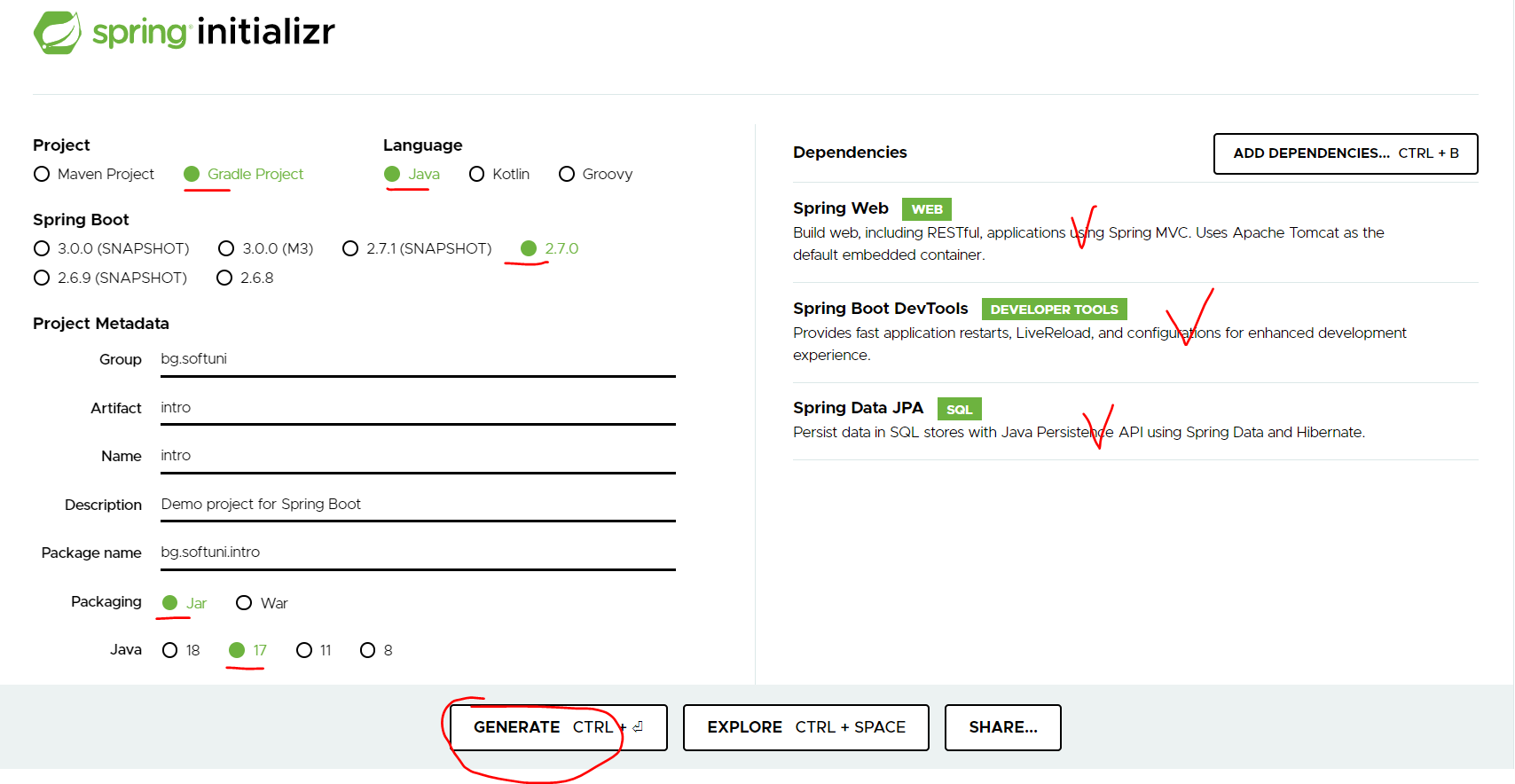
**Spring Boot**

* **Opinionated view** of building production-ready Spring applications



### Creating Spring Boot Project

* Just go to [**https://start.spring.io/**](https://start.spring.io/) or open IntelliJ
* Create it via Spring initializer



Jar = uberjar = stand alone = може да се стартира от java-.. = в него има сървър

War = webarchive

### Spring Dev Tools

* Additional set of **tools** that can make the application development **faster** and more **enjoyable**
* In **Maven**:

Pom.xml

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>**spring-boot-devtools**</artifactId>  
 <scope>runtime</scope>  
</dependency>

* In **Gradle**:

dependencies {

**compileOnly**("org.springframework.boot:**spring-boot-devtools**")

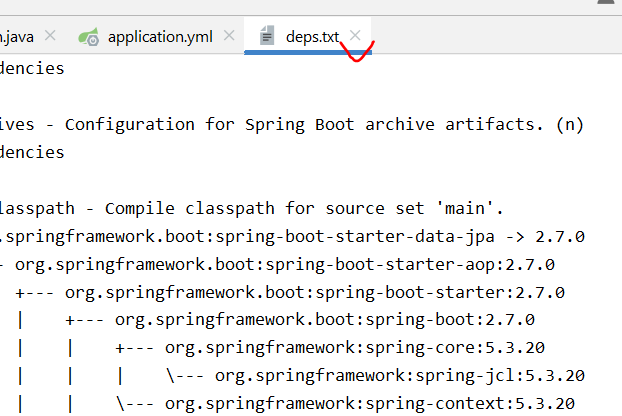
}

### Example of Gradle

<https://docs.gradle.org/current/userguide/userguide.html>

В терминала пишем:

./gradlew dependencies > deps.txt - да видим всичките dependences



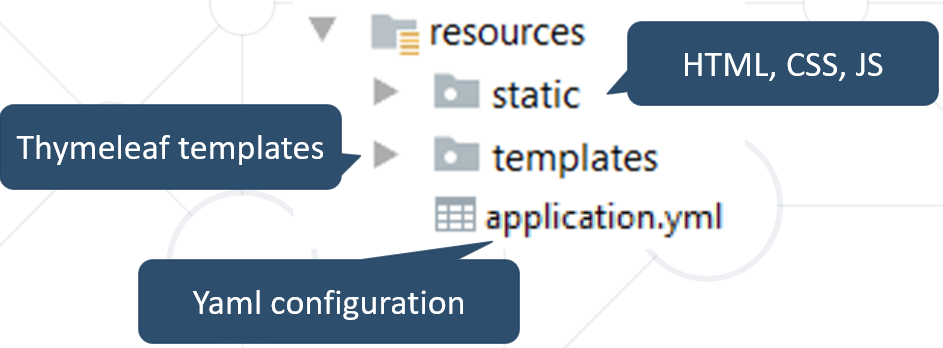
#### build.gradle file

plugins {  
 id **'org.springframework.boot'** version **'2.7.0'** id **'io.spring.dependency-management'** version **'1.0.11.RELEASE'** id **'java'**}  
  
group = **'bg.softuni'**version = **'0.0.1-SNAPSHOT'**sourceCompatibility = **'17'**repositories {  
 mavenCentral()  
}  
  
dependencies {  
 ***implementation******'org.springframework.boot:spring-boot-starter-data-jpa'*****implementation** **'org.springframework.boot:spring-boot-starter-web'  
 developmentOnly** **'org.springframework.boot:spring-boot-devtools'  
 testImplementation** **'org.springframework.boot:spring-boot-starter-test'**

**runtimeOnly** **'mysql:mysql-connector-java'**

}  
  
tasks.named(**'test'**) {  
 useJUnitPlatform()  
}

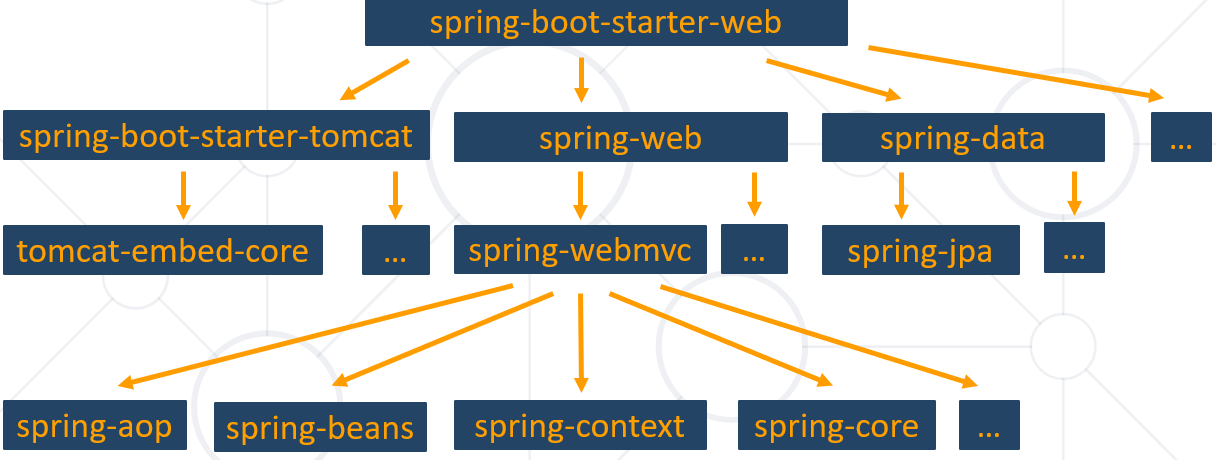
### Spring Resources



### Spring Boot Main Components

* Some main components:
  + **Spring Boot Starters** - combine a group of common or related dependencies into single dependency
  + **Spring Boot Auto-Configuration** - reduce the Spring Configuration
  + **Spring Boot Actuator** – provides EndPoints and Metrics
  + **Spring Data** – unify and ease the access to different kinds of database systems

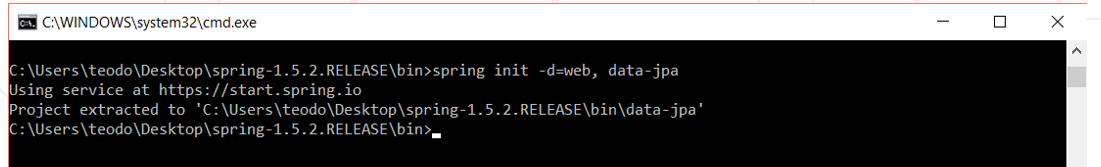
### Spring Boot Starters



### Spring Boot CLI (Command Line Interface)

* **Command Line Interface** - Spring Boot software to run and test Spring Boot applications

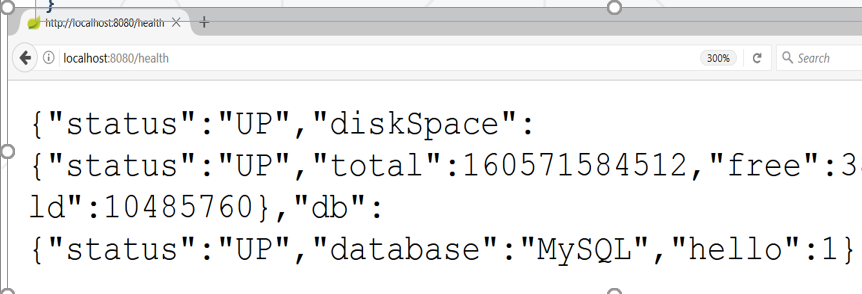
Още един начин да се стартира проект.



