Spring Web

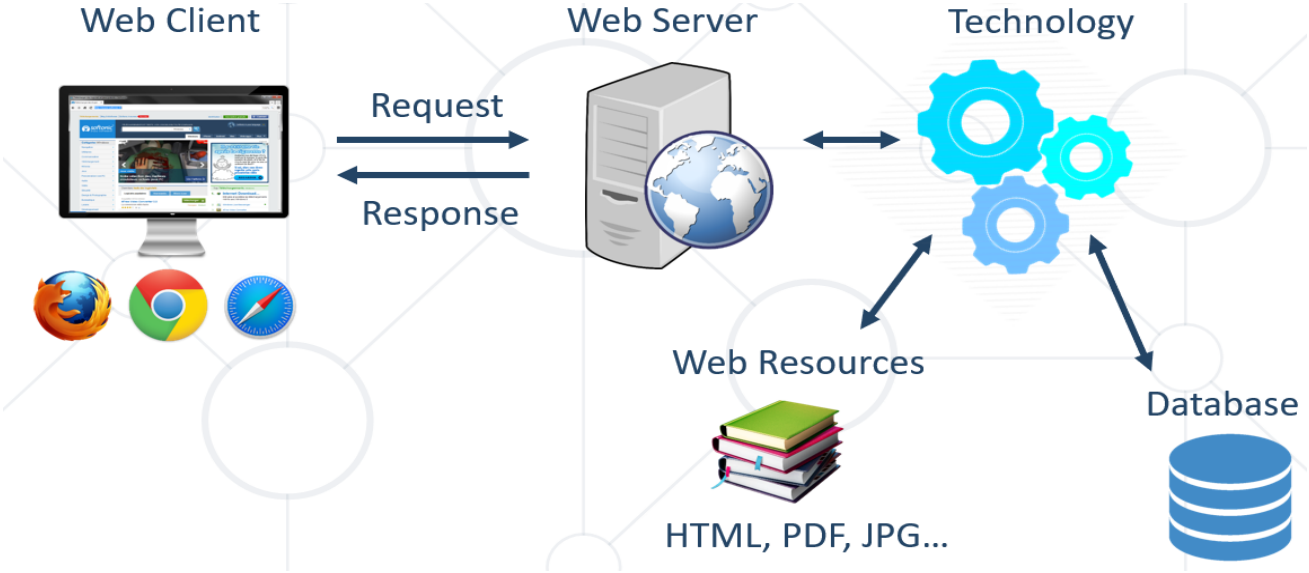
1. Internet Explained

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** – 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

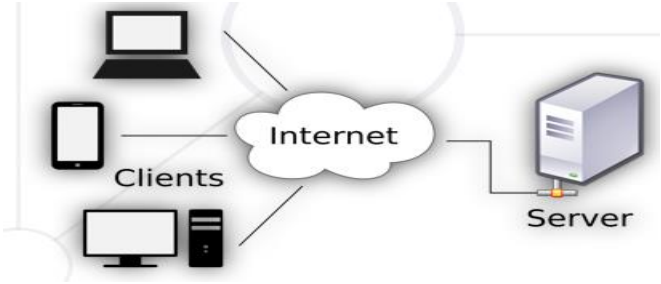


Important Definitions

- 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

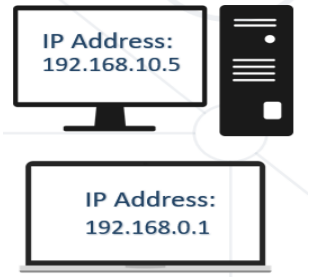
1.3. Packets (late 1960s)

- 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**
- They are called **IP Addresses**
- The IP address is **unique** to each computer or a device at the edge of the network

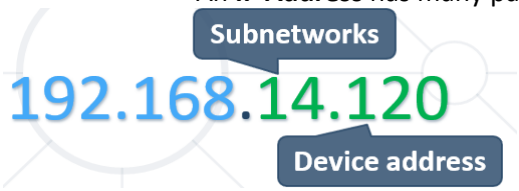


IPv4

- **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

```
Class A
  0.   0.   0.   0 = 00000000.00000000.00000000.00000000
127.255.255.255 = 01111111.11111111.11111111.11111111
                  0nnnnnnn.HHHHHHHH.HHHHHHHH.HHHHHHHH

Class B
128.   0.   0.   0 = 10000000.00000000.00000000.00000000
191.255.255.255 = 10111111.11111111.11111111.11111111
                  10nnnnnn.nnnnnnnn.HHHHHHHH.HHHHHHHH

Class C
192.   0.   0.   0 = 11000000.00000000.00000000.00000000
223.255.255.255 = 11011111.11111111.11111111.11111111
                  110nnnnn.nnnnnnnn.nnnnnnnn.HHHHHHHH

Class D
224.   0.   0.   0 = 11100000.00000000.00000000.00000000
239.255.255.255 = 11101111.11111111.11111111.11111111
                  1110XXXX.XXXXXXXX.XXXXXXXX.XXXXXXXX

Class E
240.   0.   0.   0 = 11110000.00000000.00000000.00000000
255.255.255.255 = 11111111.11111111.11111111.11111111
                  1111XXXX.XXXXXXXX.XXXXXXXX.XXXXXXXX
```

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

CIDR	IP address range	Class
10.0. 0.0/8	10.0. 0.0 – 10.255. 255.255	A
172.16. 0.0/12	172.16. 0.0 – 172.31. 255.255	B
192.168. 0.0/16	192.168. 0.0 – 192.168. 255.255	C

IPv6

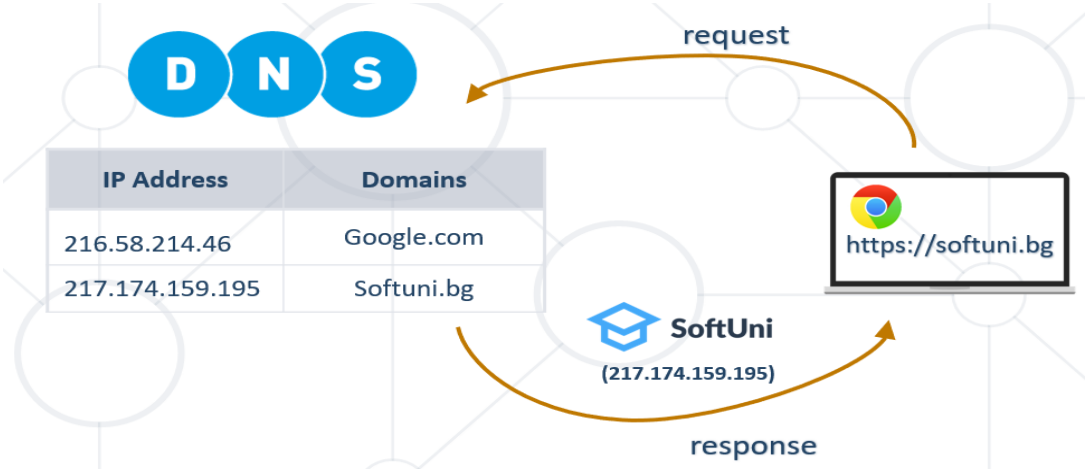
- **IPV6** uses **128 bits** - 340 undecillion unique addresses
  - That's more than the atoms on the surface of the Earth
- These **128** bits are organized into eight 16 bit sections
- Each 16 bit block is converted to hexadecimal and it's separated with a colon
- This is a full IPV6 address:
  - **3FFE:F200:0234:AB00:0123:4567:8901:ABCD**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



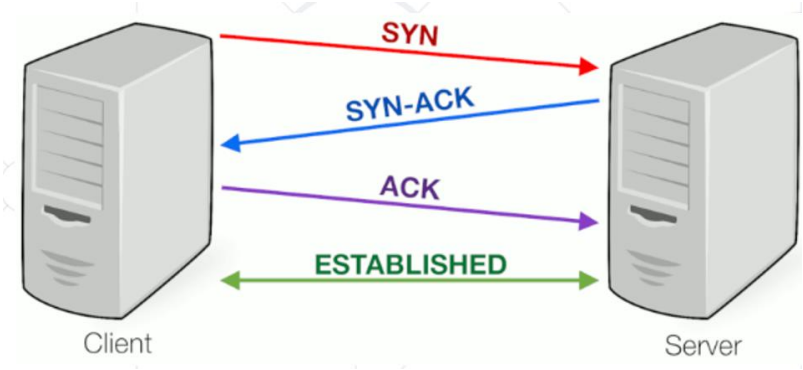
1.5. Reliability and TCP (Transmission Control Protocol)

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

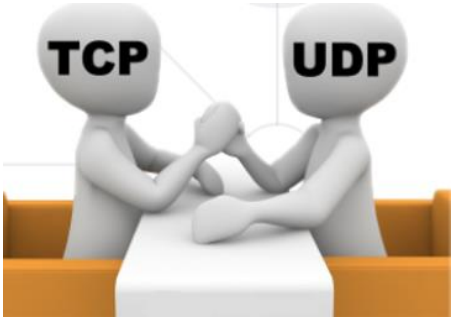
TCP - Transmission Control Protocol

- **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

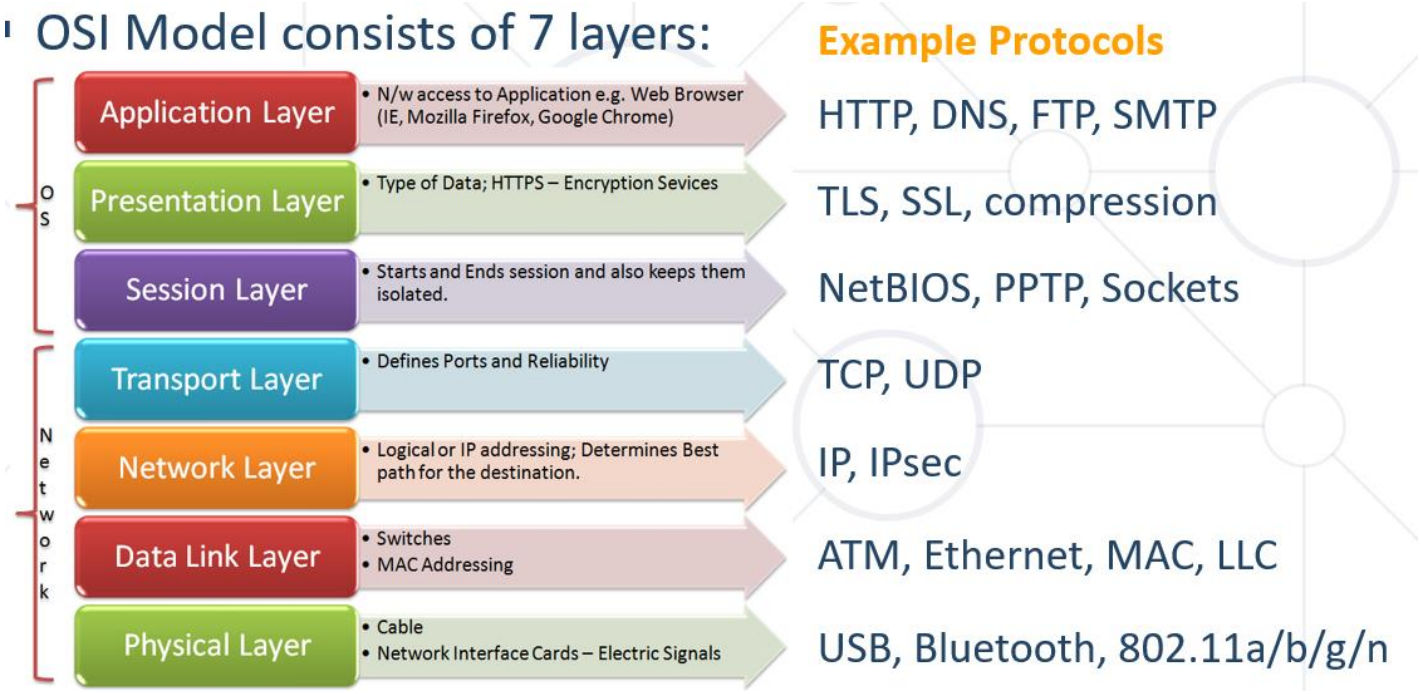
1.6. The OSI Model (Open System Interconnect)

What is the OSI Model?

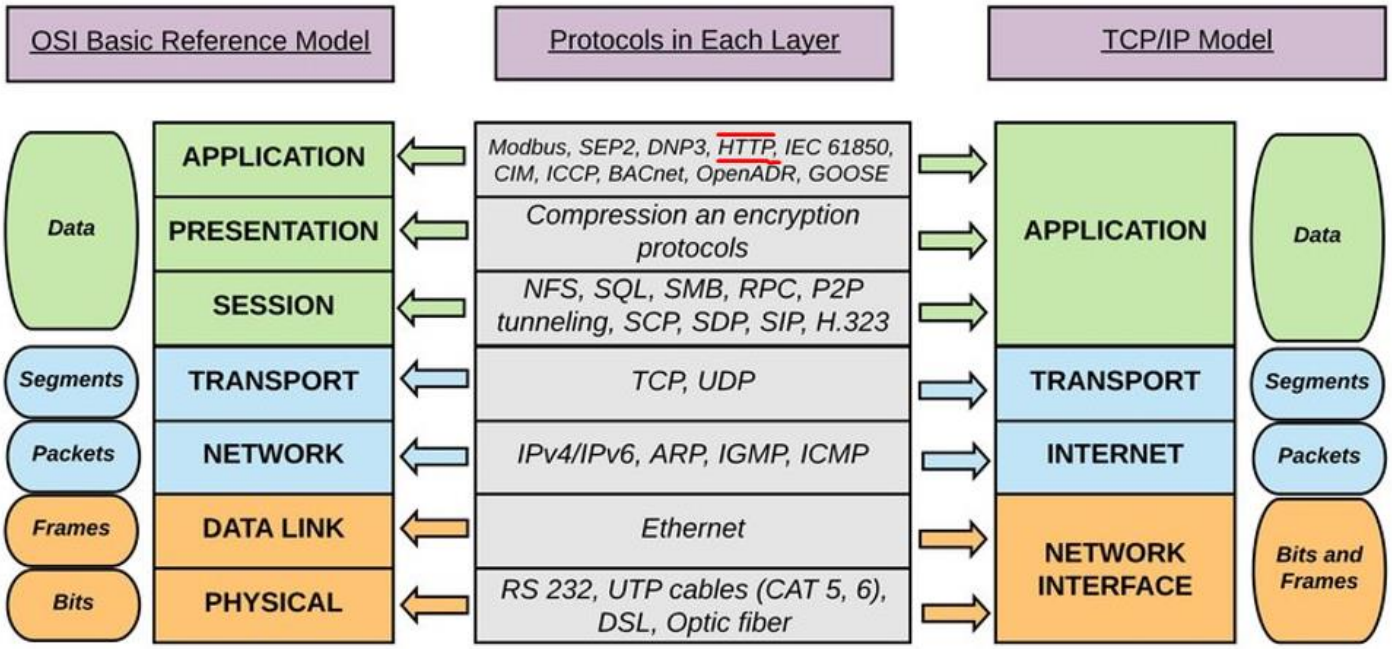
- **OSI** model stands for **O**pen **S**ystem **I**nterconnect
- It consists of 7 layers
  - Each layer serves the layer above it and in return, is served by the layer below it
- Understanding each layer of the model helps us with:
  - Troubleshooting
  - Communicating better with technical and non-technical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

OSI Layers

- OSI Model consists of 7 layers:

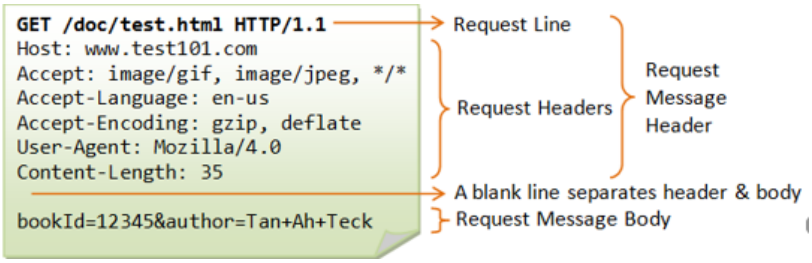


TCP/IP model mapping to OSI



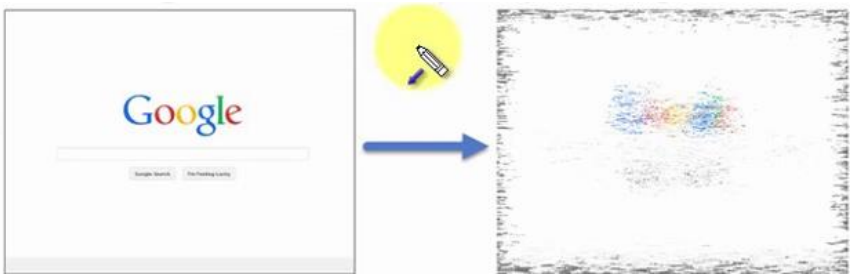
Application Layer – level 7

- 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)**



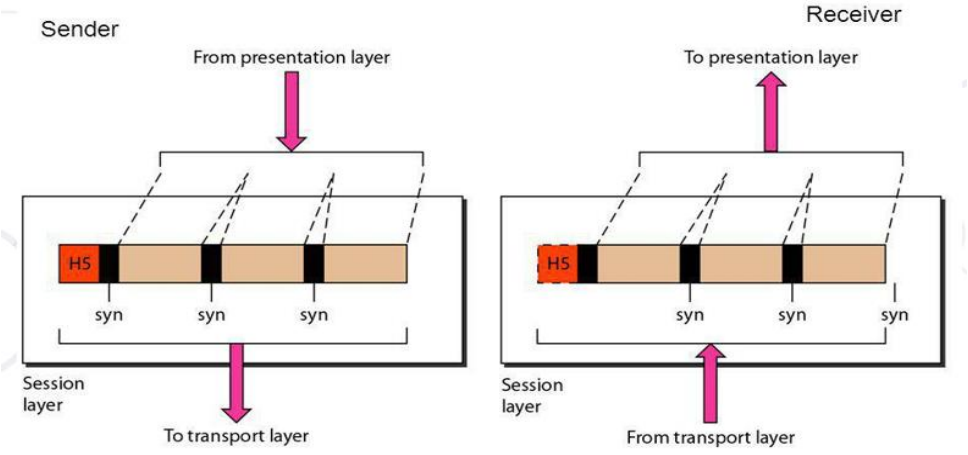
Presentation Layer – level 6

- 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



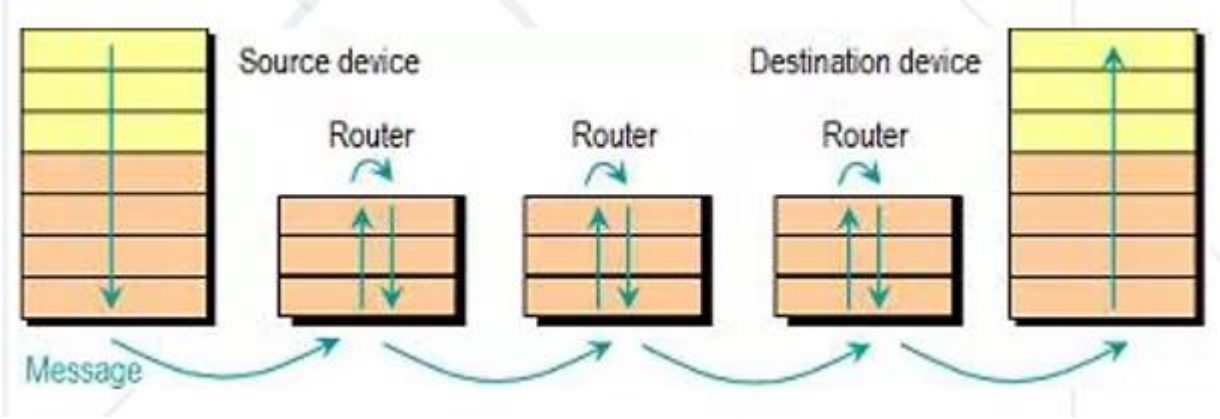
Transport Layer – level 4

- 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**

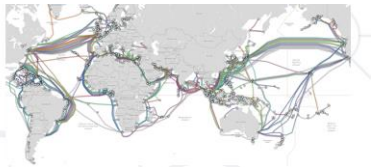
1.7. Network Hardware

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

Изначало HTTP Protocol е бил stateless – backend-server-а не трябва да пази информацията от предходната заявка на предхония или същия клиент. След това, с времето, обаче се променят нещата.