### Spring Boot Actuator

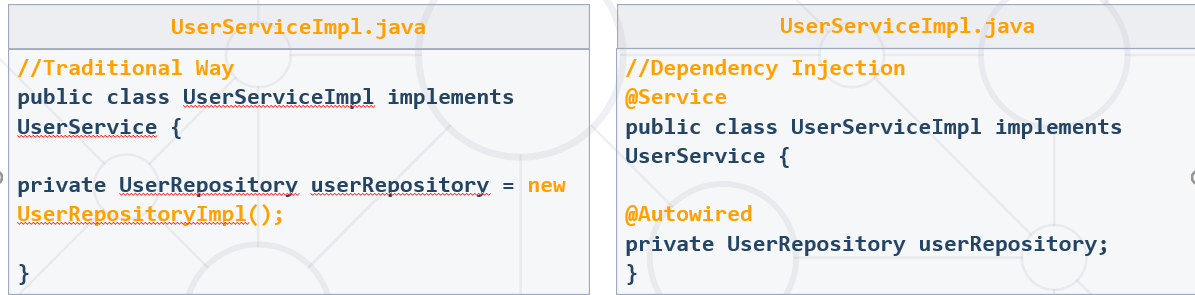
* Expose different types of information about the **running** **application**





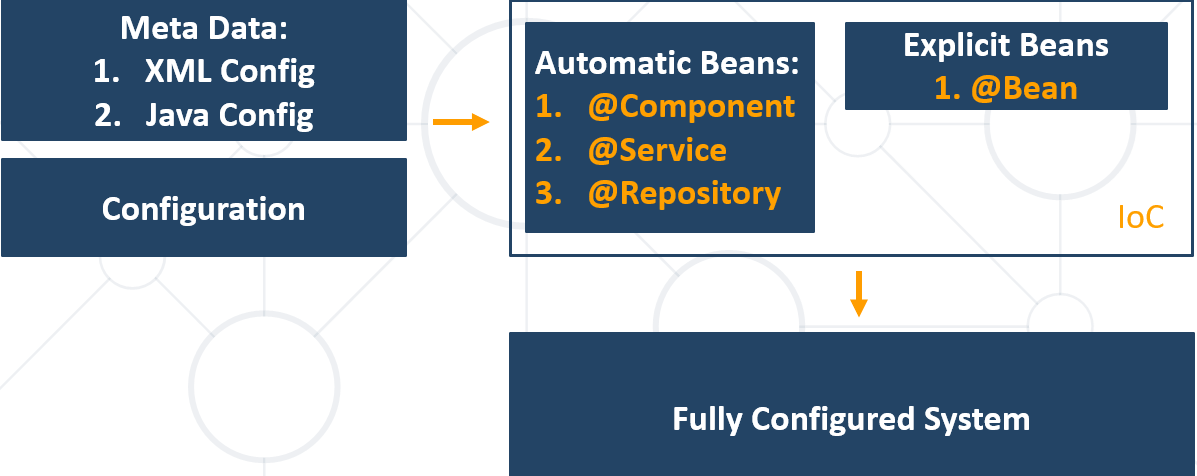
### Inversion of Control

* Spring provides **Inversion of Control** and **Dependency Injection**



В новите версии на Spring, не е необходимо да слагаме @Autowired

### Spring IoC (Inversion of Control principle)



### Beans

* Object that is **instantiated**, **assembled**, and otherwise managed by a **Spring** **IoC** container

Т.е. не е необходимо да използваме new оператора.

**public class** Dog **implements** Animal {  
  
 **private** String **name**;  
  
 **public** Dog() {}  
  
 *//GETTERS AND SETTERS*}

### Bean Declaration

@SpringBootApplication  
**public class** MainApplication {  
  
 …  
  
 @Bean  
 **public** Animal getDog(){  
 **return new** Dog();  
 }  
}

### Get Bean from Application Context

@SpringBootApplication  
**public class** MainApplication {  
 **public static void** main(String[] args) {  
 ApplicationContext context = SpringApplication.run(MainApplication.**class**, args);  
 Animal dog = context.getBean(Dog.**class**); //глобалния контекст  
 System.out.println(**"DOG: "** + dog.getClass().getSimpleName());  
 }  
}



### Beans Scopes in Spring Framework

Ако инжектираме едно и също куче на 2 места в проекта, то какво ще стане? – Отговорът е, че зависи от scope на bean-a.

* There are part of **Beans scopes**:
  + **Singleton** – една и съща инстанция на кученцето в целия проект
  + **Prototype** – еквивалента на new оператора – всеки път нова инстанция – използват се рядко
  + **Request** - в рамките на http request
  + **Session** – да пазим неща специфични за даден user

### Singletone Scope

* Container creates a **single instance** of that bean, and all requests for that bean name will return the **same object**, which is cached
* This is **default** scope

@Bean  
@Scope(**"singleton"**)<-Can be omitted (може да се пропусне)  
**public** Student student(){  
 **return new** Student();  
}

### Prototype Scope

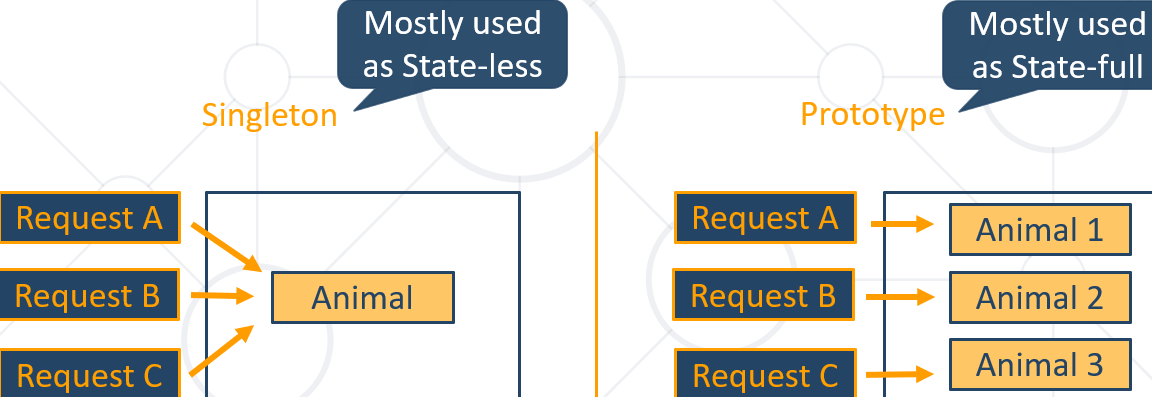
* Will return a different instance every time it is requested from the container

@Bean  
@Scope(**"prototype"**)  
**public** Student student() {  
 **return new** Student();  
}

### Bean Scope

* The default one is **Singleton**. It is easy to change to **Prototype**

State-less / state-full



### Demo with Beans

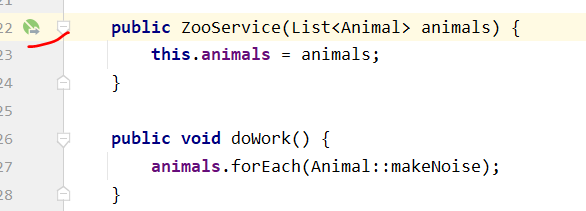
IntelliJ средата ни дава препратка към двата bean-a

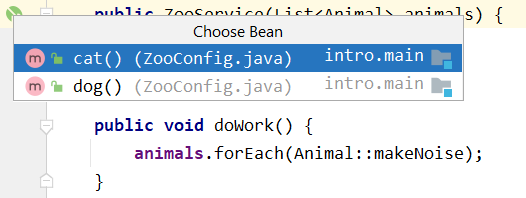
#### Вариант 1 при повече bean-ове – с обхождане на всеки от bean-овете

**public class** Cat **implements** Animal{  
 @Override  
 **public void** makeNoise() {  
 System.***out***.println(**"Meoue mew"**);  
 }  
}

@Configuration  
**public class** ZooConfig {  
 @Bean  
 **public** Animal cat() {  
 **return new** Cat();  
 }  
  
 @Bean  
 **public** Animal dog() {  
 **return new** Dog();  
 }

}





@Service  
**public class** ZooService {

**private final** List<Animal> **animals**;  
**public** ZooService(List<Animal> animals) {  
 **this**.**animals** = animals;  
}  
  
**public void** doWork() {  
 **animals**.forEach(Animal::makeNoise);  
}

}

#### Вариант 2 при повече bean-ове – с използване на анотацията @Primary

**public class** Cat **implements** Animal{  
 @Override  
 **public void** makeNoise() {  
 System.***out***.println(**"Meoue mew"**);  
 }  
}

@Configuration  
**public class** ZooConfig {  
 @Bean  
 **public** Animal cat() {  
 **return new** Cat();  
 }

**@Primary**  
 @Bean  
 **public** Animal dog() {  
 **return new** Dog();  
 }

}

@Service  
**public class** ZooService {  
  
 **private final** Animal **animal**;  
  
 @Autowired  
 **public** ZooService(Animal animal) {  
 **this**.**animal** = animal;  
 }  
  
 **public void** doWork() {  
 **animal**.makeNoise();  
 }

}

#### Вариант 3 при повече bean-ове – с използване на qualifier I BeanNameAware Interface

**public class** Dog **implements** Animal{  
 **private final boolean superDog**;  
  
 **public** Dog(){  
 **this**(**false**);  
 }  
  
 **public** Dog(**boolean** superDog){  
 **this**.**superDog** = superDog;  
 }  
  
 @Override  
 **public void** makeNoise() {  
 **if** (**superDog**) {  
 System.***out***.println(**"Super Bark super bark"**);  
 } **else** {  
 System.***out***.println(**"Bark bark"**);  
 }  
 }  
}

@Configuration  
**public class** ZooConfig {  
@Bean(**"cat"**)  
 **public** Animal cat() {  
 **return new** Cat();  
 }  
  
 @Bean(**"normalDog"**)  
 **public** Animal dog() {  
 **return new** Dog();  
 }  
  
 @Bean(**"mySuperDog"**)  
 **public** Animal superDog() {  
 *//****todo: add superpower to this dog* return new** Dog(**true**);  
 }  
}

@Service  
**public class** ZooService {  
 **private final** Animal **animal**;  
  
 @Autowired  
 **public** ZooService(@**Qualifier**(**"mySuperDog"**) Animal animal) {  
 **this**.**animal** = animal;  
 }  
  
 **public void** doWork() {  
 **animal**.makeNoise();  
 }

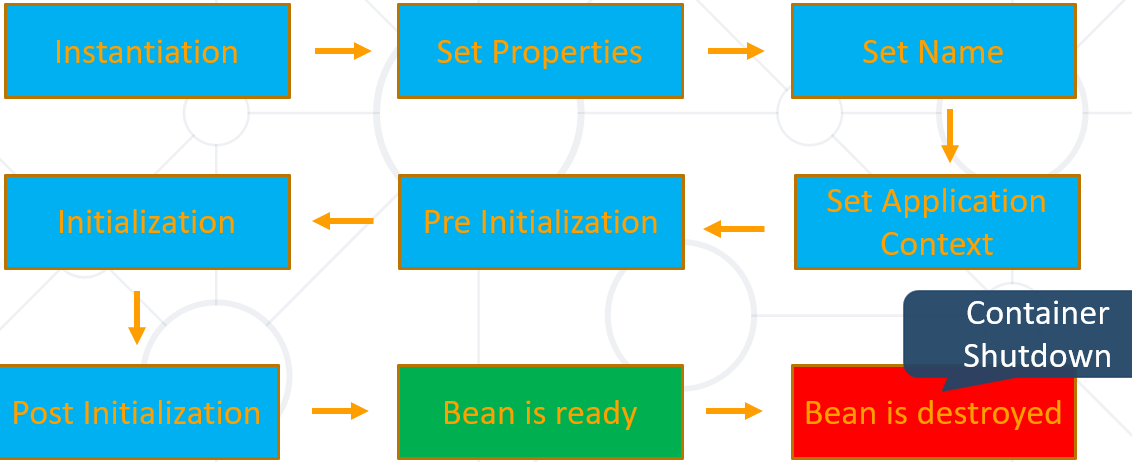
}

**ИЛИ така**

@Service  
**public class** ZooService {  
 **private final** Animal **animal1**;  
 **private final** Animal **animal2**;  
  
 @Autowired  
 **public** ZooService(@Qualifier(**"mySuperDog"**) Animal animal1,  
 @Qualifier(**"normalDog"**) Animal animal2) {  
 **this**.**animal1** = animal1;  
 **this**.**animal2** = animal2;  
 }  
  
 **public void** doWork() {  
 **animal1**.makeNoise();  
 **animal2**.makeNoise();  
 }

}

### Bean Lifecycle



### Bean Lifecycle Demo

По default, инициализацията на всички beans е eager. Щом сме създали анотацията и тя се ползва, то няма логика да го правим lazy (но има опция и за lazy).

### PostConstruct Annotation

* Spring calls methods annotated with *@****PostConstruct*** only once, just after the initialization of bean

**import** javax.annotation.PostConstruct;  
  
**public class** Dog **implements** Animal{  
 **private final boolean superDog**;  
  
 **public** Dog(){  
 **this**(**false**);  
 }  
  
 **public** Dog(**boolean** superDog){  
 **this**.**superDog** = superDog;  
 }  
  
 @Override  
 **public void** makeNoise() {  
 **if** (**superDog**) {  
 System.***out***.println(**"Super Bark super bark"**);  
 } **else** {  
 System.***out***.println(**"Bark bark"**);  
 }  
 }  
  
 **@PostConstruct**  
 **public void** afterInit() {  
 System.***out***.println(**"Dog is ready to bite"**);  
 }  
}

**По-добре да ползваме @PostConstruct анотацията (която е извън Spring) вместо BeanNameAware, BeanFactoryAware или InitializingBean (които са изцяло Spring).**

### BeanNameAware Interface

* BeanNameAware makes the object aware of the bean name defined in the container

**import javax.annotation.PostConstruct;**  
  
**public class** Dog **implements** Animal, **BeanNameAware** {  
 **private final boolean superDog**;  
  
 **public** Dog() {  
 **this**(**false**);  
 }  
  
 **public** Dog(**boolean** superDog) {  
 **this**.**superDog** = superDog;  
 }  
  
 @Override  
 **public void** makeNoise() {  
 **if** (**superDog**) {  
 System.***out***.println(**"Super Bark super bark"**);  
 } **else** {  
 System.***out***.println(**"Bark bark"**);  
 }  
 }  
  
 @PostConstruct  
 **public void** afterInit() {  
 System.***out***.println(**"Dog is ready to bite"**);  
 }  
  
 @Override  
 **public void** setBeanName(String name) {  
 System.***out***.println(**"The name of this Dog bean is: "** + name);  
 }  
}

@Configuration  
**public class** ZooConfig {  
@Bean(name = **"cat"**)  
 **public** Animal cat() {  
 **return new** Cat();  
 }  
  
 @Bean(**"normalDog"**)  
 **public** Animal dog() {  
 **return new** Dog();  
 }  
  
 @Bean(name = **"mySuperDog"**)  
 **public** Animal superDog() {  
 *//****todo: add superpower to this dog* return new** Dog(**true**);  
 }  
}

### BeanFactoryAware Interface

* BeanFactoryAware is used to **inject** the **BeanFactory object**
* With the **setBeanFactory()** method, we assign the BeanFactory reference from the IoC container to the beanFactory property

**public class** MyBeanFactory **implements BeanFactoryAware** {  
 **private** BeanFactory **beanFactory**;  
  
 @Override  
 **public void** setBeanFactory(BeanFactory beanFactory) **throws** BeansException {  
 **this**.**beanFactory** = beanFactory;  
 }  
  
 **public void** getMyBeanName() {  
 MyBeanName myBeanName = **beanFactory**.getBean(MyBeanName.**class**);  
 System.***out***.println(**beanFactory**.isSingleton(**"myCustomBeanName"**));  
 }  
}

**public class** MyBeanName **implements** BeanNameAware {  
 @Override  
 **public void** setBeanName(String beanName) {  
 System.***out***.println(beanName);  
 }  
}  
@Configuration  
**public class** Config {  
 @Bean (name = **"myCustomBeanName"**)  
 **public** MyBeanName getMyBeanName() {  
 **return new** MyBeanName();  
 }  
}

### InitializingBean Interface

* For bean implemented **InitializingBean**, it will run **afterPropertiesSet()** after all bean properties have been set

@Component  
**public class** InitializingBeanExampleBean **implements InitializingBean** {  
 **private static final** Logger ***LOG*** = Logger.getLogger(InitializingBeanExampleBean.**class**);  
  
 @Autowired  
 **private** Environment **environment**;  
  
 @Override  
 **public void** afterPropertiesSet() **throws** Exception {  
 ***LOG***.info(Arrays.asList(**environment**.getDefaultProfiles()));  
 }  
}

### PreDestroy Annotation

* A method annotated with **@PreDestroy** runs only once, just before Spring removes our bean from the application context

**import** javax.annotation.PreDestroy;

@Component  
**public class** UserRepository {  
 **private** DbConnection **dbConnection**;   
   
 **@PreDestroy**  
 **public void** preDestroy() {  
 **dbConnection**.close();  
 }  
}

**По-добре да ползваме @PreDestroy анотацията (която е извън Spring) вместо DisposableBean (които са чaсти Srping).**

### DisposableBean Interface

* For bean implemented **DisposableBean**, it will run **destroy()** after Spring container releases the bean

**public class** Dog **implements** Animal, BeanNameAware, **DisposableBean** {  
 **private final boolean superDog**;  
  
 **public** Dog() {  
 **this**(**false**);  
 }  
  
 **public** Dog(**boolean** superDog) {  
 **this**.**superDog** = superDog;  
 }  
  
 @Override  
 **public void** makeNoise() {  
 **if** (**superDog**) {  
 System.***out***.println(**"Super Bark super bark"**);  
 } **else** {  
 System.***out***.println(**"Bark bark"**);  
 }  
 }  
  
 @PostConstruct  
 **public void** afterInit() {  
 System.***out***.println(**"Dog is ready to bite"**);  
 }  
  
 @Override  
 **public void** setBeanName(String name) {  
 System.***out***.println(**"The name of tgis Dog bean is: "** + name);  
 }  
  
 @Override  
 **public void** destroy() **throws** Exception {  
 System.***out***.println(**"Dog is about to die... Bye!"**);  
 }  
}

### Common Application Properties

* Various properties can be specified inside your **application.yaml** file
* Property contributions can come from **additional jar files**
* You can define your **own properties**
* [**Link to documentation**](https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html)

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

### Application Properties Example

#### application.properties

**spring.datasource.driverClassName**=**com.mysql.cj.jdbc.Driver  
spring.datasource.url**=**jdbc:mysql://localhost:3306/thymeleaf\_adv\_lab\_exam\_db?createDatabaseIfNotExist=true  
spring.datasource.username**=**root  
spring.datasource.password**=**12345  
spring.jpa.properties.hibernate.dialect** = **org.hibernate.dialect.MySQL8Dialect  
spring.jpa.properties.hibernate.format\_sql** = **TRUE  
spring.jpa.hibernate.ddl-auto** = **update  
spring.jpa.open-in-view**=**false  
logging.level.org** = **WARN  
logging.level.blog** = **WARN  
logging.level.org.hibernate.SQL** = **DEBUG  
logging.level.org.hibernate.type.descriptor** = **TRACE  
server.port**=**8000**

**Изпълняване на SQL заявки при стартиране (execute SQL / run SQL automatically)**

*#spring.datasource.initialization-mode = always  
#spring.sql.init.mode = always  
  
#spring.datasource.data=classpath:insert-data.sql  
#spring.sql.init.data-locations=classpath:insert-data.sql*

### Application Yaml Example 1

По-лесно се чете .yaml формата

#### Application.yaml

**spring**:  
 **datasource**:  
 **driverClassName**: com.mysql.cj.jdbc.Driver  
 **url**: **url**: jdbc:mysql://localhost:3306/intro?allowPublicKeyRetrieval=true&useSSL=false&createDatabaseIfNotExist=true&serverTimezone=UTC**&useUnicode=true&characterEncoding=UTF-8**

**username**: root  
 **password**:  
 **jpa**:  
 *# Choose either MySQL8 or MySQL5 below  
  
 # for MySQL 5  
 # database-platform: org.hibernate.dialect.MySQL5InnoDBDialect* **database-platform**: org.hibernate.dialect.MySQL8Dialect  
 **hibernate**:  
 **ddl-auto**: create-drop  
 **open-in-view**: false  
 **properties**:  
 **hibernate**:  
 **format\_sql**: true

### Application Yaml Example 2

#### application.yml

**spring**:  
 **datasource**:  
 **driverClassName**: com.mysql.cj.jdbc.Driver  
 **url**: **"jdbc:mysql://localhost:3306/pathfinder?allowPublicKeyRetrieval=true&useSSL=false&createDatabaseIfNotExist=true&serverTimezone=UTC&useUnicode=true&characterEncoding=UTF-8"  
 username**: root  
 **password**:  
 **servlet**:  
 **multipart**:  
 **max-file-size**: 1MB  
 **max-request-size**: 5MB  
 **mvc**:  
 **hiddenmethod**:  
 **filter**:  
 **enabled**: **true  
 sql**: //execute SQL / run SQL from the resources folder - always  
 **init**:  
 **mode**: *always* **jpa**:  
 *# Choose either MySQL 8 or MySQL 5 below  
 # For MySQL 8* **database-platform**: org.hibernate.dialect.MySQL8Dialect  
 *#For MySQL 5  
 #database-platform: org.hibernate.dialect.MySQL5InnoDBDialect* **hibernate**:  
 **ddl-auto**: create-drop  
 **open-in-view**: **false  
 properties**:  
 **hibernate**:  
 **format\_sql**: true  
 **defer-datasource-initialization**: **true***#Cloudinary Properties  
#cloudinary:  
 #api-key:   
 #api-secret:   
 #cloud-name:*

### Convert YAML and Properties file

Има plugin, който може да обръща от .properties към .yml и обратно

### DLL-AUTO

So the list of possible options are,

**validate**: validate the schema, makes no changes to the database.

**update**: update the schema.

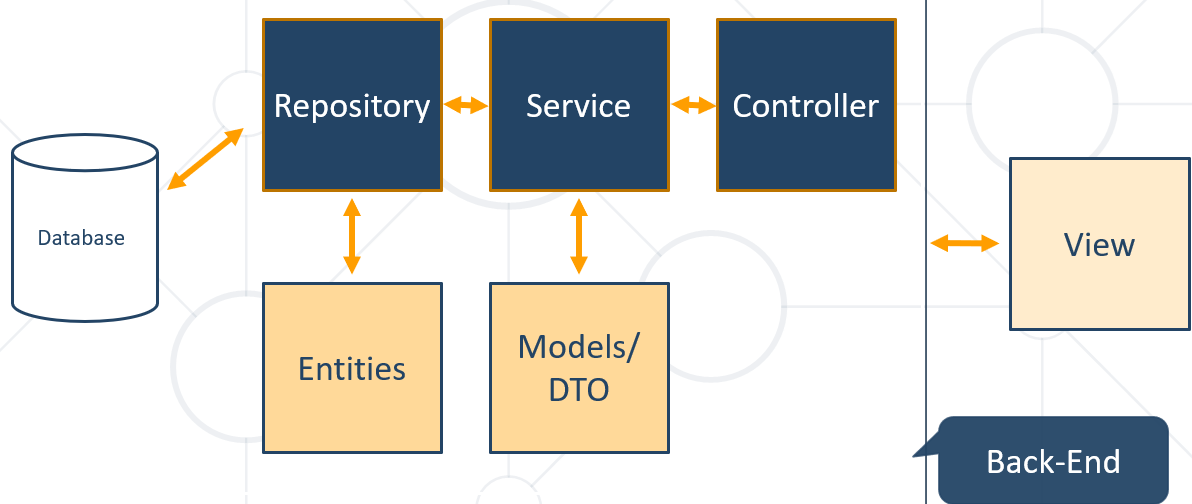
**create**: creates the schema, destroying previous data.