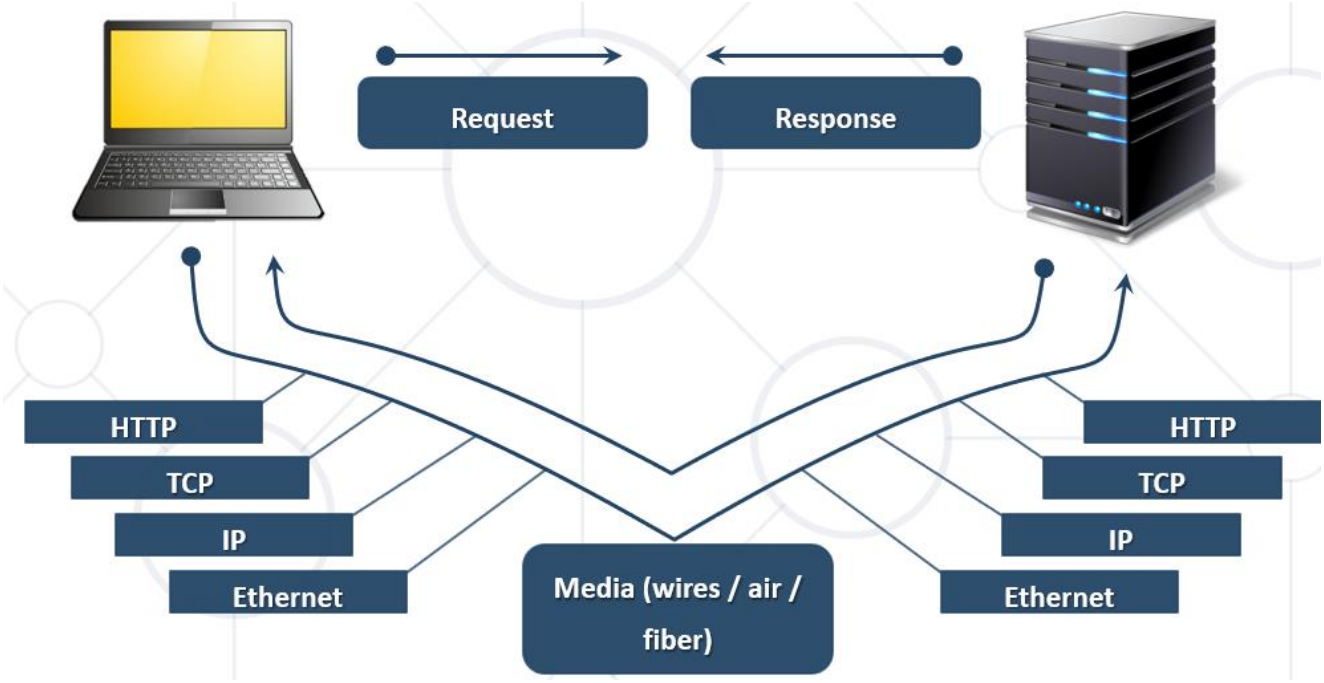
- Statelessness – няма състояние, на практика не е така – има кукита, и .т.н.

https://blog.restcase.com/4-maturity-levels-of-rest-api-design/

If you are building REST (REpresentational State Transfer) APIs or REST (REpresentational State Transfer) Services you're using HTTP. Technically, REST services can be provided over any application layer protocol as long as they conform to certain properties. In practice, basically, everyone uses HTTP Protocol.

2.1. HTTP Basics

Hyper Text Transfer Protocol

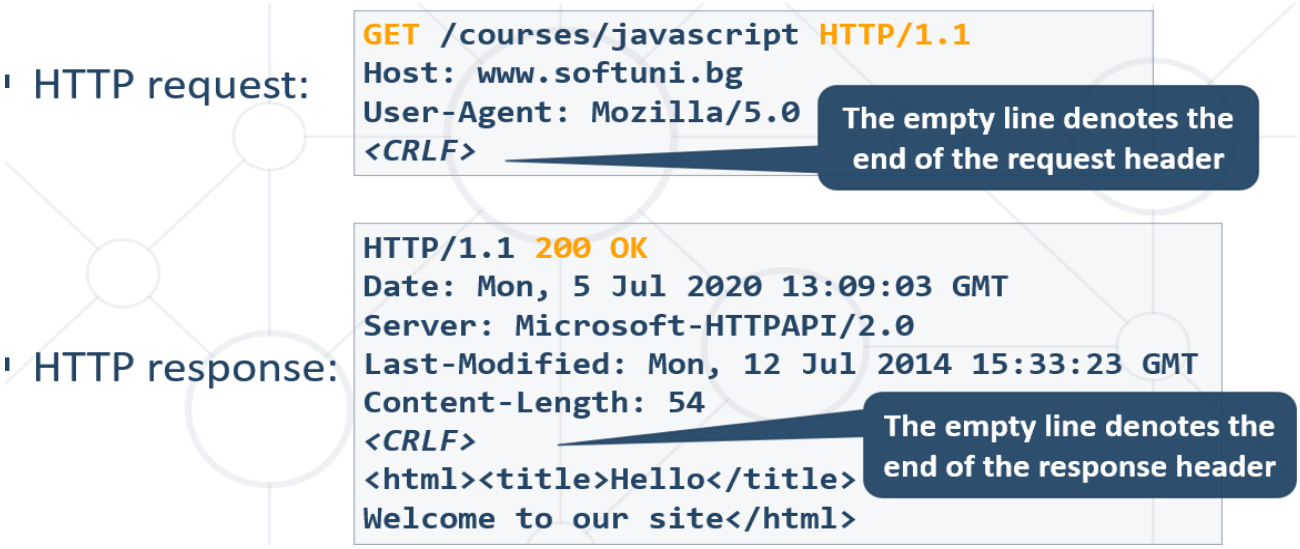


HTTP Request Methods

- **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

A Uniform Resource Identifier (URI) is a unique sequence of characters that identifies a logical or physical resource used by web technologies. URIs may be used to identify anything, including real-world objects, such as people and places, concepts, or information resources such as web pages and books. Some URIs provide a means of locating and retrieving information resources on a network (either on the Internet or on another private network, such as a computer filesystem or an Intranet); these are Uniform Resource Locators (URLs). A URL provides the location of the resource. A URI identifies the resource by name at the specified location or URL. Other URIs provide only a unique name, without a means of locating or retrieving the resource or information about it, these are Uniform Resource Names (URNs).

URL е подмножество на URI.

URN е подмножество на URI също.

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>
** The requested URL /img/logo.gif was not found on this server. <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
Content-Type: application/xml

2.5. HTTP verbs

GET, POST, PUT, PATCH, DELETE

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

Multi-Purpose Internet Mail Extensions

What is MIME?

- **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:

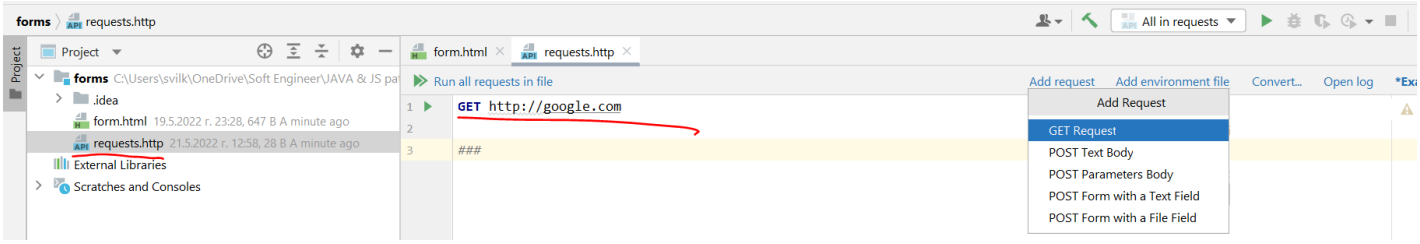
MIME Type / Subtype	Description
application/json	JSON data
image/png	PNG image
image/gif	GIF image
text/html	HTML
text/plain	Text
text/xml	XML
video/mp4	MP4 video

application/pdf	PDF document
multipart/form-data; boundary	

2.7. HTTP Tools

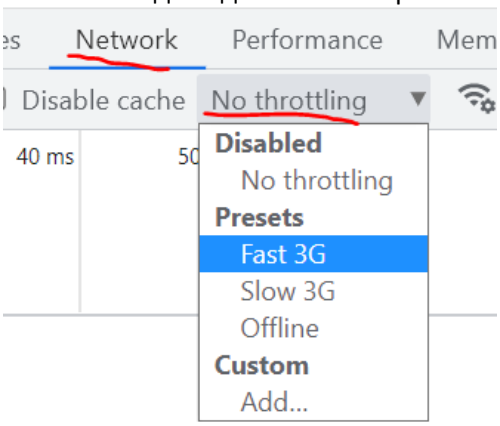
HTTP Tools for Developers:

- **През IntelliJ**



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

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



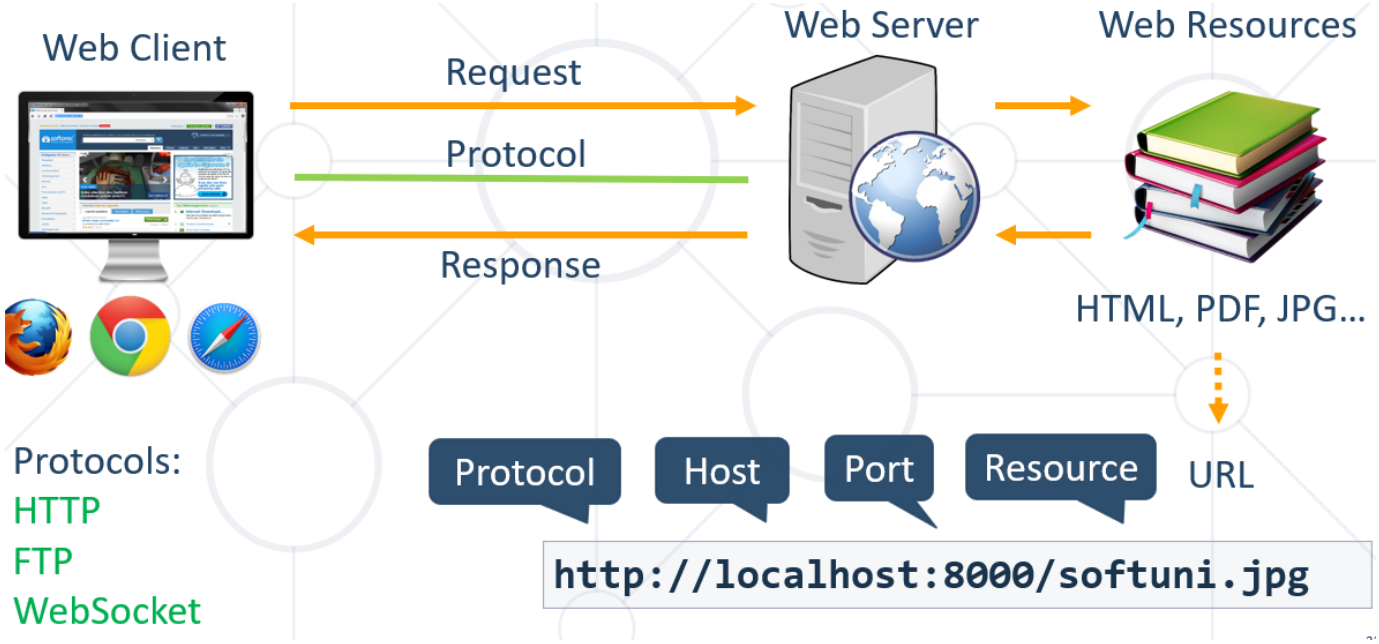
- **Telnet**
- През Ubunto (Linux) и SSL връзка
- Insomnia Rest
- Postman
- RESTClient

2.8. Web Server

What is a Web Server?

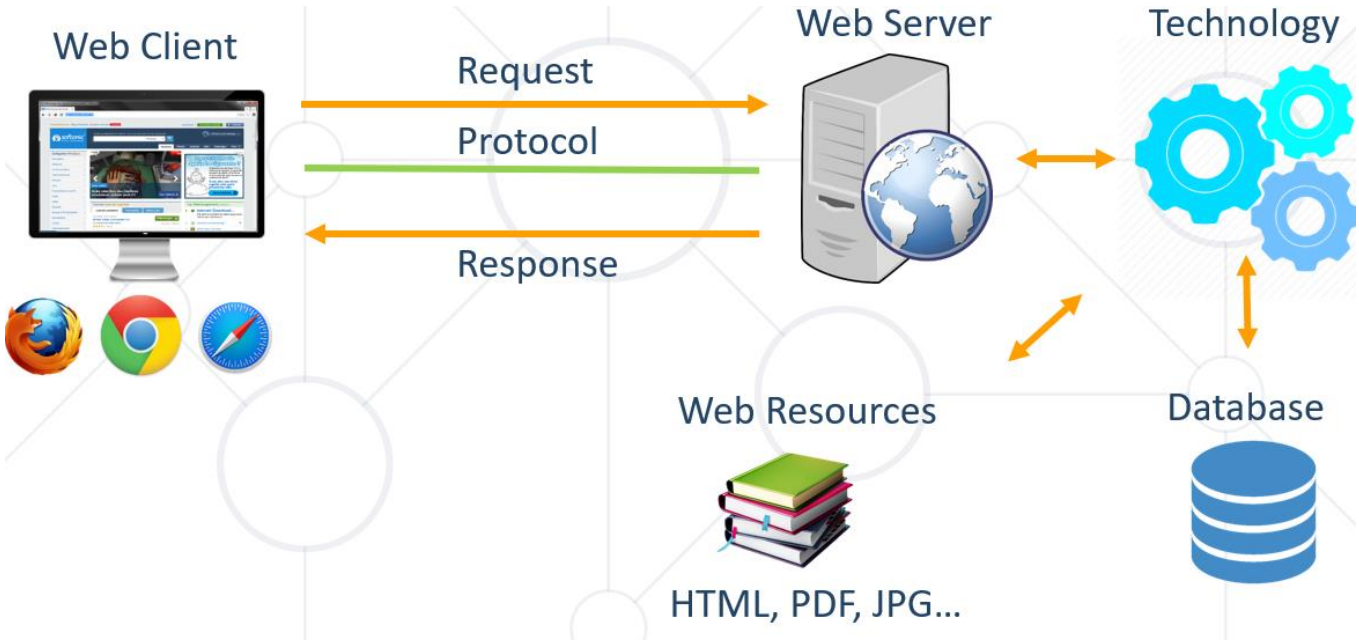
- Computer system that processes requests via **HTTP**, the basic network protocol

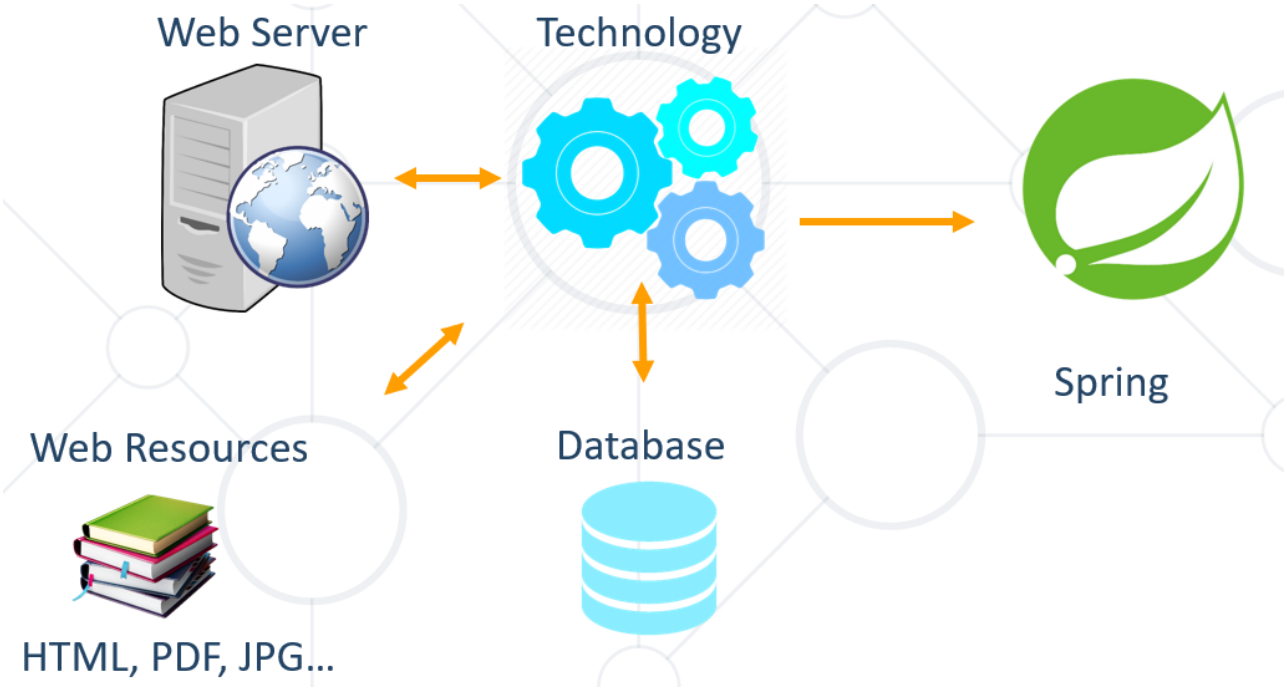
Web Server Work Model



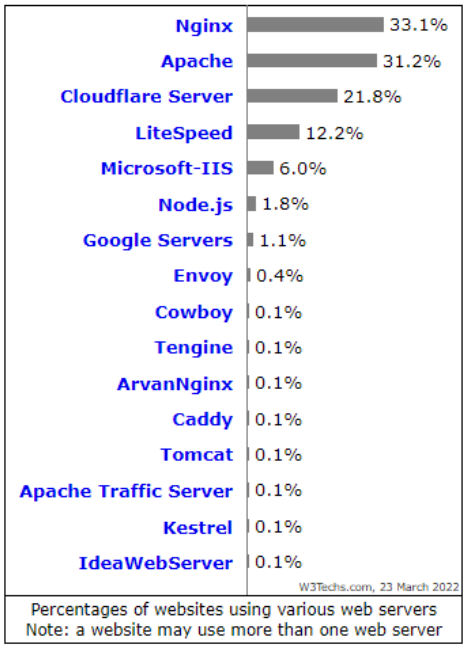
Protocols:
HTTP
FTP
WebSocket


```
http://localhost:8000/softuni.jpg
```





Most Popular Web Servers (W3Techs)



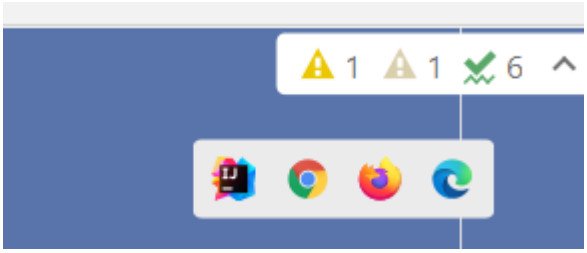
2.9. HTML Forms

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:

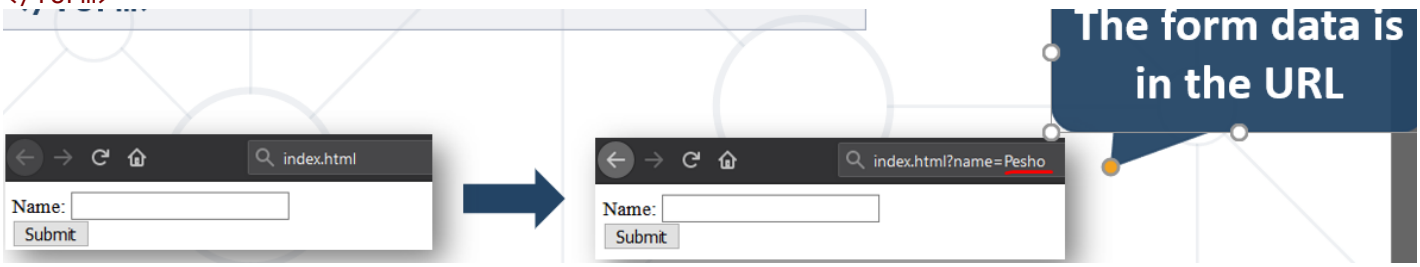
```
<form action="home.html">
  <input type="submit" value="Go to homepage"/>
</form>
```

HTML Forms – Method Attribute

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

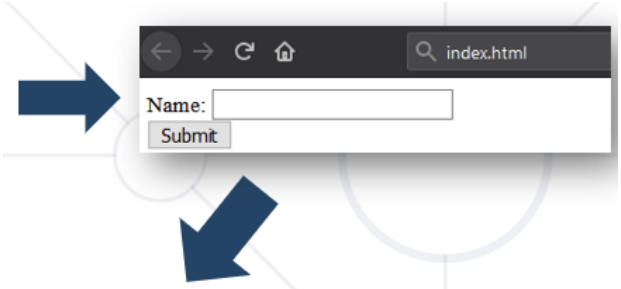
Get

```
<form action="/" method="get">
    Name: <input type="text" name="name">
        <br>
  <input type="submit" value="Submit">
</form>
```



Post

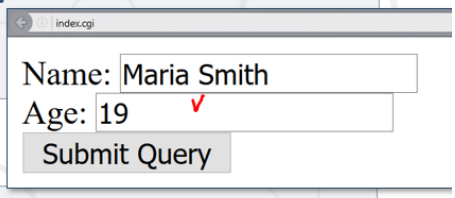
```
<form action="/" method="post">
    Name: <input type="text" name="name">
    <br>
    <input type="submit" value="Submit">
</form>
```



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

```
<form action="/" method="post">
  Name: <input type="text" name="name"/> <br/>
  Age: <input type="text" name="age"/> <br/>
  <input type="submit" />
</form>
```




```
POST http://localhost/cgi-bin/index.cgi HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 23

name=Maria+Smith&age=19
```

File uploads are not supported

2.10. HTTP/2

Old HTTP/1.1

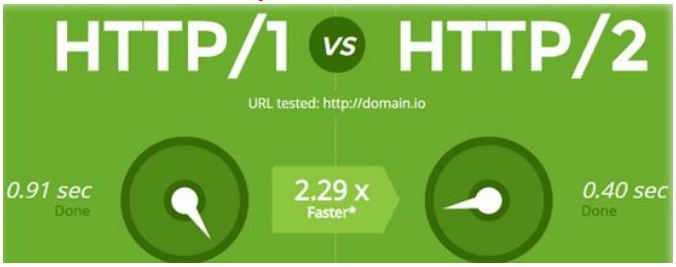
What's HTTP/2

- 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).