**create-drop**: drop the schema when the SessionFactory is closed explicitly, typically when the application is stopped.

**none**: does nothing with the schema, makes no changes to the database

## 3.2. Spring Data

### Overall Architecture



### Entities

* Entity is a lightweight persistence domain object

Добре е всички DTO (data transfer object), които записват в базата данни или четат от базата данни, то в името им да се съдържа Entity.

@Entity  
@Table(name = **"cats"**)  
**public class** CatEntity {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 **private long id**;  
  
 **private** String **name**;  
 *//GETTERS AND SETTERS*}

### Repositories

* **Persistence** layer that works with **entities**

@Repository  
**public interface** CatRepository **extends** JpaRepository<CatEntity, Long> {  
}

### Services

* **Business Layer** - All the business logic is here.

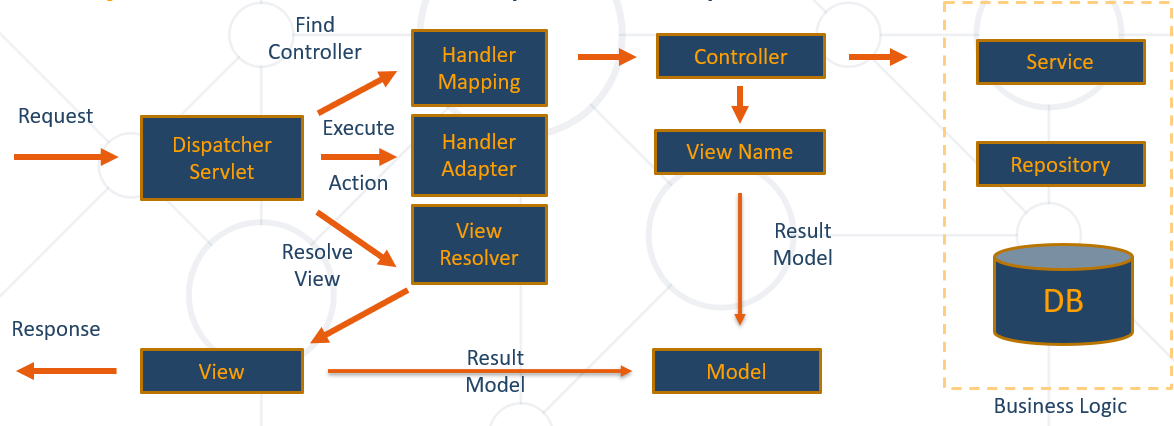
@Service  
**public class** CatServiceImpl **implements** CatService {  
 **private final** CatRepository **catRepository**;  
  
 @Autowired *//не е задължително, даже е излишно* **public** CatServiceImpl(CatRepository catRepository) {  
 **this**.**catRepository** = catRepository;  
 }  
  
 @Override  
 **public void** buyCat(CatModel catModel) { *//****TODO Implement the method }*** }  
}

# 4. MVC Spring Introduction

## 4.1. What is Spring MVC?

### What is Spring MVC?

* **Model-view-controller (MVC)** framework is designed around a **DispatcherServlet** that dispatches requests to handlers



Най-често handler е контролер.

В нашите приложения приемаме, че handler = controller

### What is servlet? – A front-end controller

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

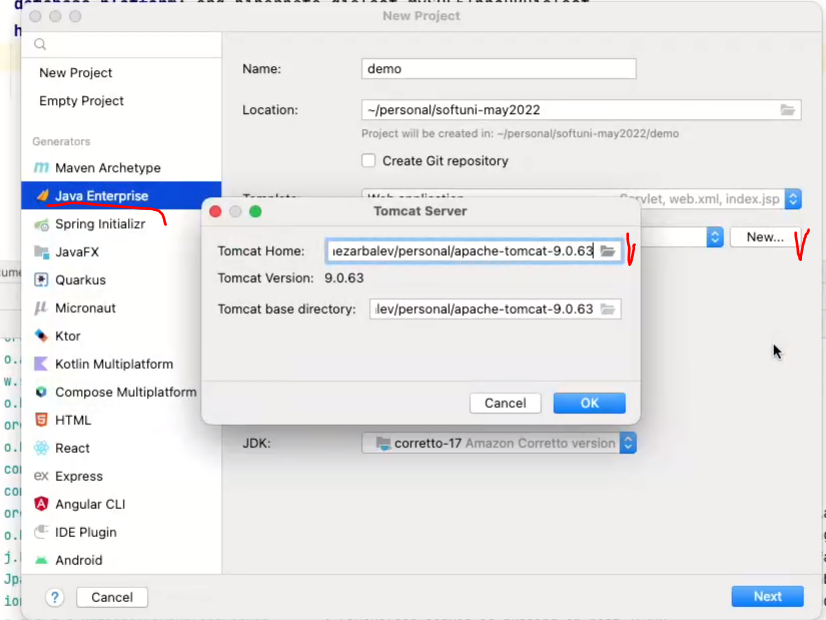
The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

This chapter focuses on writing servlets that generate responses to HTTP requests.

#### Tomcat Demo

1) Unzip Apache Tomcat

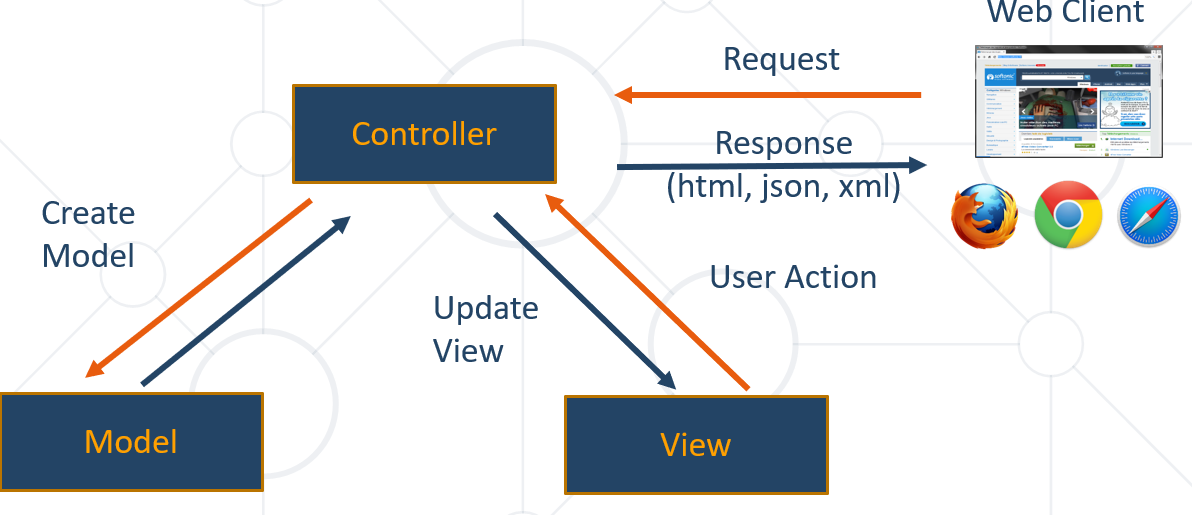
2) Create new project



3) What we generated

**import** java.io.\*;  
**import** javax.servlet.http.\*;  
**import** javax.servlet.annotation.\*;  
  
**@WebServlet**(name = **"helloServlet"**, value = **"/hello-servlet"**)  
**public class** HelloServlet **extends** HttpServlet {  
 **private** String **message**;  
  
 **public void** init() {  
 **message** = **"Hello World!"**;  
 }  
  
 **public void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** IOException {  
 response.setContentType(**"text/html"**);  
  
 *// Hello* PrintWriter out = response.getWriter();  
 out.println(**"<html><body>"**);  
 out.println(**"<h1>"** + **message** + **"</h1>"**);  
 out.println(**"</body></html>"**);  
 }  
  
 **public void** destroy() {  
 }  
}

### MVC – Control Flow



## 4.2. Spring Controllers

### Spring Controllers

* Defined with the **@Controller** annotation

**@Controller**  
**public class** HelloController {  
  
 @GetMapping(**"/hello"**)  
 **public** String hello(){  
 **return "helloworld"**; *//името на template файла*

**return "helloworld.html"**; *//името на template-а*

}  
}

* Controllers can contain multiple actions on different routes

### Request Mapping

* Annotated with **@RequestMapping(**…**)**

@RequestMapping(**"/home"**)  
**public String** home(Model model) {  
 model.addAttribute(**"message"**, “Welcome!**");  
 return "home-view"**;  
}

* Or

@RequestMapping(**"/home"**)  
**public ModelAndView** home(ModelAndView mav) {  
 mav.addObject(**"message"**, “Welcome!**");** mav.setViewName(**"home-view"**);  
 **return** mav;  
}

* Problem when using **@RequestMapping** is that it accepts all types of request methods (get, post, put, delete, head, patch)
* Example - execute on **GET** requests only. We can also use .POST, .PUT, и т.н.

@RequestMapping(value=**"/home"**, method=RequestMethod.GET)  
**public String** home() {  
 **return "home-view"**;  
}

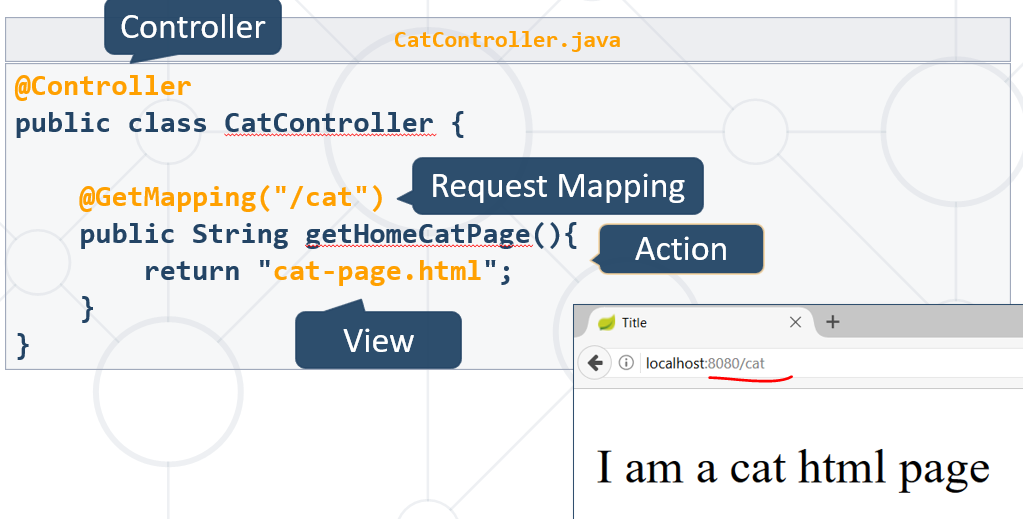
### Get Mapping

* Easier way to create route for a GET request
* This is alias (псевдоним/същото) for **RequestMapping** with method GET

@GetMapping(**"/home"**)  
**public** String home() {  
 **return "home-view"**; *//името на template файла*

**return "home-view.html"**; *//като работим с thymeleaf, то .html не е необходимо*  
}

### Actions – Get Requests



### **Controllers** - @ResponseBody – за REST request

@Controller  
**public class** DogController {  
  
 @GetMapping(**"/dog"**)  
 @ResponseBody //сериализирай кучето като json формат  
 **public** Dog getDogHomePage(){  
 Dog dog = dogService.getBestDog();  
 **return** dog;  
 }  
}

### Post Mapping

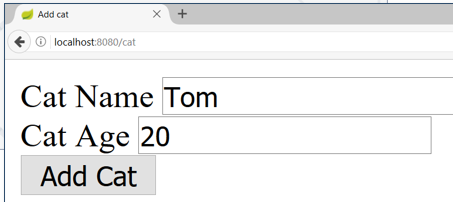
* Similar to the **GetMapping** there is also an alias for **RequestMapping** with method POST

@PostMapping(**"/register"**)  
 **public** String register(UserDTO userDto) {  
 …  
 }

* Similar annotations exist for all other types of request methods

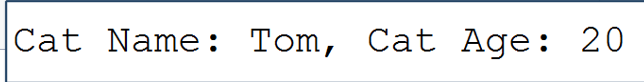
### Actions – Post Requests – анотираме целият контролер

@Controller  
@RequestMapping(**"/cat"**)  
**public class** CatController {  
  
 @GetMapping(**""**)  
 **public** String addCat(){  
 **return "new-cat.html"**;  
 }  
}



### Вземане на данни от html формата

@Controller  
@RequestMapping(**"/cat"**)  
**public class** CatController {  
  
 @PostMapping(**""**) //post заявка с /cat  
 **public** String addCatConfirm(@RequestParam String catName, @RequestParam **int** catAge){  
 System.***out***.println(String.format(  
 **"Cat Name: %s, Cat Age: %d"**, catName, catAge));  
 **return "redirect:/cat"**;  
 }  
}



### Dto вземане на данни от html формата към UserDto

Трябва да съвпадат имената Dto класа полетата и на атрибут name от html кода.

**public class** UserDto {  
 **private** String **fname**;  
 **private** String **lname**;

@Override  
 **public** String toString() {  
 **return "UserDto{"** +  
 **"fname='"** + **fname** + **'\''** +  
 **", lname='"** + **lname** + **'\''** +  
 **'}'**;  
 }  
}

@Controller  
@RequestMapping(**"/user"**)  
**public class** UserController {

@PostMapping  
**public** String createUser(UserDto userDto){  
 System.***out***.println(**"Creating new user ..."** + userDto);  
 **return "usercreated.html"**;  
}

}

newuser.html

<**body**>  
 <**form th:action="@{/user}" method="post"**>  
 <**label for="fname"**>First name:</**label**><**br**>  
 <**input for="text" id="fname" name="fname" value="John"**><**br**>  
   
 <**label for="lname"**>Last name:</**label**><**br**>  
 <**input for="text" id="lname" name="lname" value="Doe"**><**br**><**br**>  
 <**input type="submit" value="Submit"**>  
 </**form**>  
</**body**>

Creating new user ...UserDto{fname='John', lname='Doe'}

### Passing Attributes to View

#### Passing a **String** **model** to the view

@GetMapping(**"/"**)  
**public** String welcome(Model model) {  
 model.addAttribute(**"name"**, **"Pesho"**);  
 **return "index.html"**;  
}

* The **Model** object will be automatically passed to the view as context variables
* Attributes can be accessed from Thymeleaf
* **Model** са данните, които наливаме на View-то (финалната html страница)

#### Passing a **String** **ModelMap** object to the view

@GetMapping(**"/"**)  
**public** String welcome(ModelMap modelMap) {  
 modelMap.addAttribute(**"name"**, **"Pesho"**);

modelMap.put(**"name"**, **"Pesho"**);  
 **return "index"**;  
}

@GetMapping(**"/hello"**)  
**public** String hello(ModelMap modelMap){  
 modelMap.addAttribute(**"num"**, 3);  
 modelMap.put(**"num"**, 3); *//втори вариант за добавяне на атрибут* **return "helloworld.html"**; *//името на template-а*}

* The **ModelMap** object will be automatically passed to the view as context variables

Attributes can be accessed from Thymeleaf

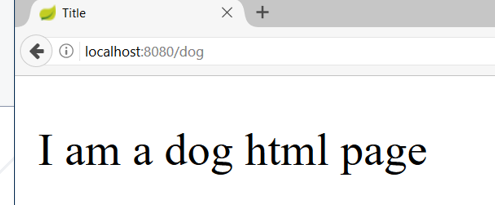
#### Passing a **ModelAndView** object to the view

@GetMapping(**"/"**)  
**public** ModelAndView welcome(ModelAndView modelAndView) {  
 modelAndView.**addObject**(**"name"**, **"Pesho"**);  
 modelAndView.**setViewName**(**"index"**)  
 **return** modelAndView;  
}

* The **ModelAndView** object will be automatically passed to the view as context variables
* Attributes can be accessed from Thymeleaf
* **ModelAndView** е комбиниран вариант, реално View-то е финалната html страница

### Models and Views

@Controller  
**public class** DogController {  
  
 @GetMapping(**"/dog"**)  
 **public** ModelAndView getDogHomePage(ModelAndView modelAndView){  
 modelAndView.setViewName(**"dog-page.html"**);  
 **return** modelAndView;  
 }  
}



### Request Query Parameters

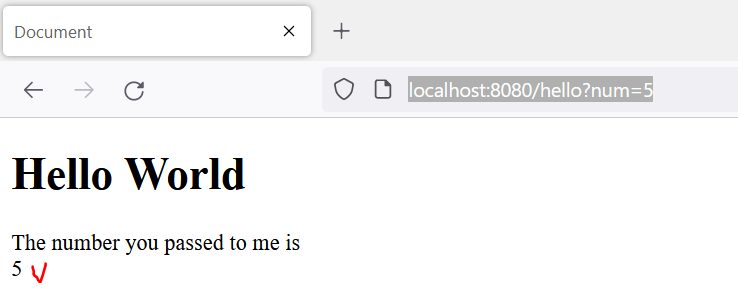
* Getting a parameter from the query string

@GetMapping(**"/details"**)  
**public** String details(@RequestParam(**"id"**) Long id) {  
...  
}

Пример 2:

@GetMapping(**"/hello"**)  
**public** String hello(Model model, @RequestParam(**"num"**) Integer num){  
 model.addAttribute(**"num"**, num);  
 **return "helloworld.html"**; *//името на template-а*}

Пишем в браузъра <http://localhost:8080/hello?num=5>



* **@RequestParam** can also be used to get POST parameters (по-добре да използваме тук Dto)

@PostMapping(**"/register"**)  
**public** String register(@RequestParam(**"name"**) String name) {  
...  
}

### Request Query Parameters with Default Value

* Getting a parameter from the query string when the query parameter is missing

@GetMapping(**"/comment"**)  
**public** String comment(@RequestParam(name=**"author"**, defaultValue = **"Annonymous"**) String author) {  
...  
}

### Making parameter optional – може да го има/може да го няма

@GetMapping(**"/search"**)  
**public** String search(@RequestParam(name=**"sort"**, required = **false**) String sort) {  
...  
}

### Path Variable

* Getting a parameter from the ***path*** part of the URL

@GetMapping(**"/details/{id}"**)  
**public** String details(@PathVariable(**"id"**) Long id) {  
...  
}

@GetMapping(**"/hello/{id}/test"**)  
**public** String hello(Model model, @PathVariable(**"id"**) Integer pathId){  
 model.addAttribute(**"num"**, pathId);  
 **return "hello.html"**;  
}

### Form Objects

* Spring will automatically try to fill objects with a form data – това го показахме вече с Dto по-горе.

@PostMapping(**"/register"**)  
**public** String register(UserDTO userDto) {  
...  
}

* The input field names must be the same as the object field names – Трябва да съвпадат имената Dto класа полетата и на атрибут name от html кода.

### Redirecting

* Redirecting after POST request

@PostMapping(**"/register"**)  
**public** String register(UserDTO userDto) {  
...  
 **return "redirect:/login"**;  
}

### Redirecting with Query Parameters

* Redirecting with query string parameters

@PostMapping(**"/register"**)  
**public** String register(UserDTO userDto,  
 RedirectAttributes redirectAttributes) {  
  
 redirectAttributes.addAttribute(**"errorId"**, 3);  
 **return "redirect:/login"**; //?errorId = 3  
}

### Redirecting with Attributes

* Keeping objects after redirect

@PostMapping(**"/register"**)  
**public** String register(@ModelAttribute UserDTO userDto,  
 RedirectAttributes redirectAttributes) {  
...  
 redirectAttributes.**addFlashAttribute**(**"userDto"**, userDto);  
 **return "redirect:/register"**;  
}

## 4.3. Inversion of Control

### Field Injection

* Easy to write
* Easy to add new dependencies
* It **hides** potential architectural **problems**!



### Constructor Injection

* Time Consuming
* Harder to add dependencies
* It **shows** potential architectural problems!



@Autowired  
**public** ControllerA(ServiceA serviceA, ServiceB serviceB, ServiceC serviceC) {  
 **this**.serviceA = serviceA;  
 **this**.serviceB = serviceB;  
 **this**.serviceC = serviceC;  
}

### Setter Injection

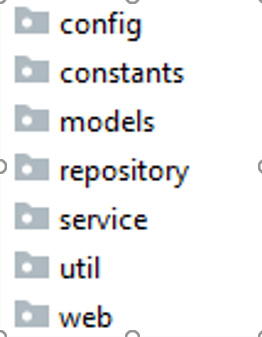
* Create setters for dependencies
* Can be combined easily with constructor injection
* Flexibility in dependency resolution or object reconfiguration!



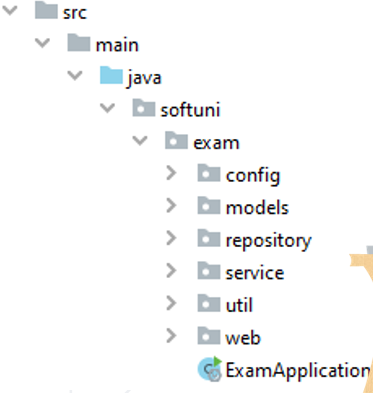
@Service  
**public class** HomeContoller(){  
 *//...* @Autowired  
 **public void** setServiceA(ServiceA serviceA) {  
 **this**.serviceA = serviceA;  
 }  
}

## 4.4. Layers - The Correct Project Structure

* We are used to **splitting** our code based **on its functionality**:
* It gets **hard to navigate** in bigger applications



* **Splitting** the project into **different modules**
  + Each module corresponding to the application layer
  + Makes it easier to navigate



## 4.5. Thin Controllers

* Controllers should follow well known principles such as **DRY** (Do not Repeat Yourself) and **KISS** (keep it simple, stupid)
* Should delegate functionality to the **service** layer
* The **service layer** consists of application logic, e.g. services, executors, strategies, mappers, DTOs, entities, etc.