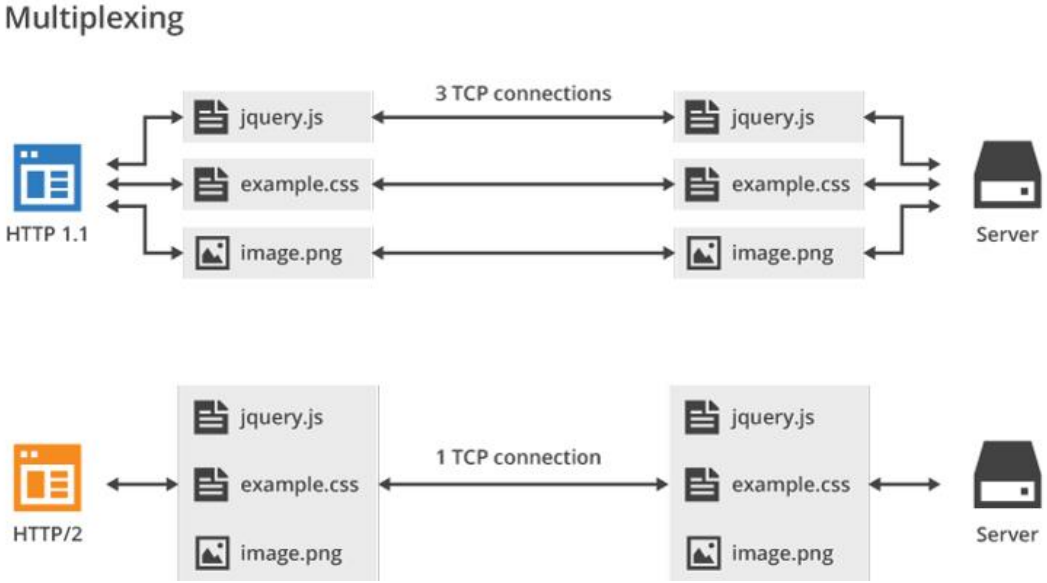
What's New?

- **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**



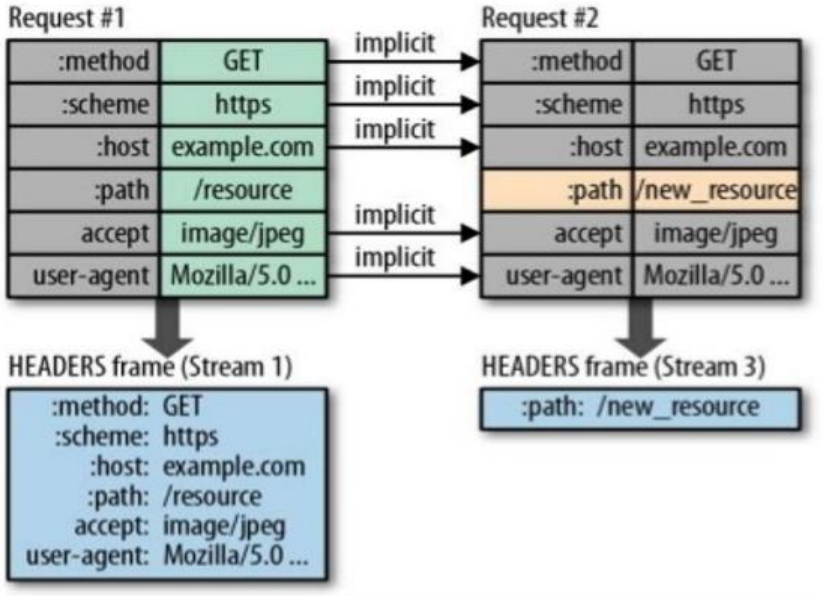
HTTP/2 Multiplexing

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



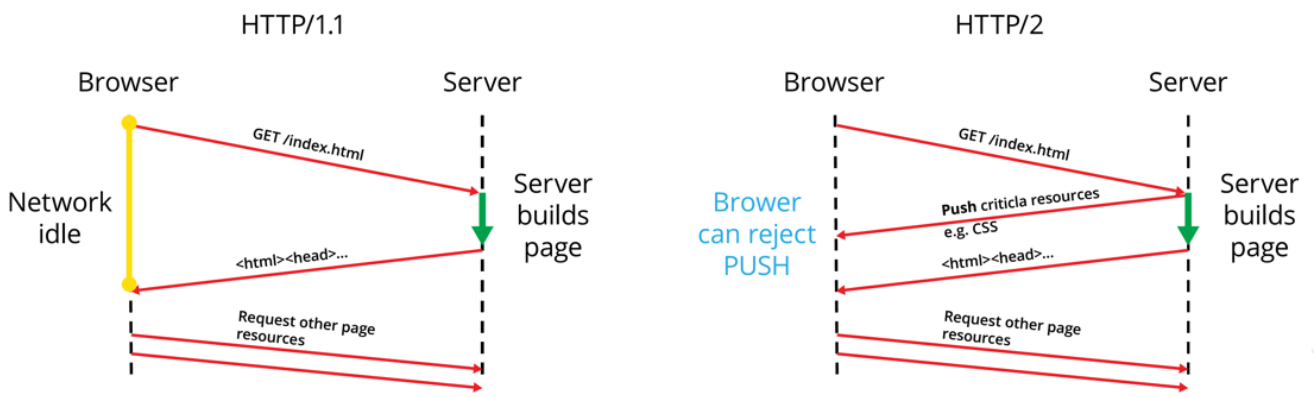
HTTP/2 Header Compression

- **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 3-way handshake

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



2.11. 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.

2.12. 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 хешированието, което е необратим процес, то крипитрането е обратим процес и може да се декриптира.

SHA256 – криптира бързо, и не е добре да криптираме с него пароли. Всяка буква се замества с друга буква. Всеки път резултата от криптирането е един и същ.

`Pbkdf2PasswordEncoder` – една парола се криптира по различни начини
Hash(salt+password) -> salt + hash(salt + password)

BCryptPasswordEncoder

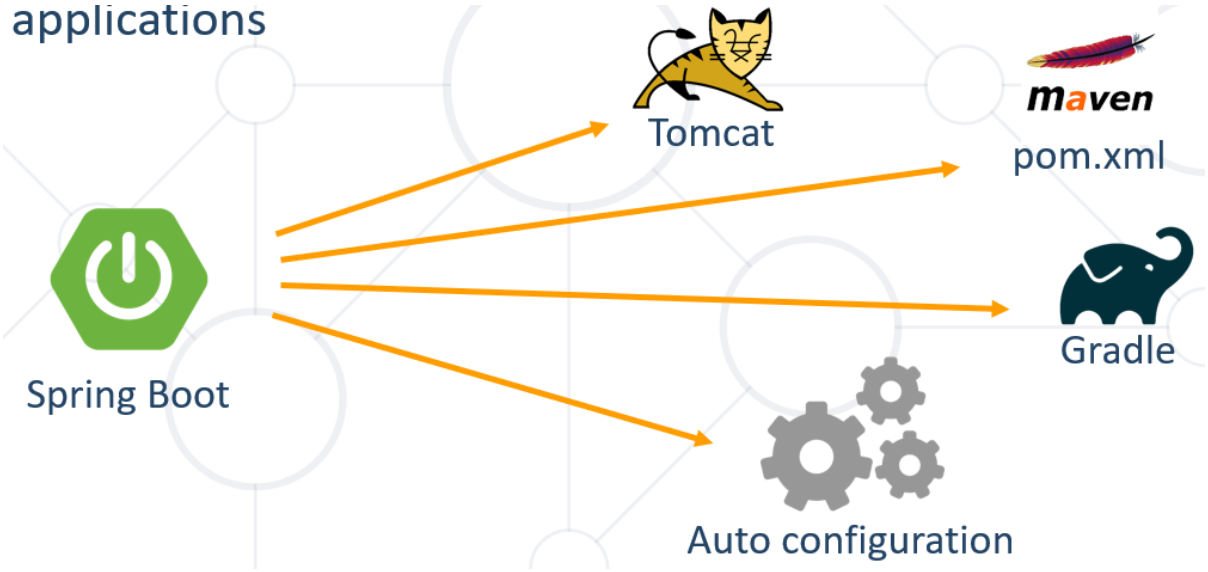
SCryptPasswordEncoder

3. Spring Boot Introduction

3.1. What is Spring Boot?

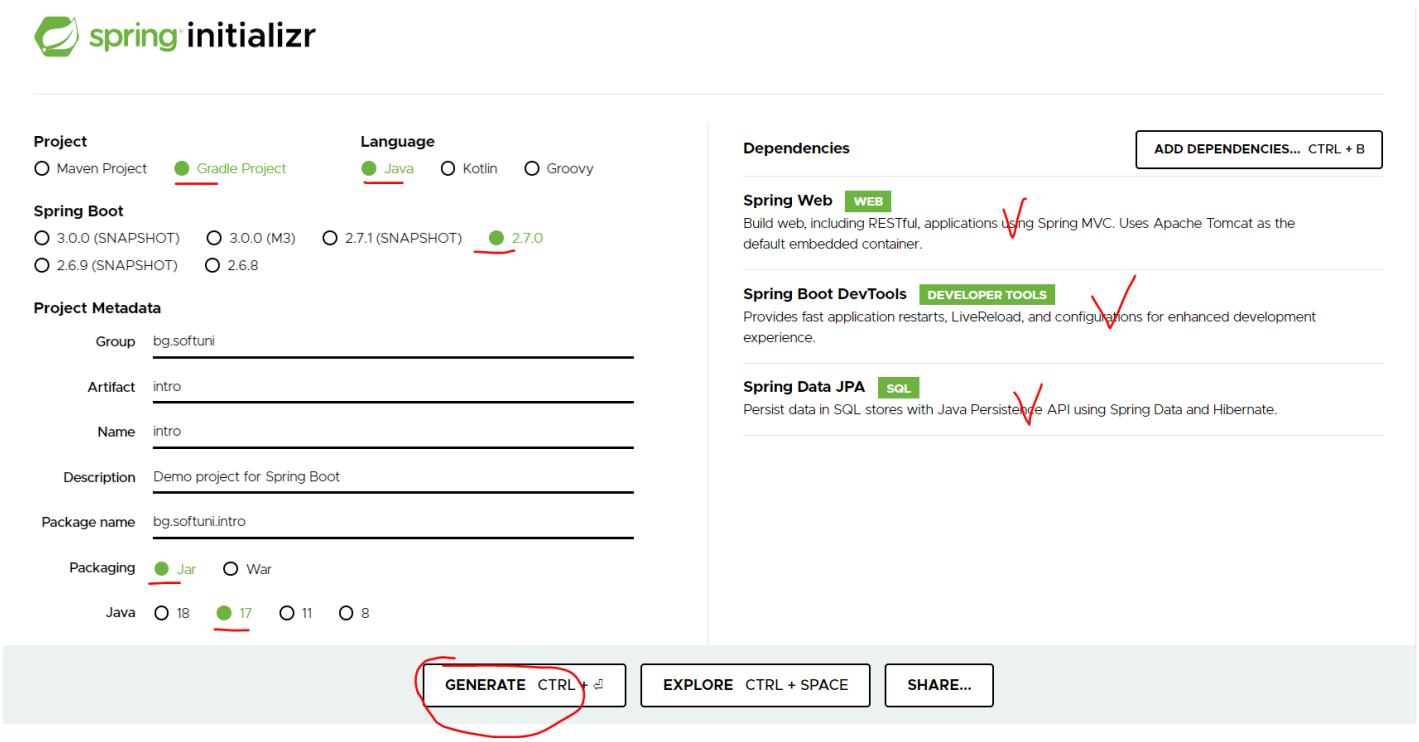
Spring Boot

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



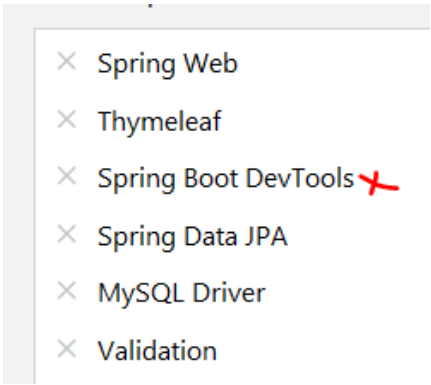
Creating Spring Boot Project

- Just go to **https://start.spring.io/** 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>
```

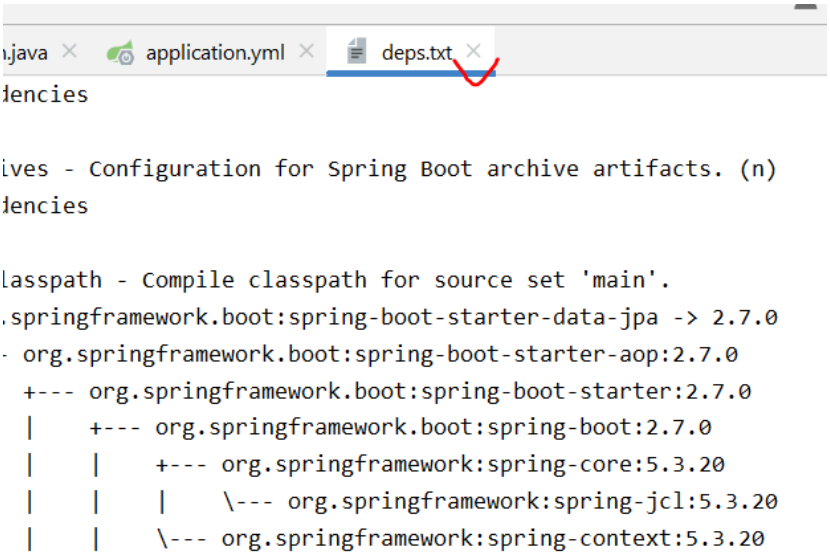
- In **Gradle**:

```
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.security:spring-security-crypto:5.5.2'
       implementation 'org.springframework.boot:spring-boot-starter-validation'
        implementation 'org.modelmapper:modelmapper:3.0.0'


    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') {
```

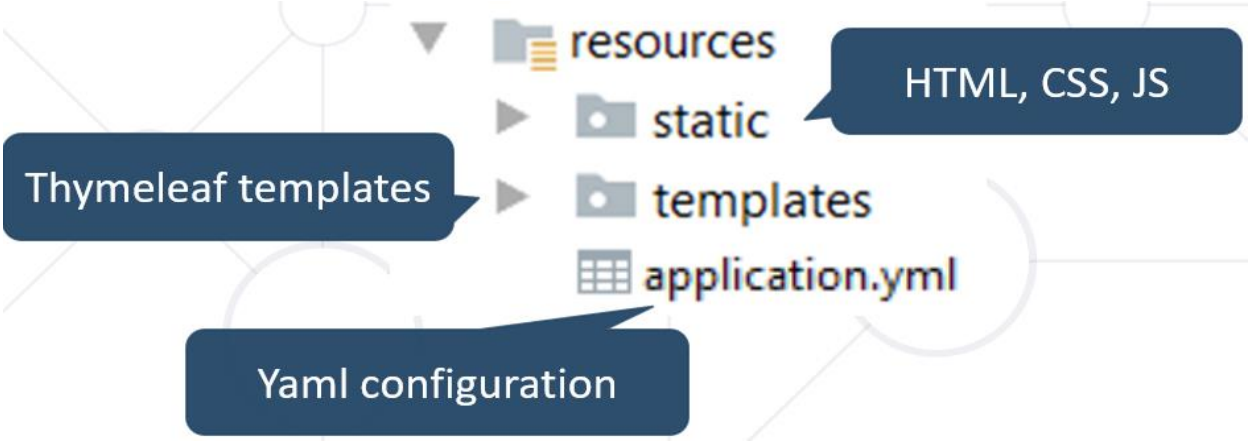
```
    useJUnitPlatform()
}
```

Example of Maven

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>bg.softuni</groupId>
    <artifactId>pathfinder</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>pathfinder</name>
    <description>pathfinder</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-devtools --
>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <version>2.7.0</version>
        </dependency>
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

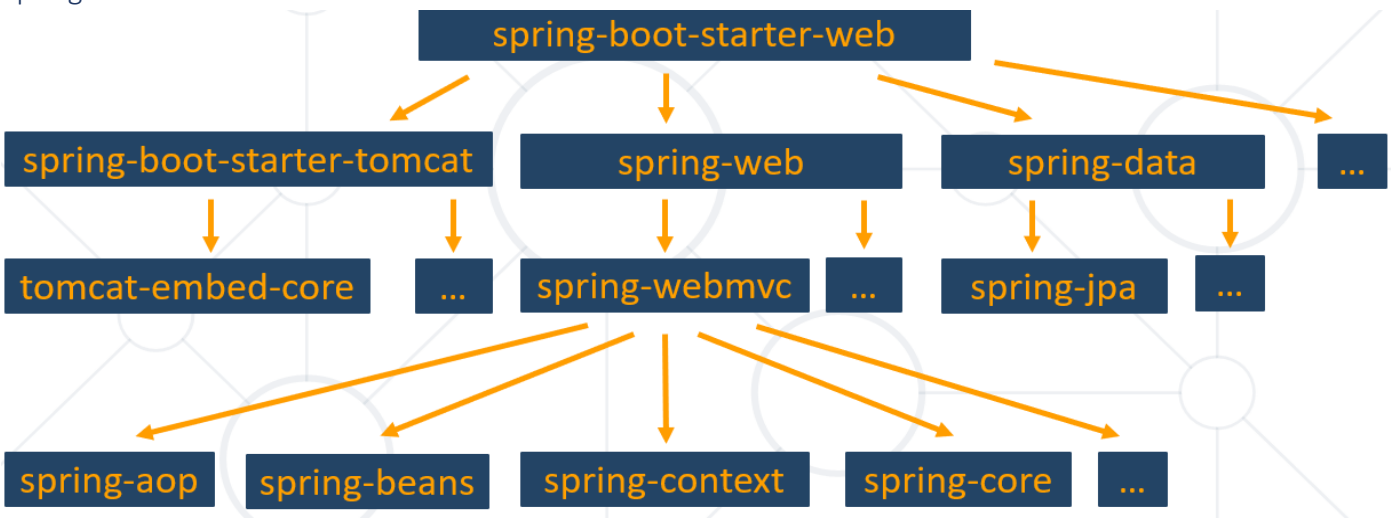
Spring Resources



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

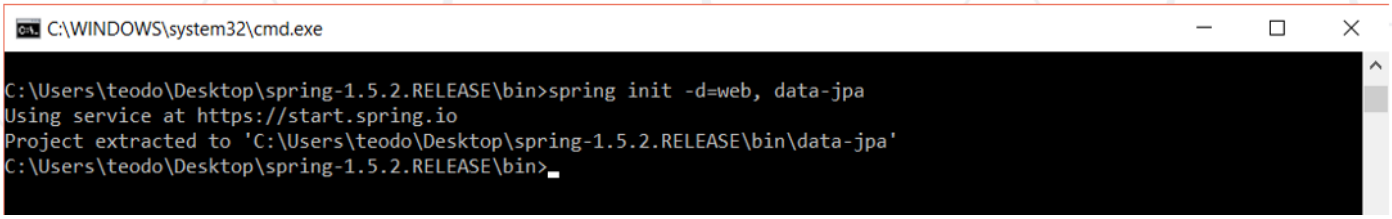
Spring Boot Starters



Spring Boot CLI (Command Line Interface)

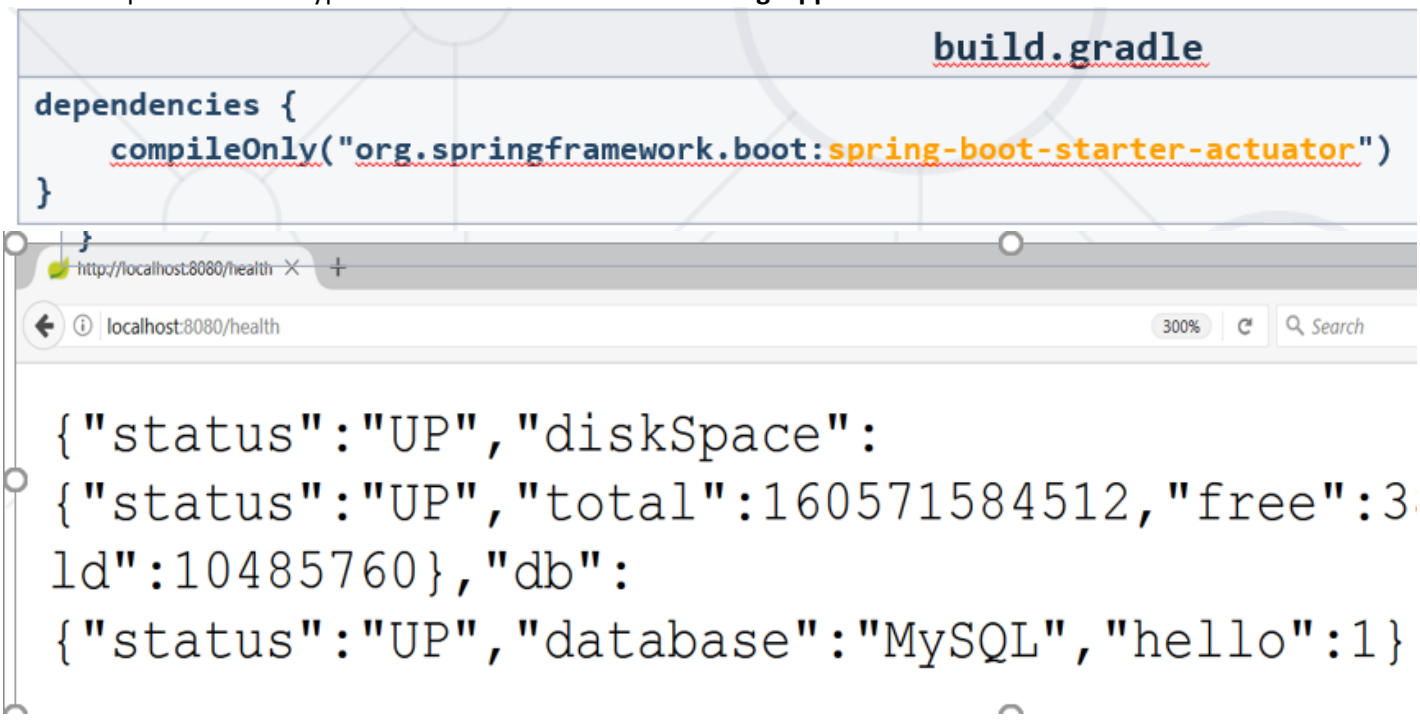
- **Command Line Interface** - Spring Boot software to run and test Spring Boot applications

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**

UserServiceImpl.java

```
//Traditional Way
public class UserServiceImpl implements
UserService {

private UserRepository userRepository = new
UserRepositoryImpl();

}
```

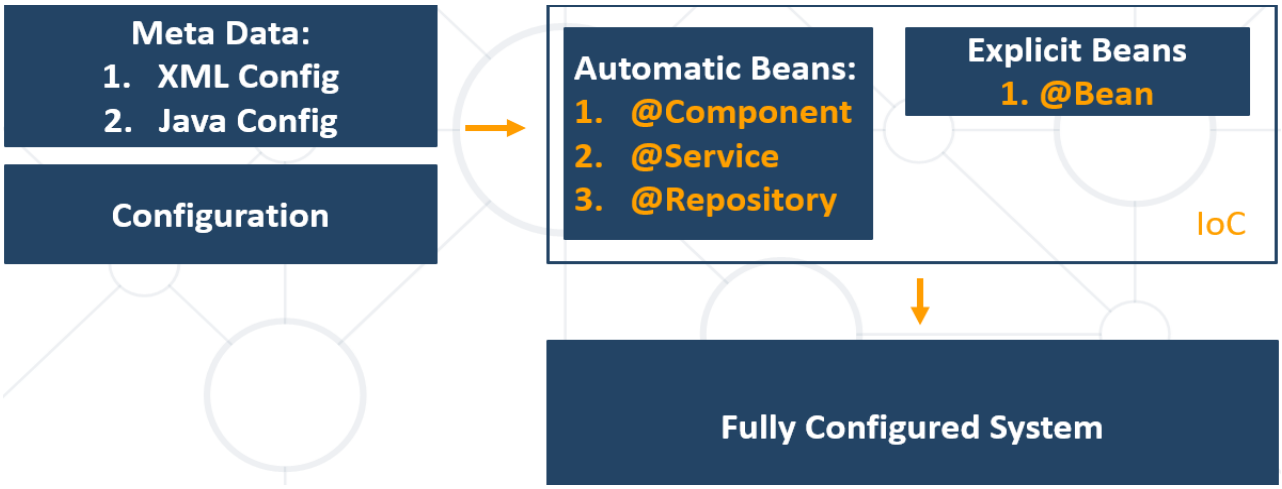
UserServiceImpl.java

```
//Dependency Injection
@Service
public class UserServiceImpl implements
UserService {

@Autowired
private UserRepository userRepository;
}
```

@Autowired – an annotation responsible for injecting a bean by its type

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());
    }
}
```

```
DOG: Dog
```

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-а.

- 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

In web-aware application, 4 more scopes:

- **Request** - в рамките на http request
- **Session** – да пазим неща специфични за даден user
- **application**
- **websocket**

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

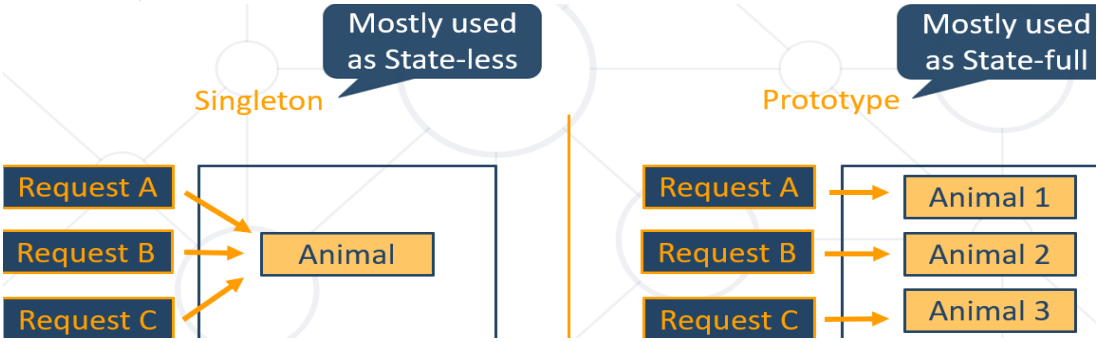
- Will return a different instance every time it is requested from the container

```
@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



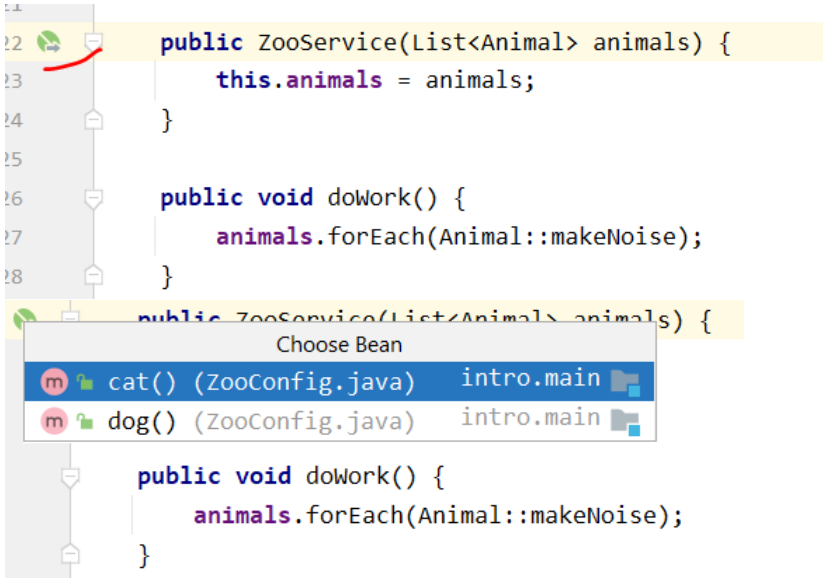
Demo with Beans

IntelliJ средата ни дава препратка към двата bean-а

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();
    }
}
```

```java
@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

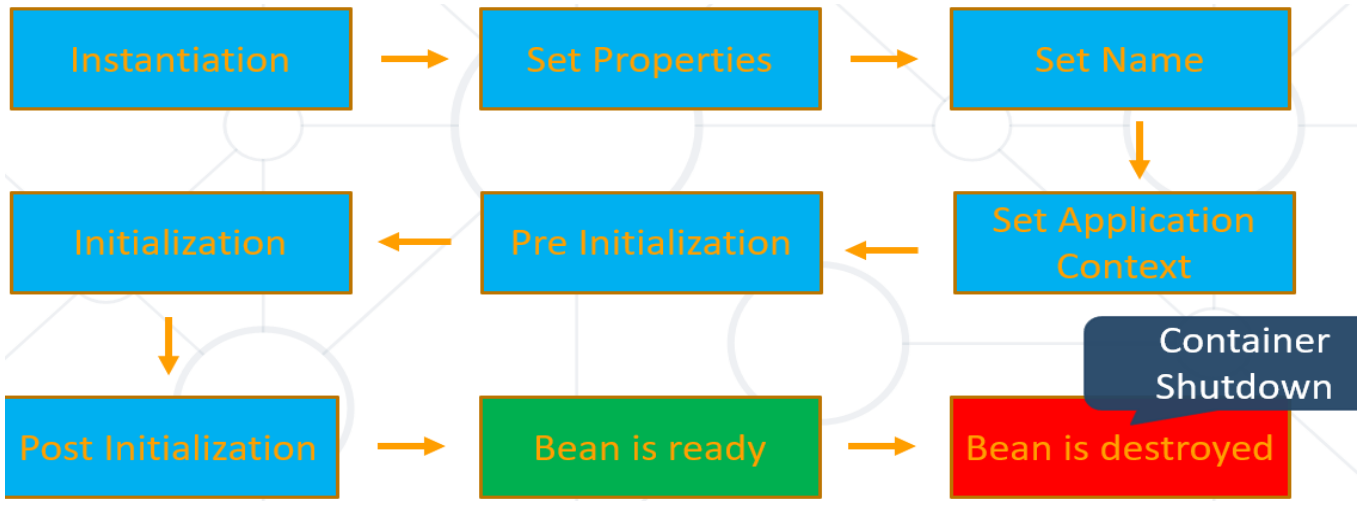
```java
public class Dog implements Animal{
    private final boolean superDog;

    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);
    }
}
```

```java
@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

```java
@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 – идва от package javax.annotation;) вместо BeanNameAware, BeanFactoryAware или InitializingBean (които са изцяло Spring).

```
Най-често използваме по този начин – или през @Component или през @Configuration
@Component
public class AppInit {
    private final UserService userService;
    private final RimService rimService;

    public AppInit(UserService userService, RimService rimService) {
        this.userService = userService;
        this.rimService = rimService;
    }

    @PostConstruct
    public void beginInit(){
        userService.init();
        rimService.init();
```

```
    }
}
```

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 части Spring).

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/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


spring.mvc.hiddenmethod.filter.enabled=true   //ще ни позволи да използваме delete
```

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&createDatabase
IfNotExist=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   never     embedded/only for in-memory database/
  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     //първо се създават таблиците с връзките, и чак
след това се изпълнява SQL скрипта за наливане на данните

        show_sql: true
```

```
#Cloudinary Properties
#cloudinary:
  #api-key:
  #api-secret:
  #cloud-name:


logging:
  level:
    org:
      hibernate:
        sql: debug
        type:
          descriptor:
            sql: trace
```

Convert YAML and Properties file

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

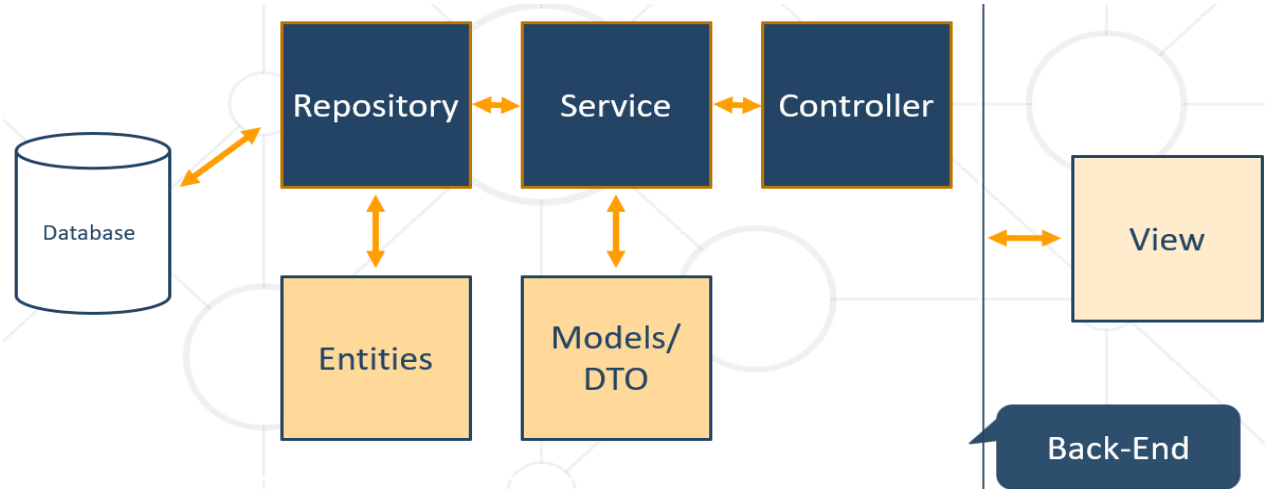
DDL-AUTO

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.

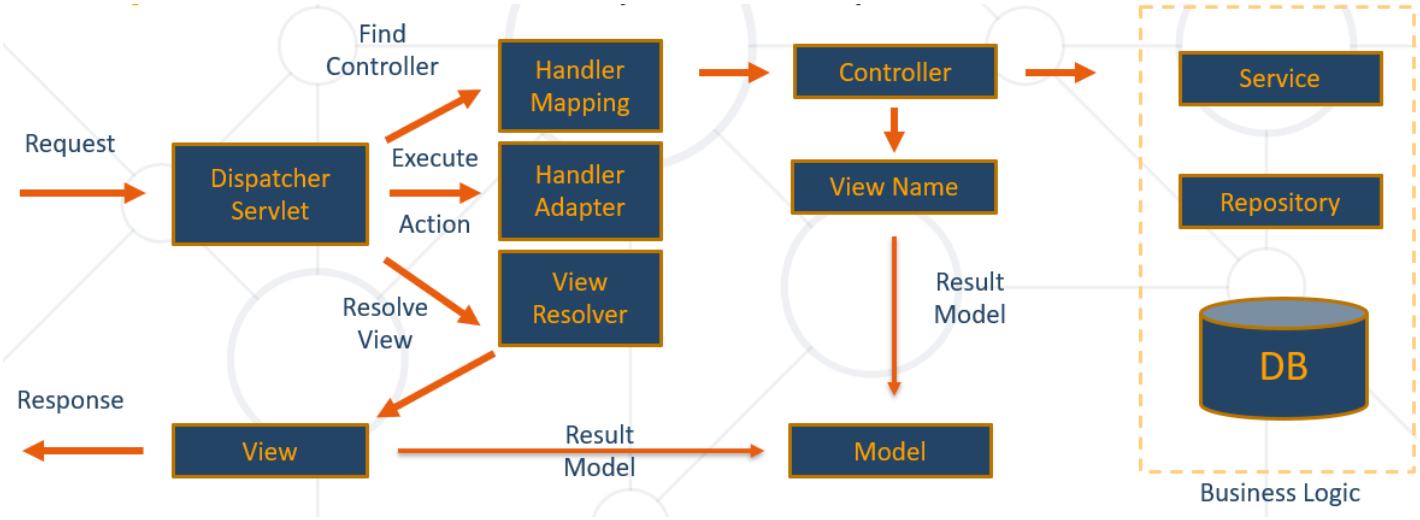
```
@Service
public class CatServiceImpl implements CatService {
    private final CatRepository catRepository;

    @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

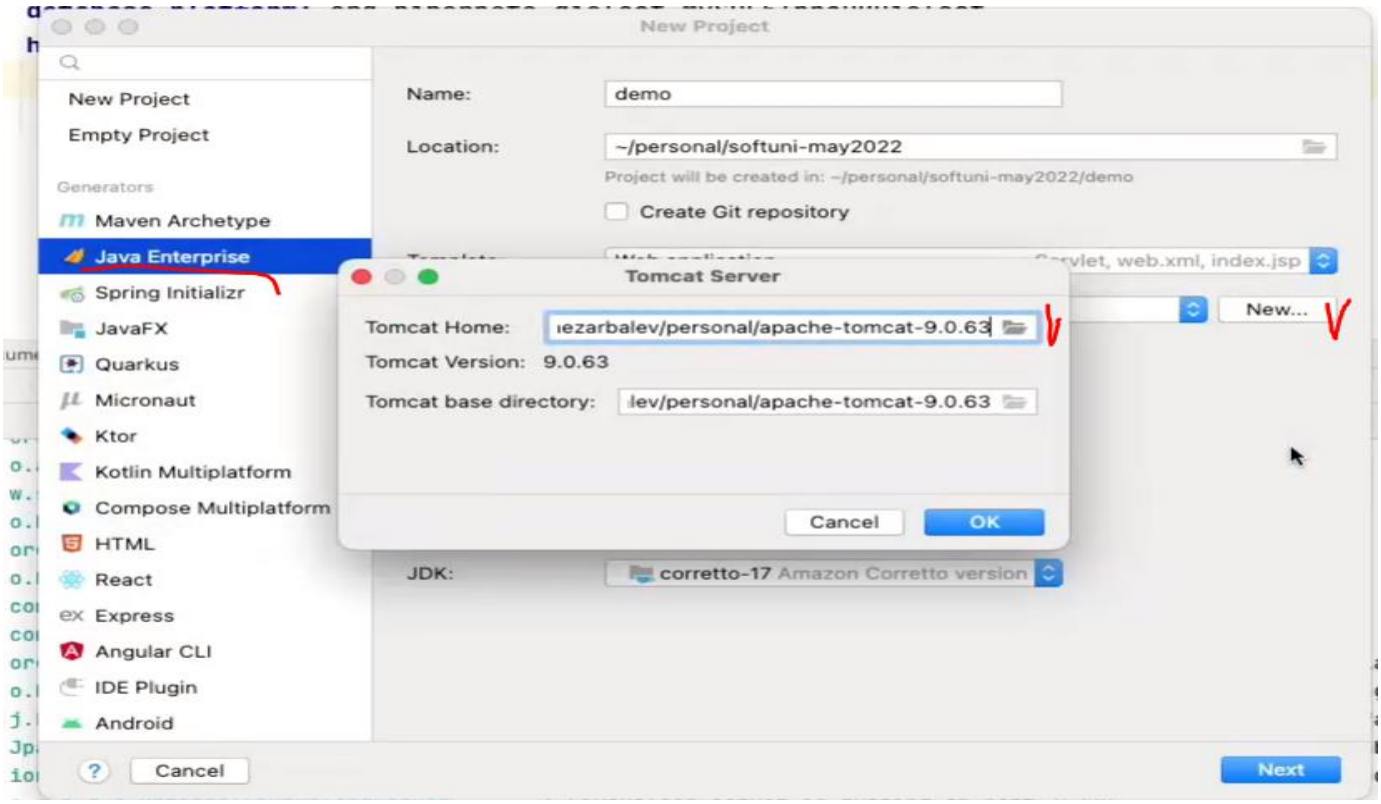
A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.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

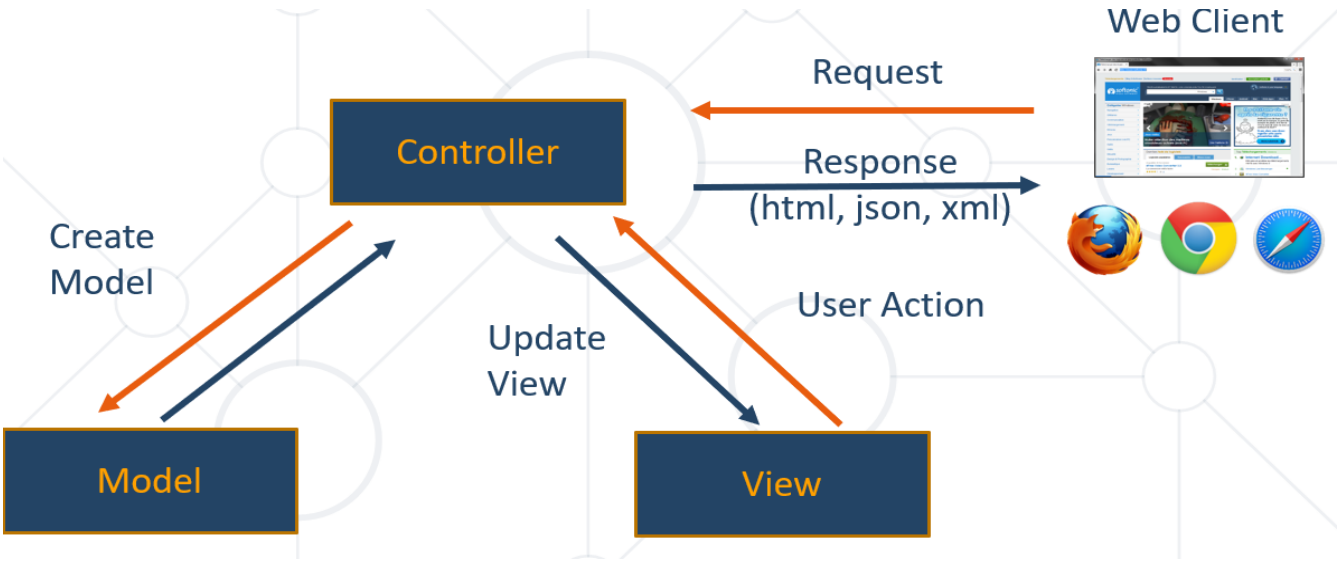
1) Unzip Apache Tomcat
2) Create new project