# 5. State Management

## 5.1. HTTP Cookies

### What Are Cookies?

* A **small file** of **plain text** with no executable code
  + Sent by the server to the client's browser
  + Stored by the browser on the client's device (computer, tablet, etc.)
  + Hold small piece of data for a particular client and a website

### What Are Cookies Used for?

* **Session management**
  + Logins, shopping carts, game scores, or anything else the server should remember
* **Personalization**
  + User preferences, themes, and other custom settings
* **Tracking**
  + Recording and analyzing user behavior
* Breakfast
  + But that’s not what we are currently talking about 😊

В incognito не се пази cookie-то.

### Session Management

* The HTTP object is **stateless –** всеки следващ request не пази нищо за предишните requests
  + It **doesn’t store** information about the requests



### Stateless HTTP – the Problem

* The **server does not know** if two requests come from the same client
* **State management** problems
  + Navigation through pages requires **authentication each time**
  + Information about the pages is lost **between the requests**
  + **Harder personalization** of page functionality

### Stateless HTTP – the Cookie Solution

* A reliable **mechanism** for websites to **remember stateful information**
  + to know whether the user is **logged in or not**
  + to know **which account** the user is logged in with
  + to record the user's **browsing activity**
  + to remember pieces of information **previously entered** into form fields (usernames, passwords(паролата няма да я запишем в cookie 😊), etc.)

### How Are Cookies Used?

Пазим езика на сайта в cookie-то например:

* The request holds the specific web site cookie within the **Cookie** header

**GET /index HTTP/1.1**

**Cookie: lang=en**

* The response holds the cookies to be saved within the **Set-Cookie** header

**HTTP/1.1 200 OK**

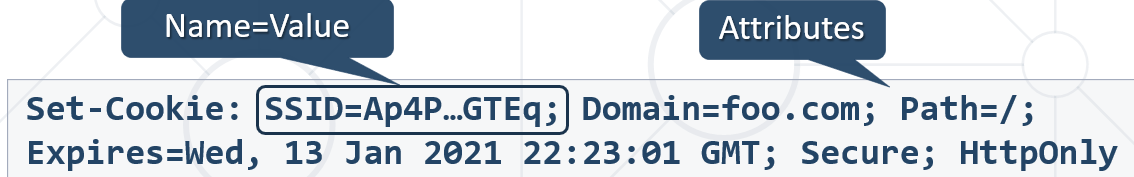
**Set-Cookie: lang=en**

### Server-Client Cookies Exchange



### Cookie Structure

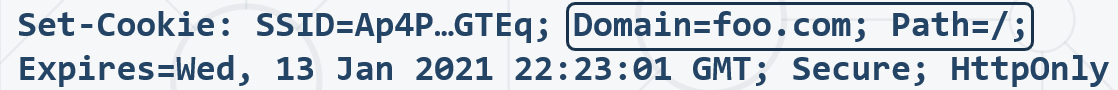
* The cookie consists of **Name**, **Value** and **Attributes** (optional)
* The attributes are **key-value pairs** with additional information
* Attributes are not included in the requests
* Attributes are used by the client to control the cookies



Датата и часа когато cookie-то ще бъде изтрито автоматично.

### Scope

* Defined by the attributes **Domain** and **Path**
* **Domain** – defines the website that the cookie belongs to
* **Path** – Indicates a **URL** path that must exist in the requested resource before sending the **Cookie** header



By Default – домейна идва оттам откъдето идват cookie-тата. Например docs.google.coom

Ако пътят ти е главен (Path=/) – то на всички пътища (от URL-и) ще се предава cookie-то, което е сетнато за главния път.

Ако зададения път е Path=foo.com/test, а ние като заредим foo.com/Pesho, то cookie-то няма да се предаде на/за този път.

### Lifetime

* Defined by the attributes **Expires** and **Max-Age(**секундите, които може да живее това cookie**)**
* **Expires** – defines the date that the browser should delete the cookie
* **By default the cookies are deleted after the end of the session**
* **Max-Age** – interval of seconds before the cookie is deleted



### Security

* Security flags do not have associated values
* **Security** - tells the browser to use cookies only via **secure/encrypted** connections (с катинарче или http**s**)
* **HttpOnly** – defines that the cookie cannot be accessed via client-side scripting languages – недостъпно за всякакъв вид script-ови езици!

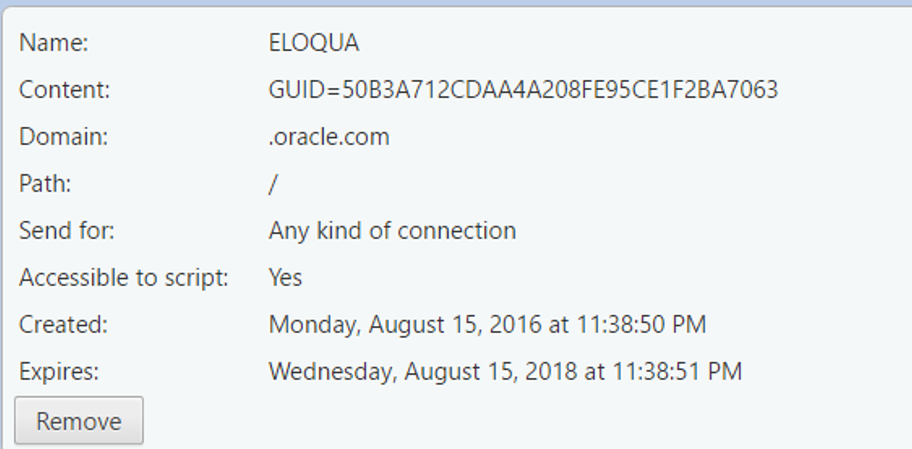


**httpOnly** – ако се помъчим да го прочетем с JS например, то няма да ни разреши. Това е много важно – защото например при CrossSide от другия IP адрес може да изпълнят JS команда, и да ни вземат cookie-тата 😊

### What is in the Cookie?

* The cookie file contains a table with **key-value** pairs

Plug-in to view cookies



### Examine Your Cookies

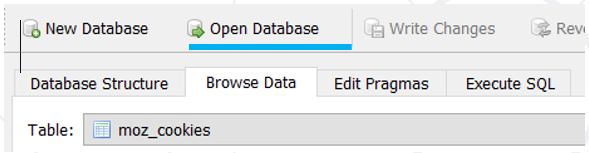
* Most cookies are stored in a **RDBMS**, usually **SQLite**
* Download **SQLite browser** from [**here**](http://sqlitebrowser.org/)
* Location of Mozilla cookies

**C:\Users\{username}\AppData\Roaming\Mozilla\Firefox\Profiles\{name}.default\cookies.sqlite**

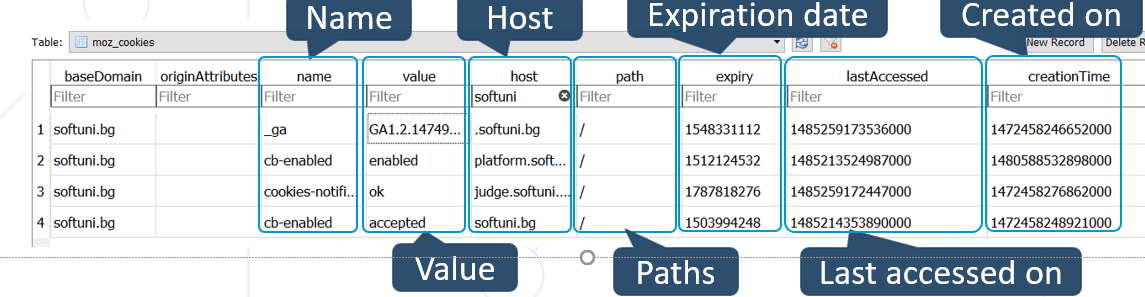
* Location of Chrome cookies

**C:\Users\{username}\AppData\Local\Google\Chrome\User Data\Default\Cookies**

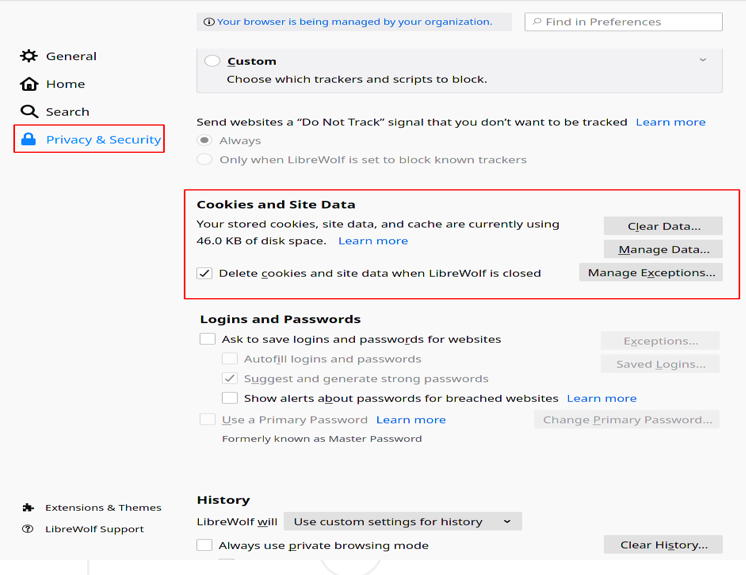
* Open the file with the **SQLite browser**