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");
response.setStatus(HttpStatus.NotFound)

        // 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

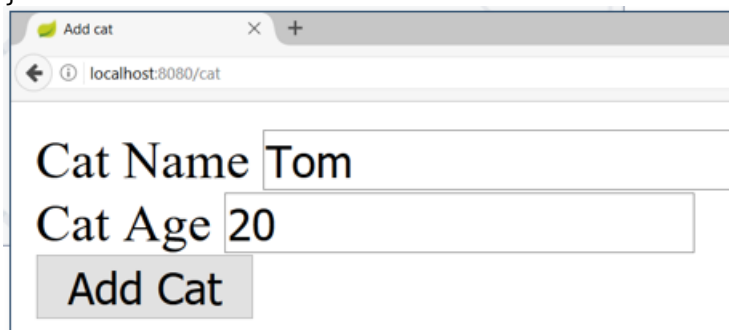
- Similar to the **GetMapping** there is also an alias for **RequestMapping** with method POST

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

- Similar annotations exist for all other types of request methods

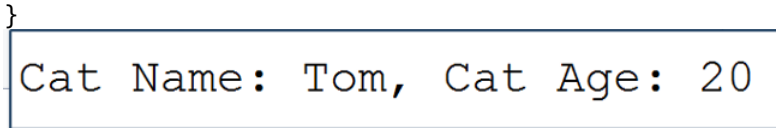
Actions – Post Requests – анотираме целият контролер

```
@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"; //редиректваме към този URL, не към html страница
    }
}
```



Dto вземане на данни от html формата към UserDto

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

Пример 1

Вариант 1 за първоначални стойности

```
@GetMapping("/users/register")
public String register(Model model){
    if (!model.containsAttribute("userRegistrationDto")) {
        model.addAttribute("userRegistrationDto", new UserRegistrationDto());
    }

    return "register";
}
```

Вариант 2 за първоначални стойности

```
    @ModelAttribute
    public UserRegistrationDto initRegistrationDto(){
        return new UserRegistrationDto();
    }

    @GetMapping("/users/register")
    public String register(Model model){
//        if (!model.containsAttribute("userRegistrationDto")) {
//            model.addAttribute("userRegistrationDto", new UserRegistrationDto());
//        }

        return "register";
    }
```

register.html файла

```
<form
        th:action="@{/users/register}"
        th:method="POST"
        th:object="${userRegistrationDto}"
      class=" text-center text-light">

    <div class="form-group row">
        <label for="username" class="col-sm-2 col-form-label">Username</label>
        <div class="col-sm-10">
            <input type="text"
                   class="form-control"
                   th:field="*{username}"
                   th:errorclass="is-invalid"
                   id="username"
                   aria-describedby="usernameHelpInline" placeholder="Username">
            <small id="usernameHelpInline"
                   class="invalid-feedback bg-danger text-light rounded">
                Username length must be between 5 and 20 characters.
            </small>
        </div>
    </div>
```

```
<form
        th:action="@{/home}"
        th:method="POST"
        th:object="${fireModel}"
        class="row mb-2">
    <div class="col-md-6">
        <div class="card flex-md-row bg-blur mb-4 box-shadow h-md-250">
            <div class="card-body d-flex flex-column align-items-start">
                <strong class="d-inline-block mb-2 text-primary">Attacker</strong>
                <span style="color: yellow" th:text="${@loggedUser.getFullName()}"></span>

                <h3 class="mb-0 text-white">
                    Select the attacker
                </h3>
                <div class="mb-1 text-white">Select one of the ships that are owned by the current
                <select
                        th:field="*{attackerShipId}"
                        class="form-select mt-5" aria-label="Default select example">
                    <option value="" selected>Select one of the ships that are owned by the currer
                    </option>
                    <option
                            th:each="ship : ${ownShips}"
                            th:value="${ship.id}"
                            th:text="|${ship.name} -- ${ship.health} -- ${ship.power}|"
                            th:selected="*{attackerShipId} == ${ship.id}"
                    />
```

Пример 2 – Plain taking info from the form

```
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*

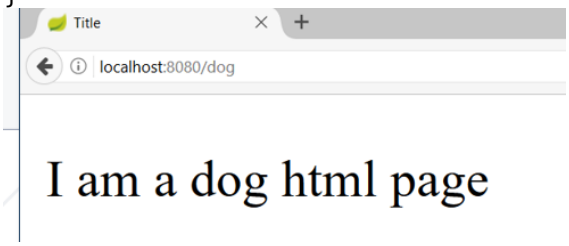
```
@GetMapping("/")
public ModelAndView welcome(ModelAndView modelAndView) {
    modelAndView.addObject("name", "Pesho");
```

```
    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 Parameters – query, form data and parts in multipart requests

In Spring MVC, "request parameters" map to **query parameters**, **form data**, and **parts in multipart requests**. This is because the Servlet API combines query parameters and form data into a single map called "parameters", and that includes automatic parsing of the request body.
In Spring WebFlux, "request parameters" map to query parameters only. To work with all 3, query, form data, and multipart data, you can use data binding to a command object annotated with ModelAttribute.

Request Parameters – query case example

- Getting a parameter from the query string

В нашето URL ще излиза така: localhost:8080/details?id=5
Или в Thymeleaf го пишем по този начин

```
<a class="ml-3 text-danger" th:href="@{/products/buy(id = *{id})}">Buy</a>
```

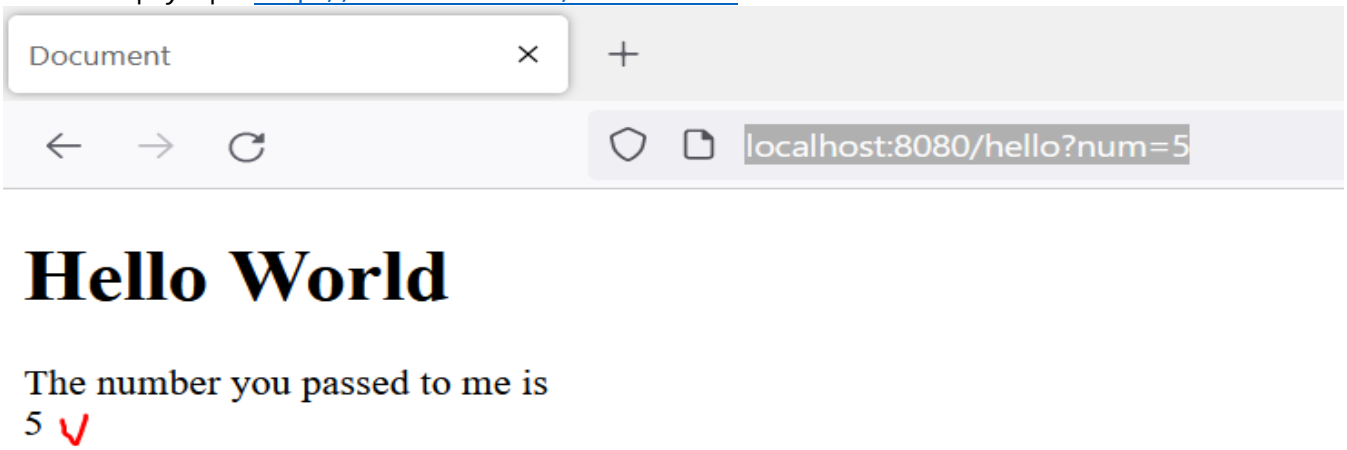
```
@GetMapping("/details")
public String details(@RequestParam("id") Long id) {
...
}
```

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 Parameters – query case 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

В Thymeleaf го пишем така:

```
<a class="ml-3 text-danger" th:href="@{/products/buy/{id}(id = *{id})}">Buy</a>
```

- Getting a parameter from the ***path*** part of the URL – в пътя на URL-то

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

```
@GetMapping("/offers/{id}/details")
public String showOfferDetail(@PathVariable String id){    //Може и без “id”
    return "details";
}
```

```
@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";  //редиректваме към този URL, не към html страница
}
```

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  //редиректваме към този URL, не към html
страница
}
```

Redirecting with Attributes

- Keeping objects after redirect

```
@PostMapping("/register")
public String register(@ModelAttribute UserDTO userDto,
```

```
                        RedirectAttributes redirectAttributes) {
...
    redirectAttributes.addFlashAttribute("userDto", userDto);
    return "redirect:/register";  //редиректваме към този URL, не към html страница
}
```

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;
}
```

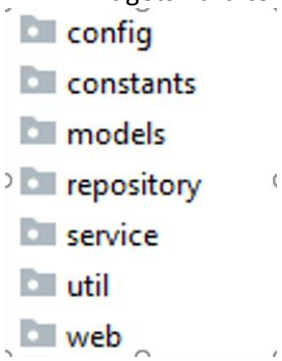
Setter Injection

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

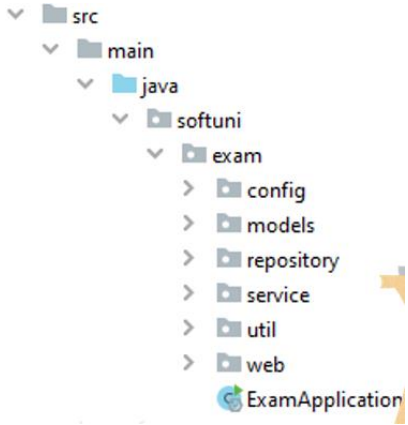
```
@Service
public class HomeContoller(){
    //...
    @Autowired
    public void setServiceA(ServiceA serviceA) {
        this.serviceA = serviceA;
    }
}
```

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

```
Set-Cookie: SSID=Ap4P…GTEq; Domain=foo.com; Path=/;
Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
```

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

```
Set-Cookie: SSID=Ap4P…GTEq; Domain=foo.com; Path=/;
Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
```

Security

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

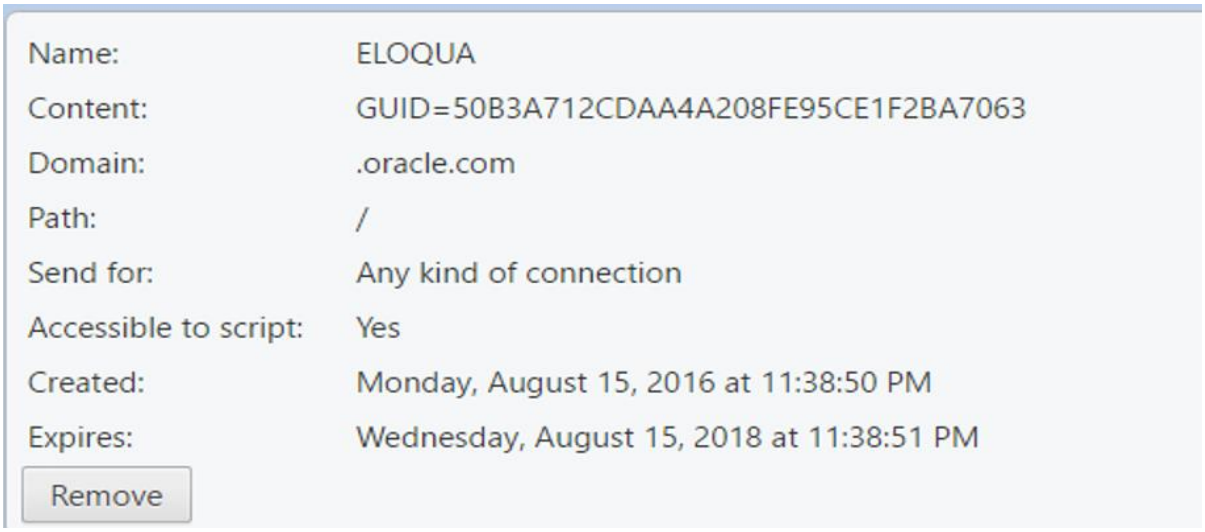
```
Set-Cookie: SSID=Ap4P…GTEq; Domain=foo.com; Path=/;
Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
```

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 1

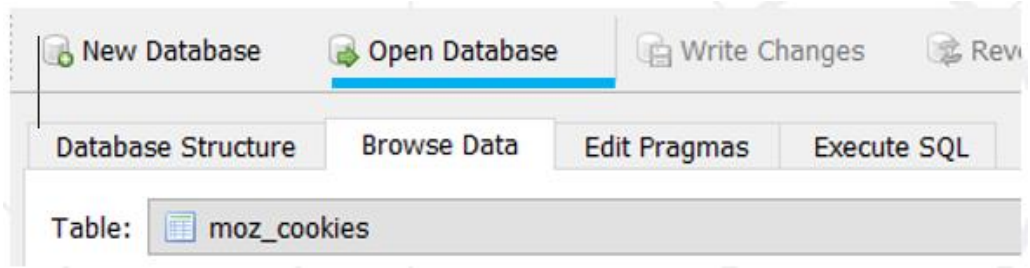
- Most cookies are stored in a **RDBMS**, usually **SQLite**
- Download DB Browser for SQLLite (the **SQLite browser)** from **here - https://sqlitebrowser.org**
- Location of Mozilla cookies