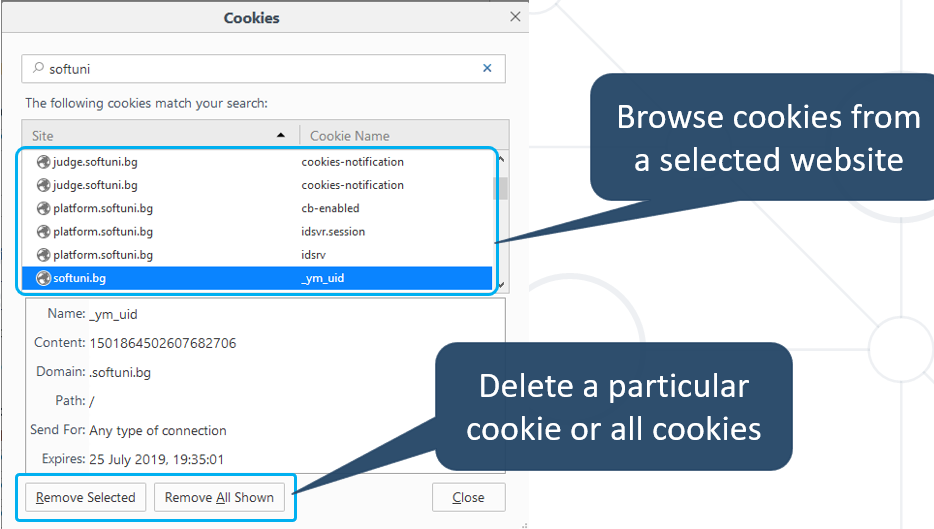


* Browse the cookies table

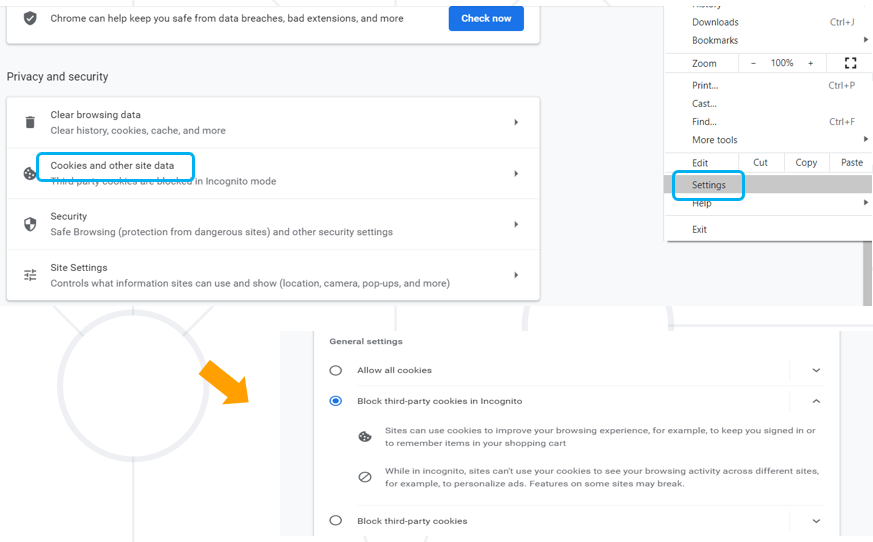


### Control Your Cookies – Firefox Browser

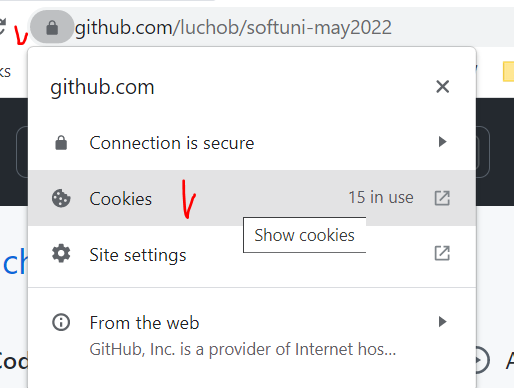


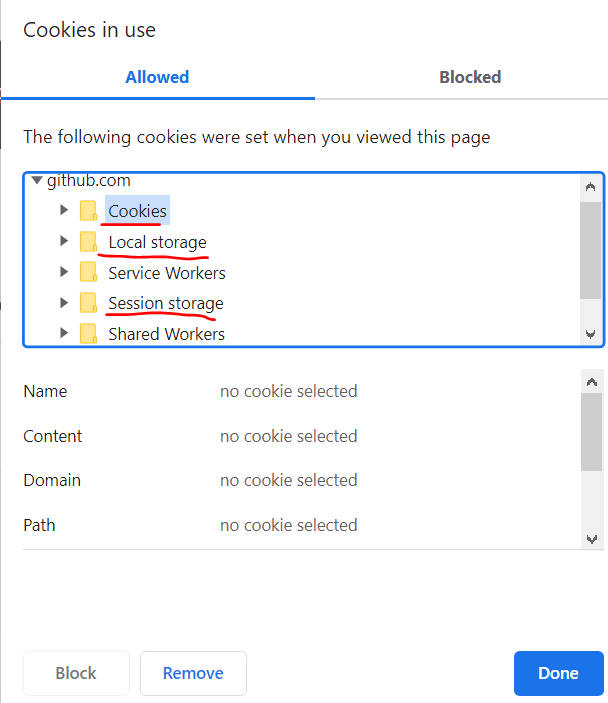


### Control Your Cookies – Chrome Browser



Или оттук:





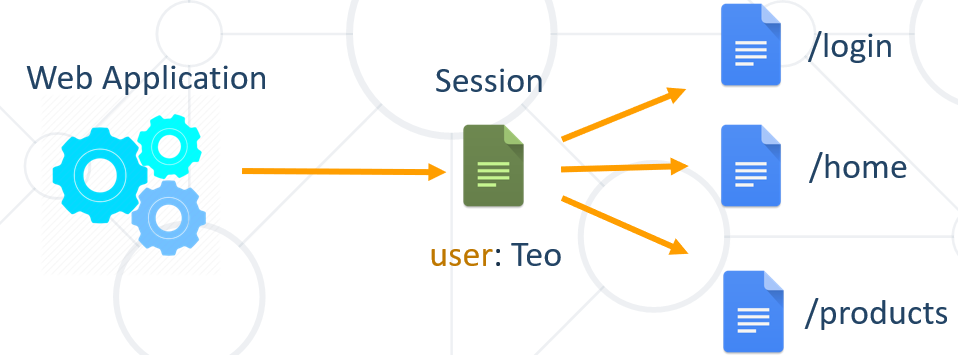
### Third Party Cookies

* Cookies stored by an **external party** (different **domain**)
* Mainly used for advertising and tracking (събират privacy инфо за мен което не е ок) across the web
* By the end of 2023, Google will stop the use of these advertising and tracking third-party cookies **ONLY!**

## 5.2. HTTP Sessions

### What Are Sessions?

* A way to store information about a user to be used across **multiple pages**

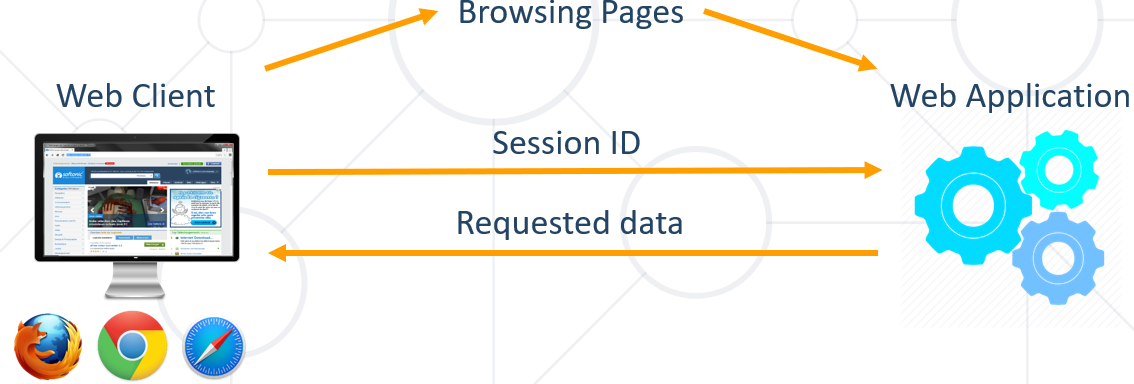


### Session Management

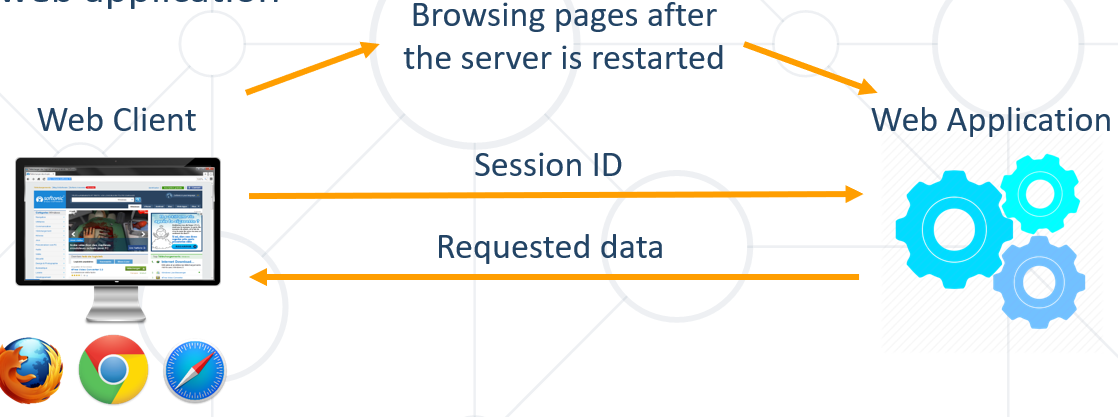
* The exchange mechanism be used between the user and the web application



* The exchange mechanism be used between the user and the web application

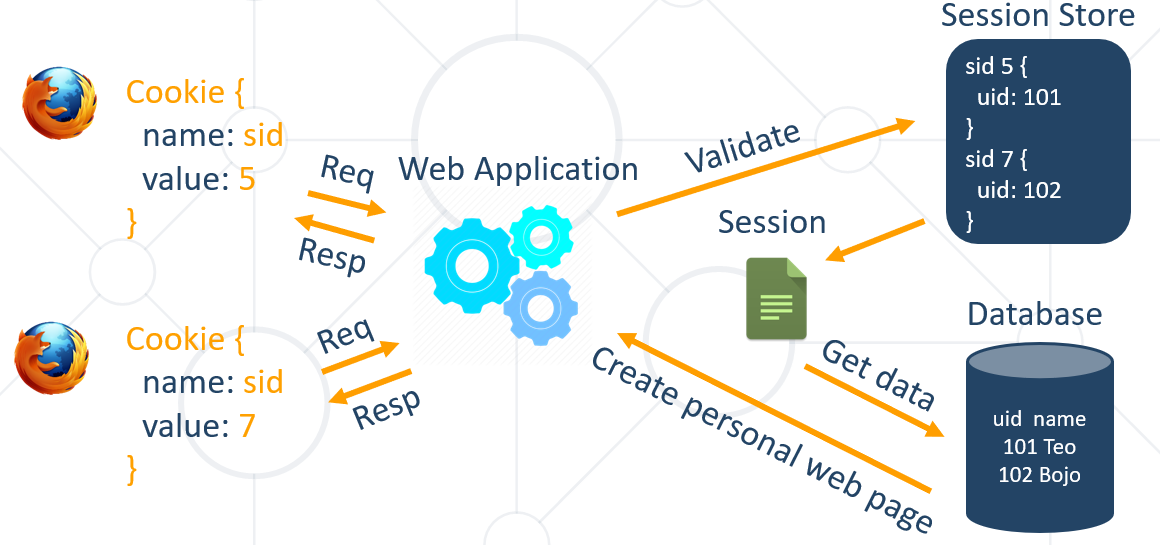


* The exchange mechanism be used between the user and the web application (when server is restarted)

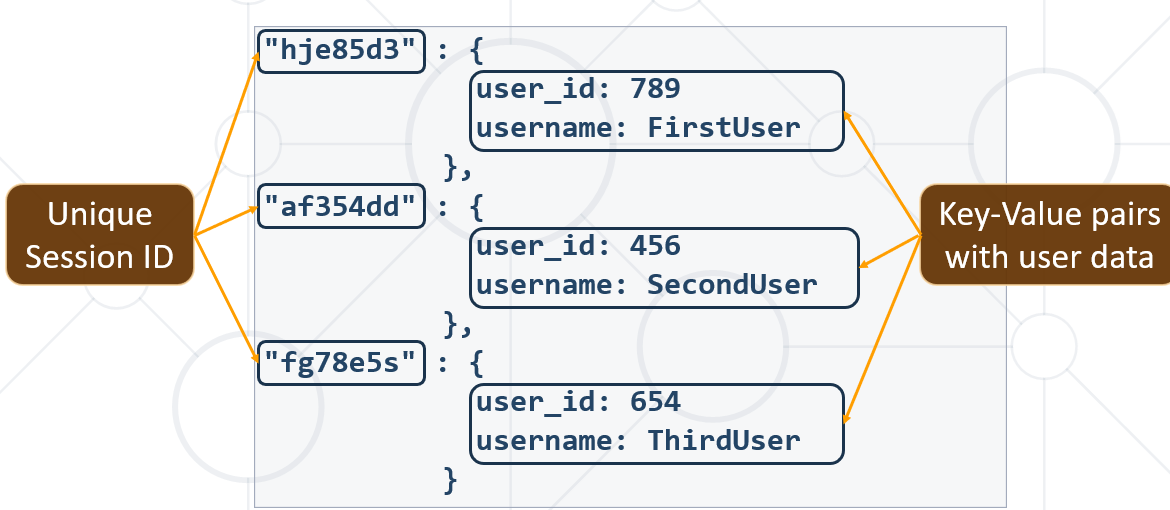


### Relation with Cookies

На back-end server-a има SessionStore И оттам се изпраща към клиента персонализирана информация/персонализирана web страница за всеки логнат потребител.