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

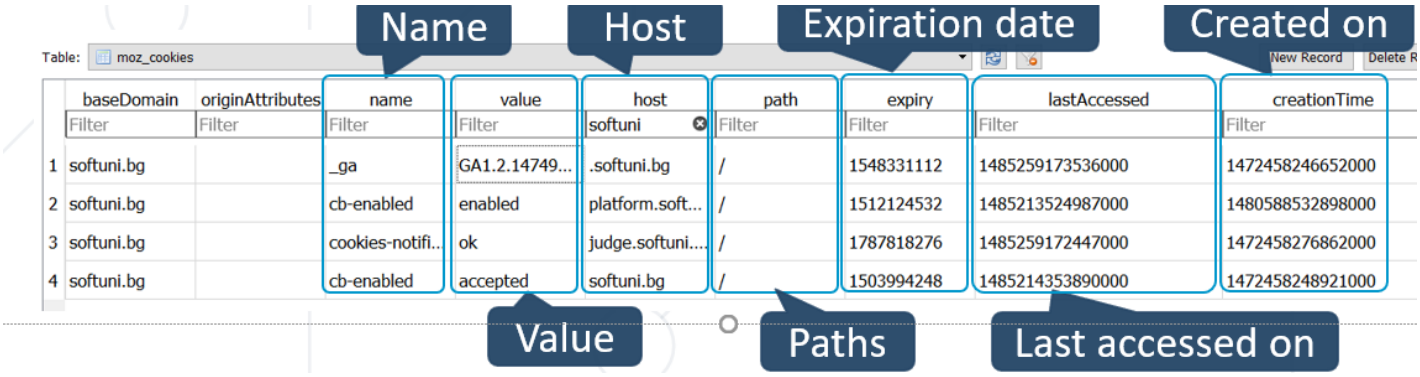
- Location of Chrome cookies

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

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



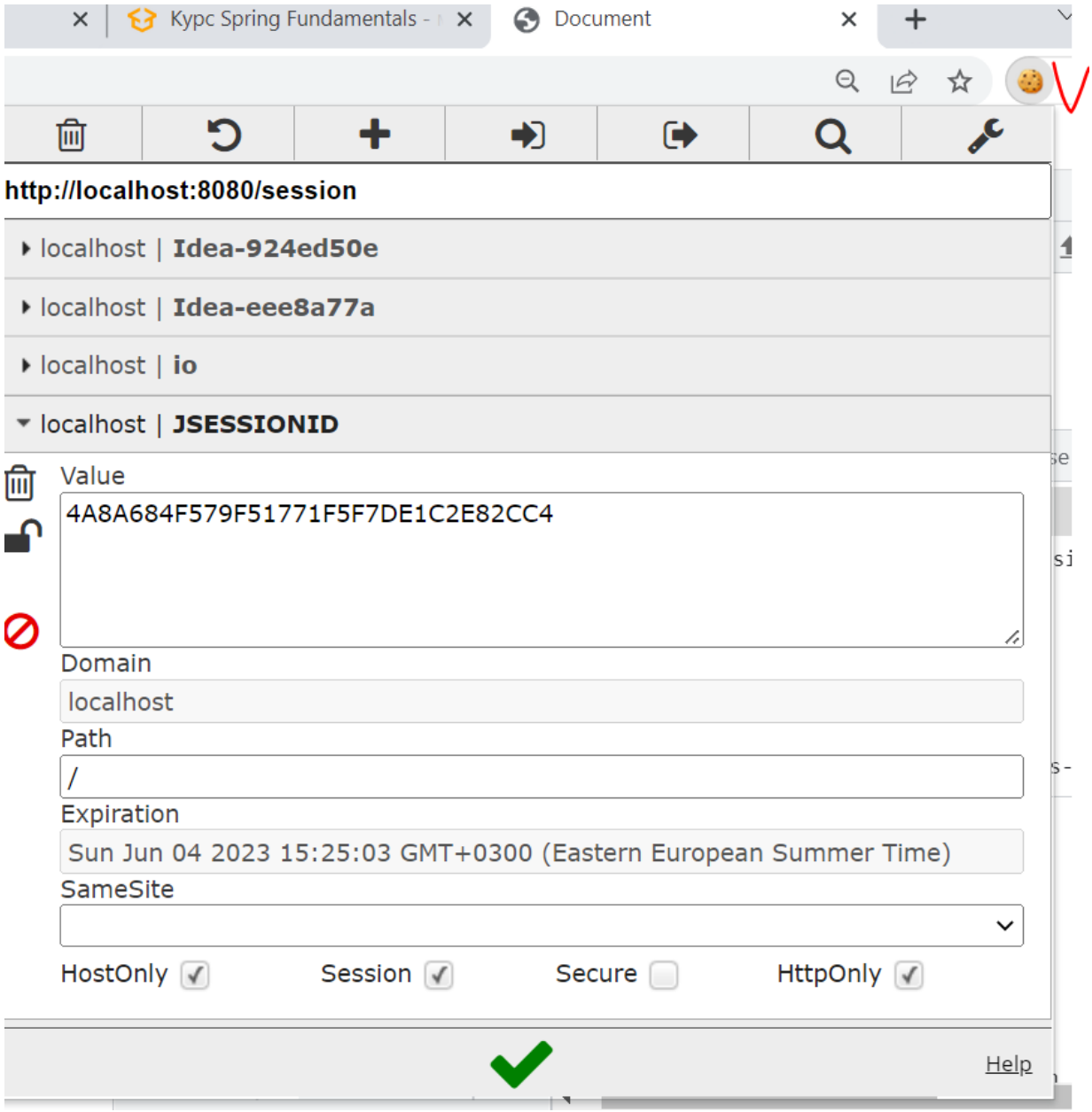
- Browse the cookies table



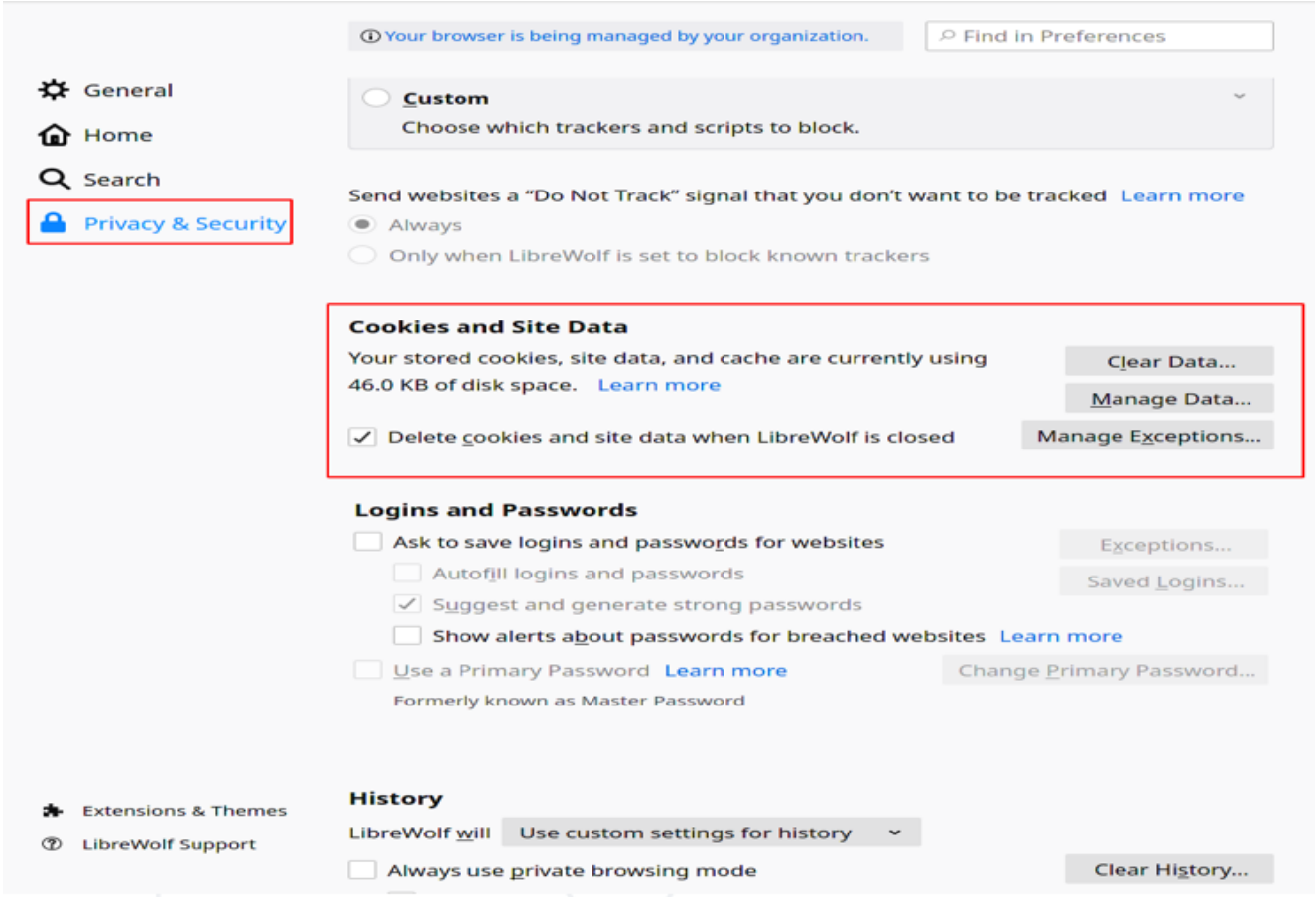
Examine Your Cookies 2

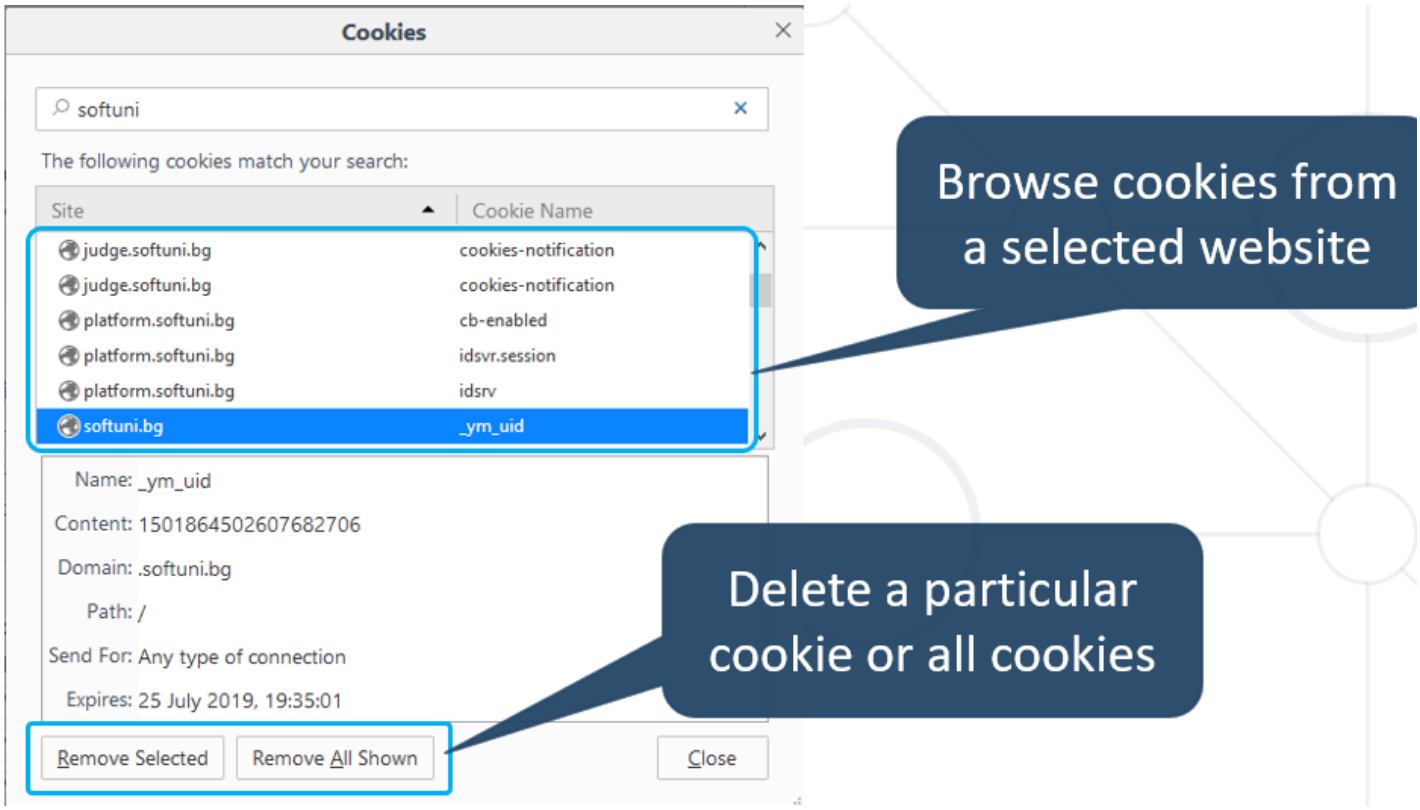
Using the EditThisCookie

https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceomclgfbg?hl=en

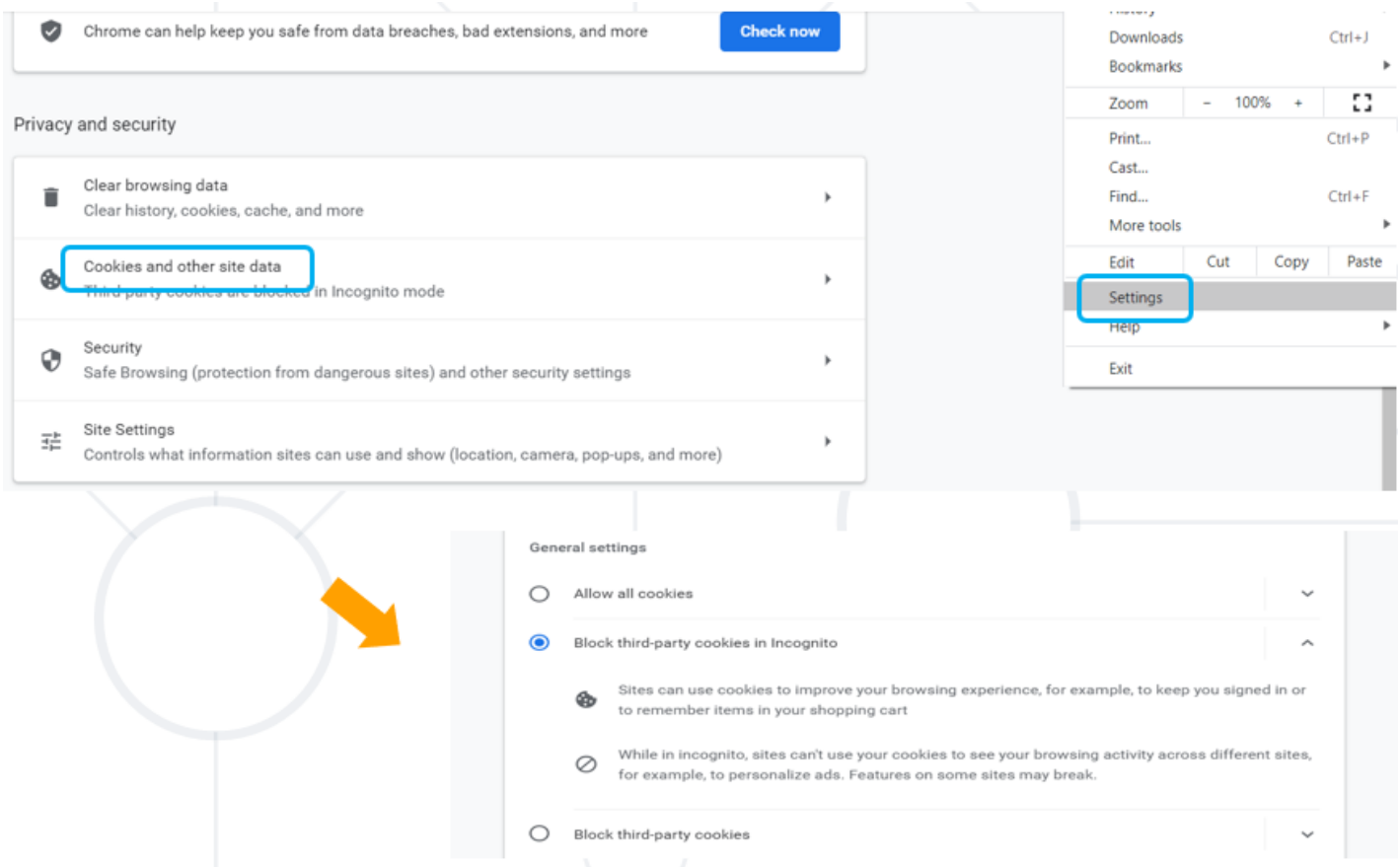


Control Your Cookies – Firefox Browser

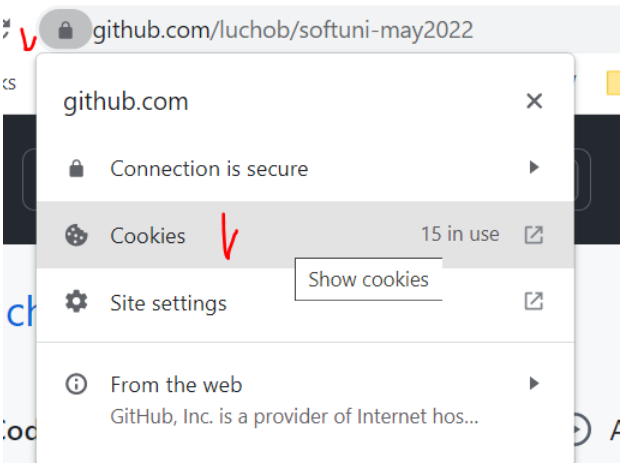


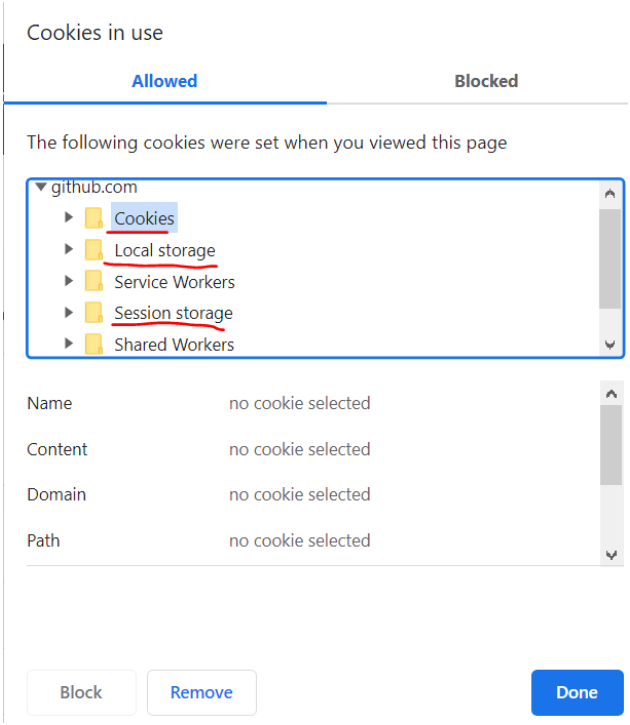


Control Your Cookies – Chrome Browser



Или оттук:





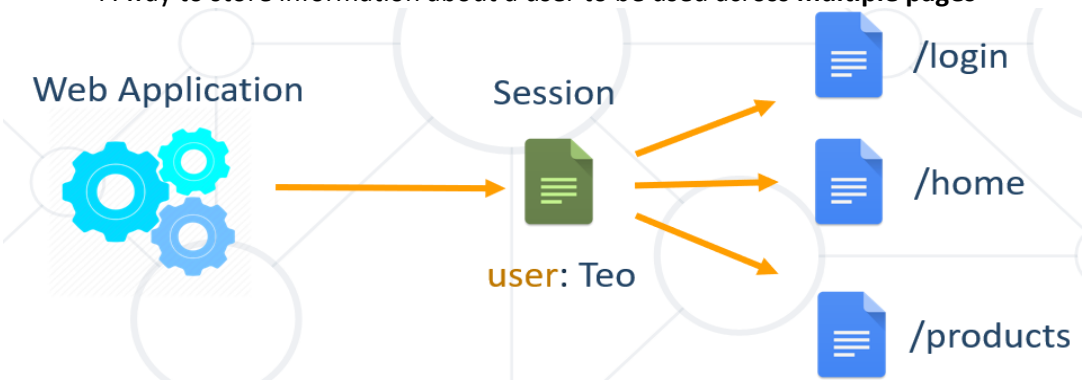
Third Party Cookies

- 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

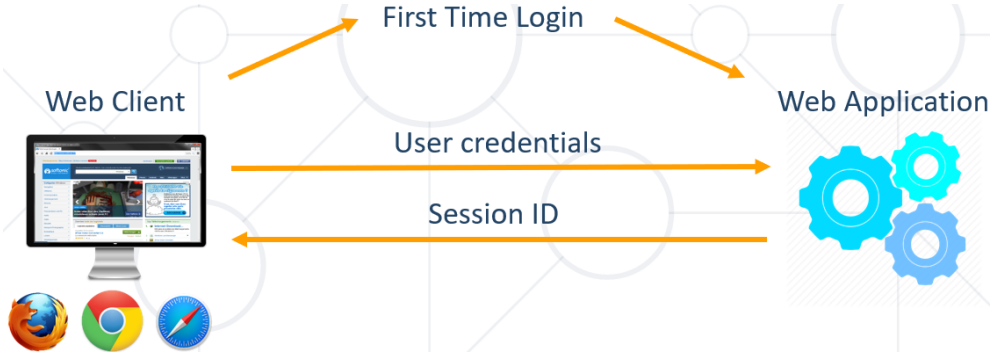
What Are Sessions?

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

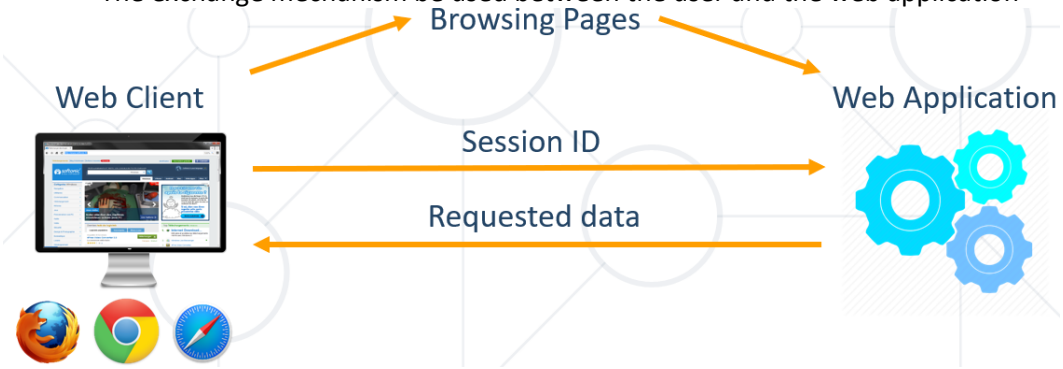


Session Management

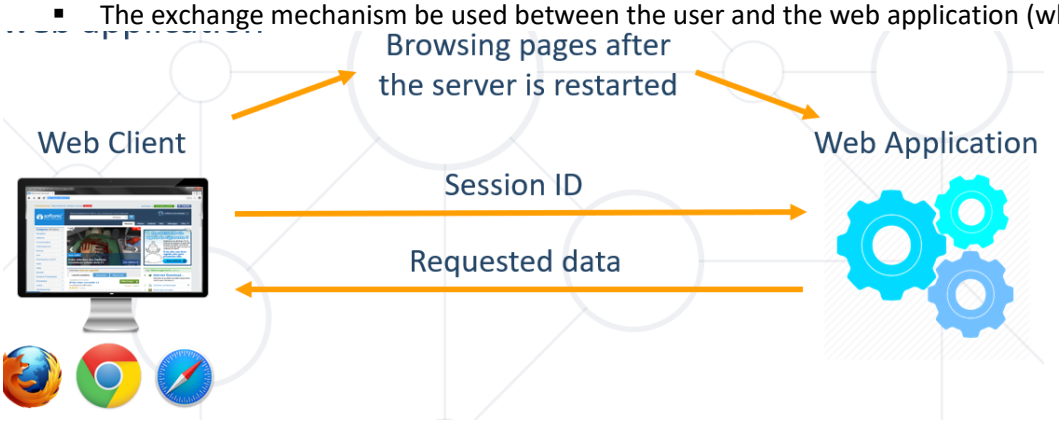
- The exchange mechanism be used between the user and the web application



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

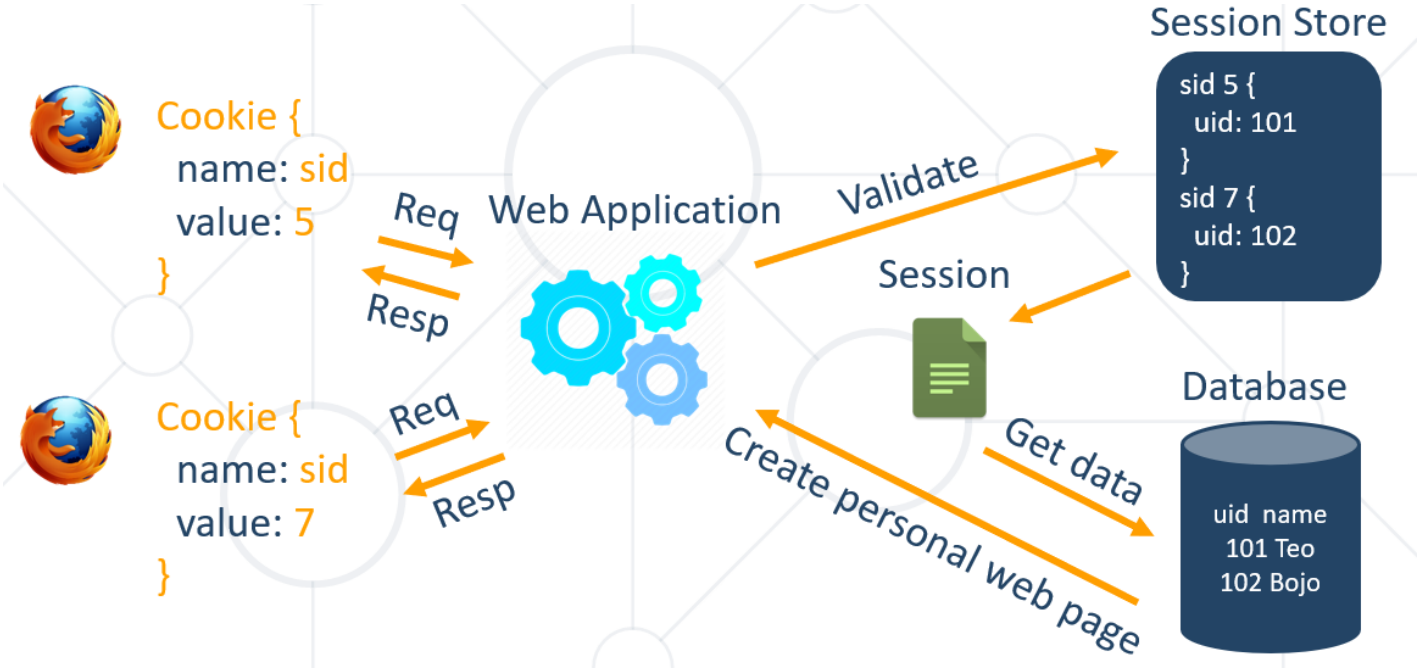


- The exchange mechanism be used between the user and the web application (when server is restarted)

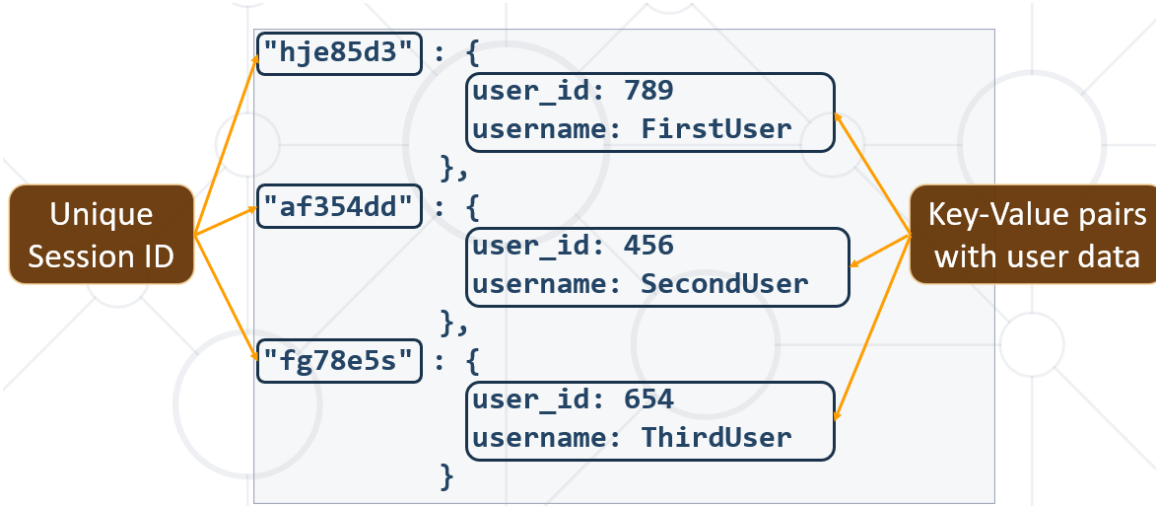


Relation with Cookies

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



Session Structure



5.3. Summary

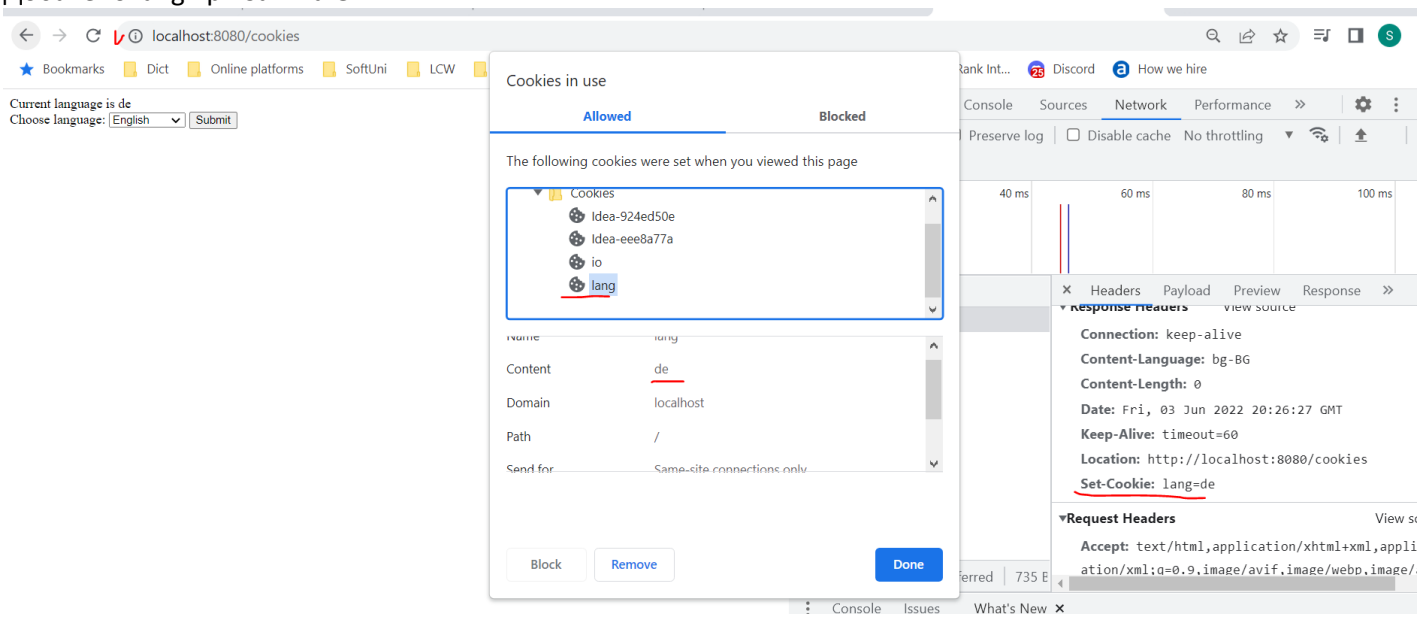
Cookies are **client based** stored information
They are created by web applications
Browser sends them back to the application

Sessions are **server based** information
They are used across multiple pages
Stores important info about the client

5.4. Demo cookies

screenshot

Добавено lang при заявка GET



cookies.html

```
<!doctype html>
<html lang="en"
    xmlns:th="http://www.thymeleaf.org" >
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>

    Current language is <span th:text="${lang}"></span>
    <form th:action="@{/cookies}" th:method="post">
        Choose language:
        <select id="language" name="language">
            <option value="en" th:selected="${lang} == 'en'">English</option>
            <option value="de" th:selected="${lang} == 'de'">Deutsch</option>
            <option value="bg" th:selected="${lang} == 'bg'">Български</option>
        </select>
        <input type="submit" value="Submit">
    </form>

</body>
</html>
```

CookieController

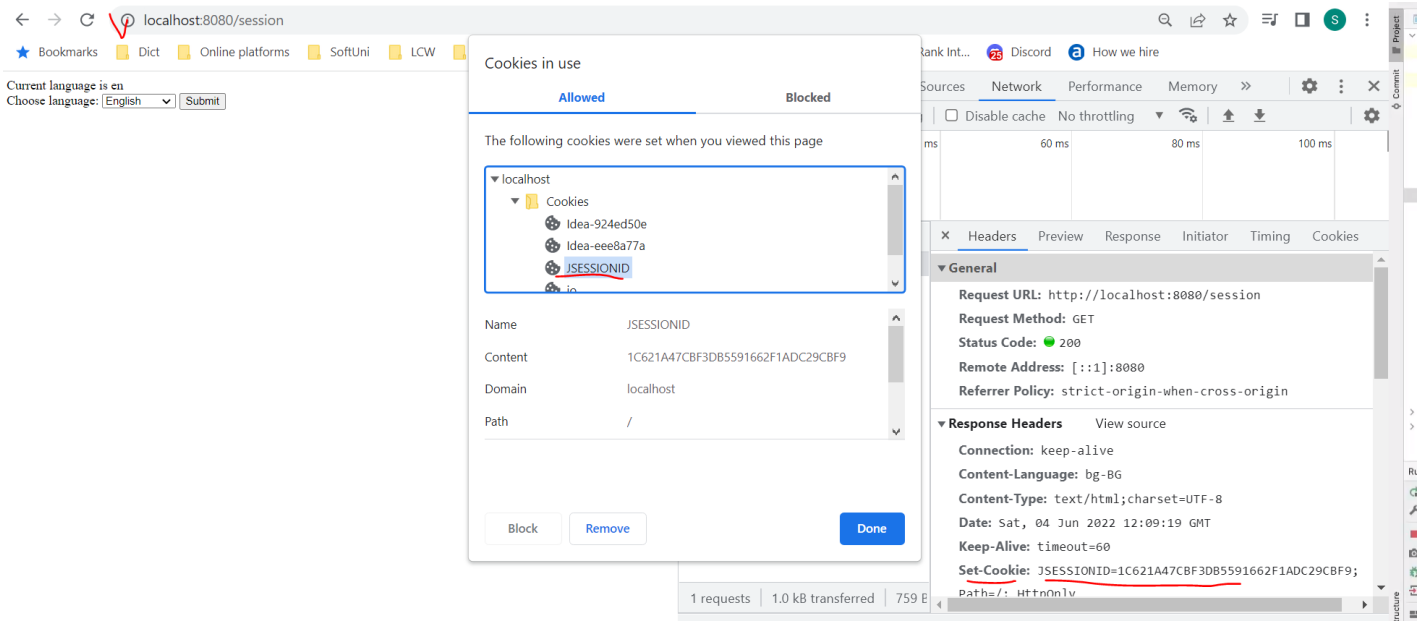
```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.CookieValue;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;

@Controller
public class CookieController {
    private static final String LANG_COOKIE_NAME = "lang";

    @GetMapping("/cookies")
    public String cookies(Model model,
                          @CookieValue(
                                  name = LANG_COOKIE_NAME,
                                  defaultValue = "en"
                          ) String language) {    //We take the cookie value with
@CookieValue(), and we set the default value to "en"
        model.addAttribute("lang", language);

        return "cookies.html";
    }

    @PostMapping("/cookies")
    public String cookies(
            HttpServletResponse response,
            @RequestParam("language") String language
            ) {

        Cookie cookie = new Cookie(LANG_COOKIE_NAME, language);  //how we set a cookie - new
Cookie(name, value)
        response.addCookie(cookie);
response.setStatus(HttpStatus.NotFound)
        return "redirect:/cookies";  //редиректваме към този URL, не към html страница
    }
}
```

5.4. Demo HTTP session

screenshot

Добавено JSessionID при заявка GET



session.html

```
<!doctype html>
<html lang="en"
      xmlns:th="http://www.thymeleaf.org" >
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>

Current language is <span th:text="${language}"></span>
<form th:action="@{/session}" th:method="post">
    Choose language:
    <select id="language" name="language">
        <option value="en" th:selected="${language} == 'en'">English</option>
        <option value="de" th:selected="${language} == 'de'">Deutsch</option>
        <option value="bg" th:selected="${language} == 'bg'">Български</option>
    </select>
    <input type="submit" value="Submit">
</form>

</body>
</html>
```

SessionController

```
import javax.servlet.http.HttpSession;
```

```java
@Controller
public class SessionController {
    private static final String LANG_SESSION_ATTRIBUTE = "lang";
    private static final String DEFAULT_LANG = "en";

    @GetMapping("/session")
    public String session(HttpSession session, Model model) {
        //сесия кода
        var sessionLang = session.getAttribute(LANG_SESSION_ATTRIBUTE);
        sessionLang = sessionLang != null ? sessionLang : DEFAULT_LANG;


        model.addAttribute("language", sessionLang); //към HTML кода

        return "session.html";
    }

    @PostMapping("/session")
    public String session(
            HttpSession session,
            @RequestParam("language") String language
    ) {
        //сесия кода - атрибута LANG_SESSION_ATTRIBUTE реално не е видим в браузъра, но в режим debug в IntelliJ можем да го видим, че се презаписва реално
        session.setAttribute(LANG_SESSION_ATTRIBUTE, language);

        return "redirect:/session"; //редиректваме към този URL, не към html страница
    }
}
```

6. Spring Essentials

6.0. Kotlin example

KotlinApplication.kt

```kotlin
import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

@SpringBootApplication
class KotlinApplication

fun main(args: Array<String>) {
   runApplication<KotlinApplication>(*args)
}
```

hello.html

```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
```

```
</head>
<body>
    <h1 th:text="${greeting}"></h1>

</body>
</html>
```

GreetingService.kt

```
interface GreetingService {
    fun greeting(person: String = "Anonymous") : String  //функция greeting, която връща String. Parameter person, Който е от тип String и има дефаултна стойност
}
```

GreetingServiceImpl.kt

```
import org.springframework.stereotype.Service

@Service
class GreetingServiceImpl : GreetingService {    //GreetingServiceImpl имплементира GreetingService
    override fun greeting(person: String): String {
        return "Hello, $person"
    }
}
```

GreeterController.kt

```
import bg.softuni.kotlin.service.GreetingService
import org.springframework.stereotype.Controller
import org.springframework.ui.Model
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestParam

@Controller
class GreeterController(val greetingService: GreetingService) {  //конструктор в Kotlin

    @GetMapping("/greet")
    fun greet(@RequestParam(name = "person", defaultValue = "Anonymous") person: String, model : Model)
    : String {

        model.addAttribute("greeting", greetingService.greeting(person = person));  //възможност за задаване на име на параметър

        return "hello.html";
    }
}
```

6.0.0. Thymeleaf without Spring

Test.java

```
import org.thymeleaf.TemplateEngine;
import org.thymeleaf.context.Context;
import org.thymeleaf.templateresolver.ClassLoaderTemplateResolver;
import java.io.StringWriter;
```

```
public class Test {
    public static void main(String[] args) {
        TemplateEngine engine = createEngine();
        Context ctx = new Context();
        ctx.setVariable("name", "Pesho");
        StringWriter sw = new StringWriter();

        engine.process("test.html", ctx, sw);

        System.out.println(sw.toString());
    }

    private static TemplateEngine createEngine(){
        TemplateEngine engine = new TemplateEngine();
        engine.setTemplateResolver(new ClassLoaderTemplateResolver());  //test.html ще бъде в
пътя на класа когато се стартира
        return engine;
    }
}
```

test.html

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    Hello, <span th:text="${name}">someone</span>
</body>
</html>
```

console result:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
      content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  Hello, <span>Pesho</span>
</body>
</html>
```

6.1. Thymeleaf – the template engine

What is Thymeleaf?

- Thymeleaf is a modern **server-side** Java **template engine** used in Spring
- It allows us to
  - Use variables in our views
  - Execute operations on our variables
  - Iterate over collections
  - Make our views dynamical

https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html

How to Use Thymeleaf?

- Use Spring Initializer to import Thymeleaf, or use a dependency

In Maven:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

In Gradle:

```
dependencies {
   implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
}
```

```
dependencies {
   compile("org.springframework.boot:spring-boot-starter-thymeleaf")
}
```

- Define the Thymeleaf library in your html

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

Thymeleaf without th:

Реално браузъра си чете HTML кода чист, и за визуална стилизация разкрасителите могат да си работят с th без реално th: да работи.

Escaping in html – като String.format подобно

```
<h3 class="mySticky bg-gray text-light rounded" th:text="|Time to prepare all orders(in min): ${totalTime}|"></h3>
```

```
<div class="card-text"><span th:text="|Seller - First and Last name: *{seller.getFirstName()} *{seller.getLastName()}|">• Seller - First and Last name</span></div>
```

```
<h5 class="card-title" th:text="*{year} + ' ' + *{model.getName()} + ' ' +
*{model.brand.getName()}">
```

Може да вземаме по field на DTO-то или по метод на DTO-то.

```
<img th:src="*{imageUrl}" class="card-img-top" alt="Car image">
<img th:src="*{getImageUrl()}" class="card-img-top" alt="Car image">
```

Thymeleaf Tags and Attributes

- All Thymeleaf tags and attributes begin with **th:** by default
- Example of Thymeleaf attribute

```
<p th:text="${user.name}">Some text</p>
```

- Example of Thymeleaf tag(element processor) – един блок изчезва **като оставя съдържанието си**

```
<th:block>
    ...
</th:block>
```

- **th:block** is an attribute container that **disappears** in the HTML

Thymeleaf Standard Expressions

Variable Expressions

```
${...}
```

Link (URL) Expressions

```
@{...}
```

Selection Expressions

```
*{...}
```

Accessing Bean

```
${@...}
<span th:text="${@loggedUser.getFullName()}"></span>
```

Fragment Expressions

```
~{...}  с тилда ако е по-навътре в пътя/йерархията на дървото на папките/директориите
<header th:replace="~{fragments/commons::headernav}"></header>
```

Web context object - неща, които винаги ги има в контекста (19 Appendix B: Expression Utility Objects)

```
${#session.getAttribute('foo')}
${#session.user.name}
```

```
${#request.getAttribute('foo')}

${#dates.}

${#calendars.}

<small th:if="${#fields.hasErrors('fullname')}"> Some custom errors </small>
```

Thymeleaf Variable Expressions

- Variable Expressions are executed on the context variables

```
${...}
```

- Examples

```
${#session.user.name}
${title}
${game.id}
```

If else & switch

If – else

```
<div th:if="${student.passExam}">Show results</div>  ако е true ще се покаже само
<div th:unless="${student.passExam}">Not pass</div>   ако е false ще се покаже само


<strong th:if="${badCredentials}"     ако съществува badCredentials със стойност ще се покаже само
<strong th:if="${badCredentials} != null" ако съществува badCredentials със стойност ще се покаже само
class="rounded  p1-3 text-danger">Enter valid username and password</strong>
```

Switch

```
<div th:switch="${user.role}">
    <p th:case="'admin'">User is an administrator</p>
    <p th:case="#{roles.manager}">User is a manager</p>
</div>
```

Default expressions (Elvis operator)

- A special kind of conditional value **without a 'then' part**. It is equivalent to the **Elvis** operator present in some languages

```
<p>Age:
    <span th:text="*{age} ?: 'missing age'"> </span>
</p>
```

- Equivalent to:

```
<p>Age:
    <span th:text="*{age != null} ? *{age} :'missing age'"></span>
</p>
```

Thymeleaf Link Expressions

- Link Expressions are used to build URLs

`@{...}`

- Example

```
<a th:href="@{/register}">Register</a>
```

- You can also pass query string parameters

```
<a th:href="@{/details(id=${game.id})}">Details</a>
```

Result -> /details?id=3

- Create dynamic URLs

```
<a th:href="@{/games/{id}/edit(id=${game.id})}">Edit</a>
```

Result -> /games/3/edit

```
<a th:href="@{/orders/ready/{id}(id = *{id})}">Ready</a>
```

Iteration

*When we want to **iterate** over collection*

Ако има 50 студента, то ще се генерират 50 реда като всеки ред ще съдържа name, score и age

```
<tr th:each="s : ${students}">
    <td th:text="${s.name}"></td>
    <td th:text="${s.score}"></td>
    <td th:text="${s.age}"></td>
</tr>
```

*We can attach the **object** to the parent element*

Използваме * като по този начин достъпваме property-то на обекта – за по-съкратен запис се използва реално

```
<tr th:each="s : ${students}" th:object="${s}">
    <td th:text="*{name}"></td>
    <td th:text="*{score}"></td>
    <td th:text="*{age}"></td>
</tr>
```

Пример с dropdown menu:

```
<div class="col-sm-10">
    <select id="category" name="category" class="custom-select"
            th:field="*{category}"
            th:errorclass="is-invalid"       aria-describedby="categoryHelpInline">
        <option value="" selected>Category</option>
        <option value="">Coffee</option>
        <option value="">Cake</option>
        <option value="">Drink</option>
        <option value="">Other</option>
```

typecasting - можем да го направим по следния начин – за dropdown меню:

```
<select id="category" name="category" class="custom-select"
        th:field="*{category}"
        th:errorclass="is-invalid"
```

```
        aria-describedby="categoryHelpInline">
    <option value="" selected>Category</option>
Започваме от root папката
    <option th:each = "c : ${T(com.example.coffeeshop.model.enums.CategoryNameEnum).values()}"
th:value="${c}" th:text="${c}"></option>
</select>
```

Кастваме пътят до Java класа, който разглеждаме, вземаме масив от стойности, и задаваме th:value и th:text
value атрибута заминава към сървъра (**== name**)
text атрибута визуализира на клиента

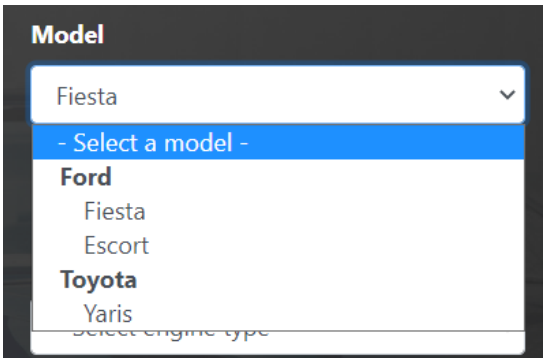
Още един пример за dropdown

```
<div class="form-group col-md-6 mb-3">
    <label class="text-center text-white font-weight-bold"
for="transmission">Transmission</label>
    <select
            id="transmission"
            name="transmission"
            th:errorclass="is-invalid"
            class="form-control"
    >
        <option value="">- Select transmission type -</option>
        <option th:each="e : ${T(bg.softuni.mobilele.model.enums.TransmissionEnum).values()}"
                th:text="${e}"
                th:value="${e}"
                th:selected="${e} == *{transmission}"
        >
            Transmission type
        </option>
    </select>
    <p class="invalid-feedback errors alert alert-danger">
        Transmission type is required.
    </p>
</div>
```

Nested dropdown

```
<form
        th:action="@{/offers/add}"
        th:method="POST"
        th:object="${addOfferModel}"
        class="main-form mx-auto col-md-8 d-flex flex-column justify-content-center">
    <div class="row">
        <div class="form-group col-md-6 mb-3">
            <label class="text-center text-white font-weight-bold" for="modelId">Model</label>
            <select id="modelId"
                    name="modelId"
                    th:errorclass="is-invalid"
                    class="form-control">
                <option value="">- Select a model -</option>
                <optgroup
                        th:each="brand : ${brands}"
                        th:label="${brand.name}">
                    <option
                            th:each="model : ${brand.models}"
                            th:text="${model.name}"
```

```
                            th:value="${model.id}"
                            th:selected="*{modelId} == ${model.id}">
                    </option>
                </optgroup>
            </select>
            <p class="invalid-feedback errors alert alert-danger">
                Vechicle model is required.
            </p>
        </div>
```



Appending and prepending

- **th:attrappend** and **th:attrprepend** attributes, which append (suffix) or prepend (prefix) the result of their evaluation to the existing attribute values

Добавяне на допълнително свойство/стил на даден съществуващ атрибут.

```
<input type="button" value="Play" class="btn" th:attrappend="class=${' ' + cssStyle}" />
```

- **th:classappend**

Добавяне на допълнително свойство/стил на съществуващ атрибут class (тук изрично става въпрос само за атрибута class).

```
<li th:classappend="${module == 'home' ? 'active' : ''}">
```

Forms in Thymeleaf

- In Thymeleaf you can create almost normal HTML forms

```
<form th:action="@{/users}"th:method="post">
    <input type="number" name="id"/>
    <input type="text" name="name"/>
    <button type="submit"/>
</form>
```

- You can have a controller that will accept an object of given type

```
@PostMapping("/user")
public ModelAndView register(User user) { ... }
```

Fragments in Thymeleaf

Create Fragment with th:fragment

- Often we want to include in our templates **fragments** from other **templates**
  - Common uses for this are footers, headers, menus
  - Define the fragments available for inclusion, which we can do by using the **th:fragment** attribute
  - After than we can easily include in our home page using one of the **th:include** or **th:replace** attributes