### Session Structure



## 5.3. Summary

**Cookies** are **client based** stored information

They are created by web applications

Browser sends them back to the application

**Sessions** are **server based** information

They are used across multiple pages

Stores important info about the client

# 20. Other

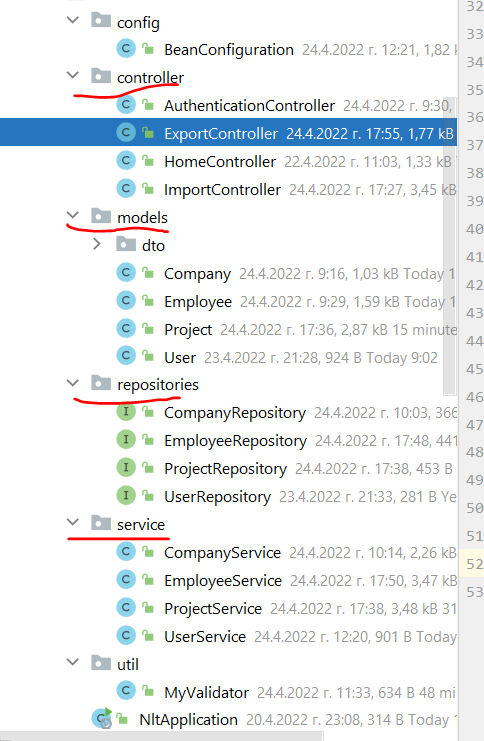
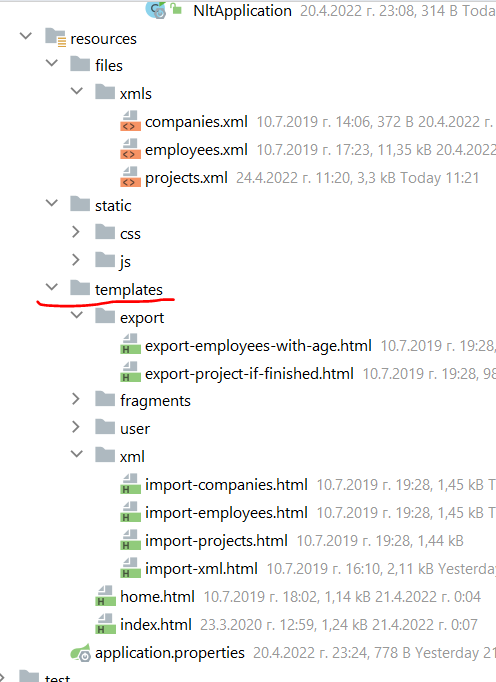
Content-Type: application/x-www-form-urlencoded

@PostMapping(value = **"/users/register"**, consumes = MediaType.***APPLICATION\_FORM\_URLENCODED\_VALUE***)  
**public** String doRegister(@RequestBody MultiValueMap<String, String> paramMap){  
 System.***out***.println(paramMap.get(**"user"**));  
  
 **return "user/register"**;  
}

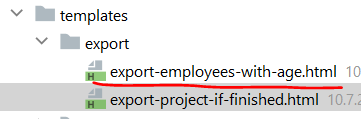
@PostMapping(value = **"/users/register"**, consumes = MediaType.***APPLICATION\_JSON\_VALUE***)  
**public** String doRegister(@RequestBody RegistrationDTO dto){  
 System.***out***.println(dto);  
  
 **return "user/register"**;  
}

@PostMapping(value = **"/users/register"**)  
 **public** String doRegister(**@Valid** RegistrationDTO dto, BindingResult validationResult) {  
*// RegistrationDTO dto = new RegistrationDTO("user", "pass", "pass", "mail@abv.bg");* **if** (validationResult.hasErrors()) {  
 **return "user/register"**;  
 }  
  
 **userService**.register(dto);  
  
 **return "user/login"**;  
 }

Пример за контролер

@Controller  
**public class** ExportController {  
 **private final** ProjectService **projectService**;  
 **private final** EmployeeService **employeeService**;  
 **private final** Gson **gson**;  
  
 @Autowired  
 **public** ExportController(ProjectService projectService, EmployeeService employeeService, Gson gson) {  
 **this**.**projectService** = projectService;  
 **this**.**employeeService** = employeeService;  
 **this**.**gson** = gson;  
 }  
  
 @GetMapping(**"/export/project-if-finished"**)  
 **public** ModelAndView showFinishedProjects(){  
 ModelAndView modelAndView = **new** ModelAndView(**"export/export-project-if-finished"**);  
  
 String result = **this**.**projectService**.getFinishedProducts();  
  
 modelAndView.addObject(**"projectsIfFinished"**, result);  
  
 **return** modelAndView;  
 }  
  
 @GetMapping(**"/export/employees-above"**)  
 **public** ModelAndView showEmployeesAbove25(){  
 ModelAndView modelAndView = **new** ModelAndView(**"export/export-employees-with-age"**);  
  
 List<ExportEmployeeDTO> employeesAbove25 = **this**.**employeeService**.getEmployeesAbove25();  
  
 StringBuilder sb = **new** StringBuilder();  
 **this**.**gson**.toJson(employeesAbove25, sb);  
  
 modelAndView.addObject(**"employeesAbove"**, sb.toString());  
  
 **return** modelAndView;  
 }  
}



<!DOCTYPE **html**>  
<**html lang="en" xmlns="http://www.w3.org/1999/xhtml"  
 xmlns:th="http://www.thymeleaf.org"**>  
  
<**head**>  
 <**th:block th:include="~{fragments/head}"**></**th:block**>  
</**head**>  
<**body**>  
<**header**>  
 <**th:block th:include="~{fragments/nav-bar}"**></**th:block**>  
</**header**>  
<**main**>  
 <**div class="jumbotron"**>  
 <**form** >  
 <**div class="col-md-12 d-flex justify-content-center"**>  
 <**label class="display-4" for="export-project-textarea"**>Employees with age above 25:</**label**>  
 </**div**>  
 <**div class="col-md-12 d-flex justify-content-center"**>  
 <**div class="col-md-8 d-flex justify-content-center"**>  
 <**textarea class="form-control" id="export-project-textarea" rows="20" th:text="${employeesAbove}" readonly**></**textarea**>  
 </**div**>  
 </**div**>  
 </**form**>  
 </**div**>  
</**main**>  
<**footer**>  
 <**th:block th:include="~{fragments/footer}"**></**th:block**>  
</**footer**>  
</**body**>  
</**html**>

<**div class="form-group"**>  
 <**label for="number" class="col-lg-2 control-label"**>Number</**label**>  
 <**div class="col-lg-10"**>  
 <**input type="text" autofocus="autofocus" name="number" title="Number" class="form-control"  
 pattern="\+?[0,9]{6,}" //валидира за телефонен номер с регекс  
 id="number"**/>  
 </**div**>  
</**div**>

## Uploading a file

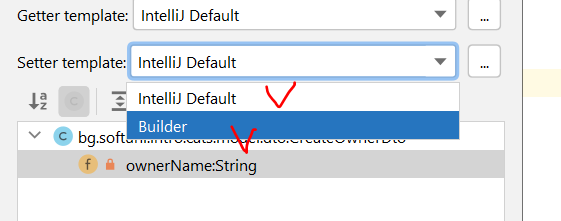
@PostMapping(**"/"**) *//покажи всички***public** String storeContact(Contact contact, @RequestParam(**"picture"**)**MultipartFile** picture) {  
 **this**.**contacts**.add(contact);  
  
 **return "redirect:/"**; *//върни ме на началната страница*}

<**div class="form-group"**>  
 <**label for="picture" class="col-lg-2 control-label"**>Picture</**label**>  
 <**div class="col-lg-10"**>  
 <**input type="file" autofocus="autofocus" name="picture" title="Picture" class="form-control"  
 id="picture"**/>  
 </**div**>  
</**div**>

Websocket-и работят с постоянни данни – подходящо за чатове!!!

Tomcat е на apache

**Builder pattern and Alt + Insert**



Когато искаме да работи с 2 service-а или 2 ModelMapper-и, то използваме **@Qualifier**

* + - веднъж на класа/метода който разписва логиката, и втори път на по-горна инстанция от InversionOfControl структурата го викаме този Qualifier.

How to download a specific folder from a github repository - <https://download-directory.github.io/>

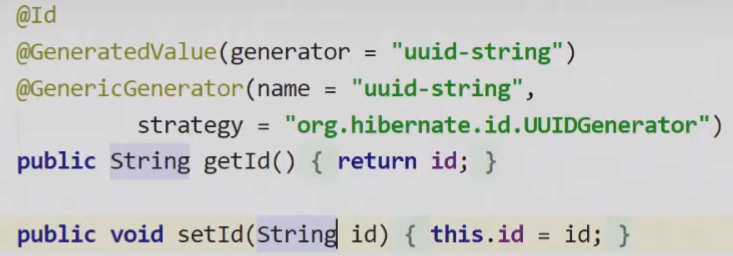
## UUID

**UUID** – A universally unique identifier (UUID) is a 128-bit label used for information in computer systems. The term globally unique identifier (GUID) is also used. When generated according to the standard methods, UUIDs are, for practical purposes, unique.

**Не е добра идея да слагаме UUID като primary key – прекалено бавно стават insert-е поради проблем с A-B дърво алгоритми.**

UUID е глобално уникално в света - част от алгоритъма на UUID е вземане на IP-то на машината и други неща специфични за даден компютър. Не е проблем да наливаме данни от две бази с различно IP.

UUID дава privacy, не точно security – коя поръчка кой има право да я вижда в URL-то примерно.



**import** org.hibernate.annotations.Type;  
  
**import** javax.persistence.Column;  
**import** javax.persistence.Entity;  
**import** javax.persistence.Id;  
**import** javax.persistence.Table;  
**import** java.util.UUID;  
  
@Entity  
@Table(name = **"offers"**)  
**public class** OfferEntity {  
 @Id  
 @GeneratedValue(generator = **"UUID"**)  
 @GenericGenerator(  
 name = **"UUID"**,  
 strategy = **"org.hibernate.id.UUIDGenerator"** )  
 @Type(type = **"uuid-char"**) *//Hibernate annotation @Type //работи, но не е достатъчно  
// @Column(columnDefinition = "VARCHAR(255)") //по този начин не е редно да работим с UUID* **private** UUID **id**;