Правим си navbar.html с менюто в случая, и във всяка от другите template pages го извикваме така:

index.html

```
<div th:replace="fragments/navbar.html">Navbar</div>
```

navbar.html

```
<!doctype html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
<div>
    <nav class="navbar navbar-expand-lg bg-dark navbar-dark fixed-top">
        <a class="navbar-brand" href="/"><img alt="car picture" src="/img/car.png"
class="logo"></a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarSupportedContent"
                aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle
navigation">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav mr-auto col-12 justify-content-between">
                <li class="nav-item"
                >
                    <a class="nav-link" href="/brands/all">All Brands</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/offers/add">Add Offer</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/offers/all">All Offers</a>
                </li>
                <li class="nav-item dropdown">
                    <a class="nav-link dropdown-toggle" href="/" id="navbarDropdown"
role="button"
                       data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                        Admin
                    </a>
                    <div class="dropdown-menu" aria-labelledby="navbarDropdown">
                        <a class="dropdown-item" href="/">Action</a>
                        <a class="dropdown-item" href="/">Another action</a>
```

```
                        <div class="dropdown-divider"></div>
                        <a class="dropdown-item" href="/">Something else here</a>
                    </div>
                </li>
                <li class="nav-item" th:if="${@currentUser.isLoggedIn()}">
                    <div class="form-inline my-2 my-lg-0 border px-3">
                        <div class="logged-user"
                             text="Welcome, Gosho"></div>
                        <a class="nav-link" th:href="@{/users/logout}">Logout</a>
                    </div>
                </li>

                <li class="nav-item" th:if="${@currentUser.isAnonymous()}">
                    <a class="nav-link" th:href="@{/users/register}">Register</a>
                </li>
                <li class="nav-item" th:if="${@currentUser.isAnonymous()}">
                    <a class="nav-link" th:href="@{/users/login}">Login</a>
                </li>

            </ul>
        </div>
    </nav>
</div>
</body>
</html>
```

- Create class with fragments

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<body>
<div th:fragment="copy">
    &copy; Spring Team 2021
</div>
</body>
</html>
```

- Easily include in our home page using one of the **th:include** or **th:replace** attributes

```
<body>
...
<footer th:include="footer::copy"></footer>
//OR
<footer th:replace="footer::copy"></footer>
...
</body>
```

- Difference between include and replace

```
<footer th:include="footer :: copy"></footer>
<footer th:replace="footer :: copy"></ footer>
```

The result is

```
<footer>
    &copy; Spring Team 2021
</footer>
```

```
<div>
    &copy; Spring Team 2021
</div>
```

Create Fragment with ***th:fragment – extended***

index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head th:replace="fragments/commons::head"></head>

<body class="bg-secondary">
      <header th:replace="fragments/commons::nav"></header>

      <main role="main" class="bg-secondary"></main>

      <footer th:replace="fragments/commons::footer"></footer>
</body>
</html>
```

commons.html

```
<!doctype html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head th:fragment="head">
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Coffee Shop Application</title>
    <link rel="stylesheet" href="/css/reset-css.css"/>
    <link rel="stylesheet" href="/css/bootstrap.min.css"/>
    <link rel="stylesheet" href="/css/style.css"/>
</head>
<body>

<header th:fragment="nav">
    <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-info rounded">
        <div class="collapse navbar-collapse" id="navbarsExampleDefault">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="/">Home</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/users/login">Login</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/users/register">Register</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/orders/add">Add Order</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/users/logout">Logout</a>
                </li>
            </ul>
```

```
            <img width="50px" src="/images/coffee.png" alt="Coffee image">
        </div>
    </nav>
</header>

<footer th:fragment="footer" class="container">
    <p>&copy;SoftUni Spring Team 2021. All rights reserved.</p>
</footer>

</body>
</html>
```

*Create Fragment **without th:fragment** – не е добра практика да се прави*

footer.html

```
<th:block>
    <footer> Spring Team 2020 </footer>
</th:block>
```

- Use Fragment

index.html

```
...
<th:block th:include="~{/fragments/footer.html}>    </th:block>
...
```

6.2. Additional Spring Functionalities

@ModelAttribute

- When the annotation is used at the **method level**, it indicates **the purpose of that method**
  - to add one or more model attributes
- In the example, a method adds an attribute named **message** to all models defined in the controller class

method level

Пример 1:

```
 @ModelAttribute
public void addAttributes(Model model) {
    model.addAttribute("message", "Welcome to SoftUni!");
}
```

Пример 2:

```
//    The name of the model attribute to bind to.
//    The default model attribute name is inferred from the declared attribute type
//    (i.e. the method parameter type or method return type), based on the non-qualified class
name:
//    e.g. "orderAddress" for class "mypackage.OrderAddress",
//    or "orderAddressList" for "List<mypackage.OrderAddress>".
```

```
@ModelAttribute() //тук името на атрибута идва автоматично от името на класа
public OrderAddBindingModelDTO orderAddBindingModelDTO(){
    return new OrderAddBindingModelDTO();
}

@PostMapping("/add")
public String addConfirm(@Valid OrderAddBindingModelDTO orderAddBindingModelDTO, BindingResult
bindingResult,
                         RedirectAttributes redirectAttributes) {

    if (bindingResult.hasErrors()) {
        redirectAttributes.addFlashAttribute("orderAddBindingModelDTO",
orderAddBindingModelDTO);

redirectAttributes.addFlashAttribute("org.springframework.validation.BindingResult.orderAddBindi
ngModelDTO", bindingResult);

        return "redirect:add";
    }

    //add to the DB
    return "redirect:/";
}


<form th:action="@{/orders/add}" th:method="POST" th:object="orderAddBindingModelDTO"
```

Пример 3:

```
@ModelAttribute("userModel")  //изпълнява се в рамките на текущия контролер само!!!
public void initUserModel(Model model){
    model.addAttribute("userModel", new UserRegisterDto());
}

@GetMapping("/register")
public String register() {
    // когато зареждаме за първи път страницата, то автомиатично ще влезе към модела атрибут
userModel == празен new UserRegisterDto()
    return "auth-register.html";
}
```

method argument

- When used as a **method argument**, it **indicates the argument** should be retrieved from the model
- When **not present**, it should be **first instantiated** and then added to the model.
- Once **present in the model**, the arguments **fields should be populated** from all request parameters that have matching names.
- Example of using **@ModelAttribute as a method argument**

```
@RequestMapping(value = "/cars/add",
        method = RequestMethod.POST)
public String submit(@ModelAttribute("car") Car car) {
// Some code ...
```

```
        return "carView.html";
    }
```

алтернтива на modelAttribute

```
@GetMapping("/offers/add")
public String addOffer(Model model){
    if (!model.containsAttribute("addOfferModel")) {
        model.addAttribute("addOfferModel", new AddOfferDTO());
    }
    return "offer-add";
}
```

@CrossOrigin

- **@CrossOrigin**
  - marks the annotated method or type as permitting cross origin requests

```
@CrossOrigin(origins = "http://example.com")
@RequestMapping("/hello")
public String hello() {
    return "Hello World!";
}
```

Same Origin Policy – само JS заявки от същия сайт на банката могат да изпращат заявки към сървъра на банката. Ако е позволен CrossOrigin, и сме цъкнали на фалшив сайт в нов tab/page примерно на нашия браузър, то фалшивия сайт има достъп до нашите cookie-та, и може да изпрати легитимна заявка, с която да ни вземе парите.

CORS – разрешение от сървъра да се извикват някакви JS кодове

При 3ply заявки, винаги се пускат 2 http заявки – един път request method е OPTIONS като предварително разрешение, и втора заявка вече реалната заявка.

Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: името на сайта / IP адреса, който да има достъп

```
import org.springframework.stereotype.Controller
import org.springframework.ui.Model
import org.springframework.web.bind.annotation.CrossOrigin
import org.springframework.web.bind.annotation.PostMapping
import org.springframework.web.bind.annotation.RequestParam

@Controller
class BankController {

//    @CrossOrigin(origins = arrayOf("http://example.com"))
    @CrossOrigin("*") //отвсякъде да има достъп
    @PostMapping("/transfer")
    fun transferMoney(@RequestParam("from") from : String,
                      @RequestParam("to") to : String,
                      model: Model
```

```
    ) : String {
        model.addAttribute("from", from);
        model.addAttribute("to", to);
        return "transfer.html"
    }
}
```

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    Money transferred from <span th:text="${from}">FROM</span>
        to <span th:text="${to}">TO</span>
</body>
</html>
```

@Qualifier

- We use **@Qualifier** along with **@Autowired** to provide the bean id or bean name

```
@Component
@Qualifier("bike")
class Bike implements Vehicle {
    private String make;
    private String model;
}

@Component
@Qualifier("car")
class Car implements Vehicle {
    private String make;
    private String model;
    private Integer seats;
}
```

- If we want to get Bike, we need to specify it with adding **@Qualifier("bike")** before injecting Vehicle

```
@Autowired
Biker(@Qualifier("bike") Vehicle vehicle) {
    this.vehicle = vehicle;
}
```

@Primary

- We can use **@Primary** to simplify this case:
    - if we mark the most frequently used bean with **@Primary**

```
@Component
@Qualifier("bike")
class Bike implements Vehicle {
```

```
    private String make;
    private String model;
}

@Component
@Primary
class Car implements Vehicle {
    private String make;
    private String model;
    private Integer seats;
}
```

- The example of **@Primary** use case

```
@Component
class Driver {
    @Autowired
    Vehicle vehicle;
}


@Component
class Biker {
    @Autowired
    @Qualifier("bike")
    Vehicle vehicle;
}
```

6.3. Scope of the URL

```
@Controller
@RequestMapping("/users")
public class UserController {

return "redirect:/users/login";  е същото като
return "redirect:login"; //re-direct-ва към текущия път /user плюс login, като ни връща и
сесията JSessionId
}
```

6.4. Working with Http Sessions, Cookies and Headers

HTTP session и Cookies се ползват впоследствие в Headers на Http REST/AJAX заявките.

Working with the Session

- The session will be **injected from the IoC** container when called

```
  @GetMapping("/")
  public String home(HttpSession httpSession) {
…
      httpSession.setAttribute("id", 2);
…
  }
```

```java
@Controller
public class HomeController {

    @GetMapping("/")
    public String index(HttpSession httpSession) {

        return httpSession.getAttribute( s: "user") == null
                ? "index" : "home";
    }
}
```

Така го логваме потребителя:

```java
    httpSession.setAttribute( s: "user", userServiceModel);
```

Така го разлогваме - invalidate

```java
@GetMapping("/logout")
public String logout(HttpSession httpSession){
    httpSession.invalidate();

    return "redirect:/";
}
```

- Later the session attributes can be accessed from Thymeleaf using the expression syntax and the **#session** object

Reading HTTP Cookie

- The annotation **@CookieValue**

```
От браузъра към вървъра
@GetMapping("/")
public String readCookie(@CookieValue(name или value (едно и също е) = "username", defaultValue
= "Guest") String username) {
    return "login";
}
```

- Using the **ResponseCookie** object

```
ResponseCookie cookie = ResponseCookie.from("username", "pesho")
        .httpOnly(true)
        .secure(true)
        .path("/")
        .maxAge(60)
```

```
        .domain("softuni.bg")
        .build();   //builder for cookies

ResponseEntity
    .ok()
    .header(HttpHeaders.SET_COOKIE, cookie.toString())
    .build();
```

- **@CookieValue** – по по-нормален начин е тук

```
@GetMapping("/change-username")
public String setCookie(HttpServletResponse response) {
    // create a cookie
    Cookie cookie = new Cookie("username", "Pesho");
    //add cookie to response
    response.addCookie(cookie);
response.setStatus(HttpStatus.NotFound)
    return "index";
}
```

RequestHeader

- Reading **HTTP Header**

```
@GetMapping("/greeting")
public ResponseEntity<String> greeting(
        @RequestHeader("accept-language") String language) {
    // code that uses the language variable
    return new ResponseEntity<String>("greeting", HttpStatus.OK);
}
```

ResponseStatus

- We can specify the desired **HTTP status** of the response

```
@RequestMapping(method = RequestMethod.POST)
@ResponseStatus(HttpStatus.CREATED)
public void storeEmployee(@RequestBody Employee employee) {
...
}
```

6.5. Request & Response Body

@RequestBody

- Maps the **HttpRequest body** to a transfer or domain object, enabling automatic deserialization of the inbound HttpRequest body on to a Java objects – **десериализация на body-то до Java обект**

```
@PostMapping("/students/add")
public ResponseEntity postController(
        @RequestBody StudentAddBindingModel bindingModel){
    myService.add(bindingModel);
    return ResponseEntity.ok(HttpStatus.OK);
}
```

Въпреки че много често http request body-то е JSON обект.

@ResponseBody

- Tells a controller that the object returned is automatically serialized into **JSON** and passed back into the **HttpResponse object**

Сериализирай го като JSON и го сложи като част/цялото body на http reponse-а

```
@GetMapping("/response")
@ResponseBody
public Exercise getLastEx() {
    // Get exercise from service
    return exercise;
}
```

```
{"id":"0b5963eb-4f4d-4718-bd34-
d0206d80046a","name":"SPRING DATA
INTRO","startedOn":"2021-01-
14T19:26:00","dueDate":"2021-02-05T19:26:00"}
```

6.6. Session as @Bean - Security access requirements - Example

The **`Security Requirements`** are mainly access requirements. Configurations about which users can access specific functionalities and pages. – backend functionality checkings / validations – във всичките контролери да добавим currentUser-а и да направим if-ове дали е логнат – и готово.

```
@Component
@SessionScope
public class LoggedUser {
    private Long id;
    private String userame;

    public void login(UserEntity user){
        this.id = user.getId();
        this.userame = user.getUsername();
    }

    public LoggedUser() {
        this.id = null;
        this.userame = null;
    }

    public void logout(){
        this.id = null;
        this.userame = null;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUserame() {
```

```
        return userame;
    }

    public void setUserame(String userame) {
        this.userame = userame;
    }
}


@Controller
public class HomeController {
    private final LoggedUser loggedUser;

    public HomeController(LoggedUser loggedUser) {
        this.loggedUser = loggedUser;
    }

    @GetMapping("/")
    public String loggedOutIndex(){
        //if user is already logged-in
        if (loggedUser.getId() != null) {
            return "redirect:/home";
        }
        return "index";
    }

    @GetMapping("/home")
    public String loggedInIndex(){
        //if user is logged-out
        if (loggedUser.getId() == null) {
            return "redirect:/";
        }
        return "home";
    }
}
```

Задължително слагаме и на Get заявката, и на Post заявката!!!

```java
@GetMapping("/login")
public String login(Model model) {
    //If user has already logged in
    if (loggedUser.getId() != null) {
        return "redirect:/home";
    }

    if (!model.containsAttribute( attributeName: "userLoginBindingDto")) {
        model.addAttribute( attributeName: "userLoginBindingDto", new UserLoginBindingDto());
        model.addAttribute( attributeName: "notFound",  attributeValue: false);
    }

    return "login";
}

@PostMapping("/login")
public String loginConfirm(@Valid UserLoginBindingDto userLoginBindingDto,
                           BindingResult bindingResult, RedirectAttributes redirectAttributes){
    //If user has already logged in
    if (loggedUser.getId() != null) {
        return "redirect:/home";
```

```
<a th:if="${@loggedUser.getId() != null}" class="nav-link text-white active h5" href="/">Add
Product</a>
```

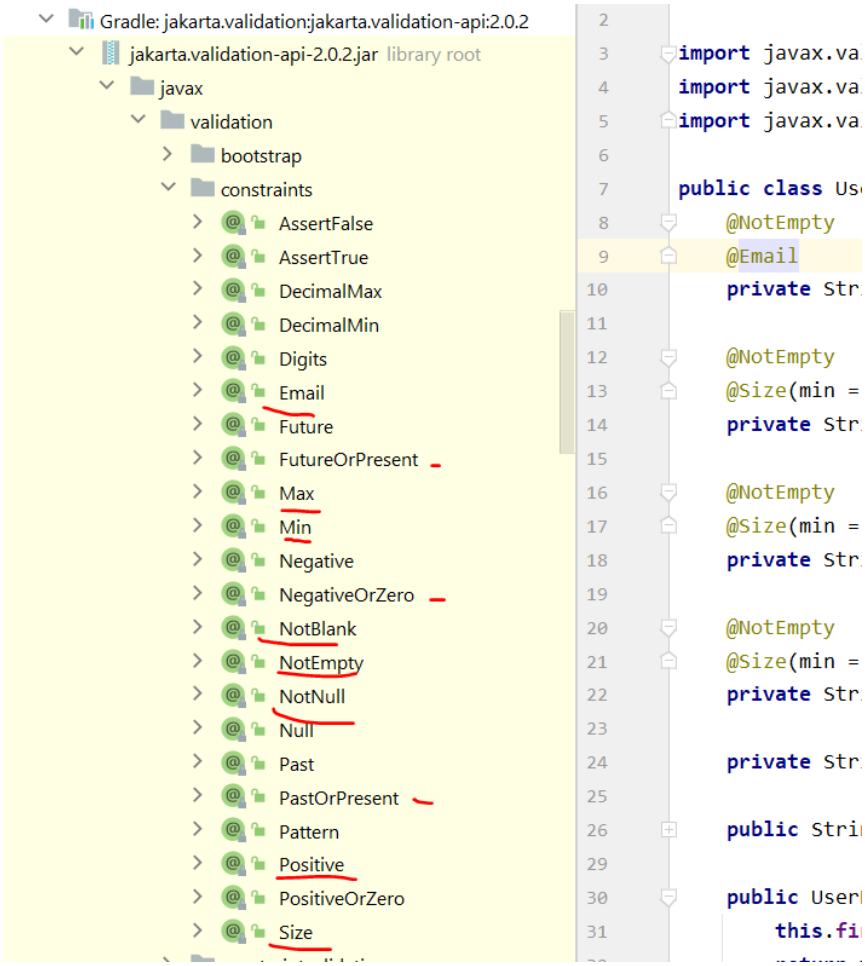
7. Thymeleaf & Validation

https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html

Server-side validation

Gradle dependency

```
implementation 'org.springframework.boot:spring-boot-starter-validation'
===
javax.validation.constraints
|    \--- org.hibernate.validator:hibernate-validator:6.2.3.Final
|         +--- jakarta.validation:jakarta.validation-api:2.0.2
|         +--- org.jboss.logging:jboss-logging:3.4.1.Final -> 3.4.3.Final
```



Pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

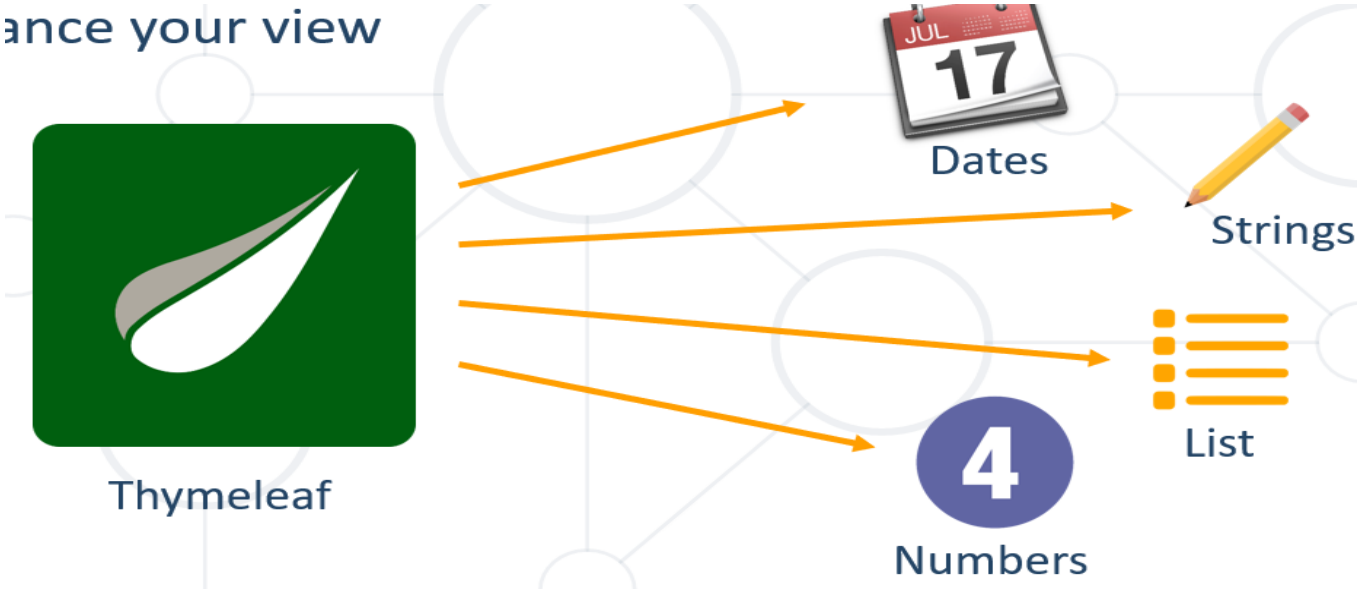
Client-side validation

```
<div class="form-group">
    <label for="username" class="text-white font-weight-bold">E-mail(username)</label>
    <input value="username" id="username" name="username" type="text" min="2" max="50"
           class="form-control"
           placeholder="Username"/>
</div>
```

7.1. Thymeleaf Helpers

Helpers

- Objects that provide built-in functionalities that helps you enhance your view



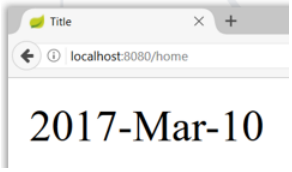
Date – Custom Format

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model){
    model.addAttribute("myDate", new Date());
    return "whiskey-home.html";
}
```

whiskey-home.html

```
<div th:text="${#dates.format(myDate,'yyyy-MMM-dd')}"></div>
```



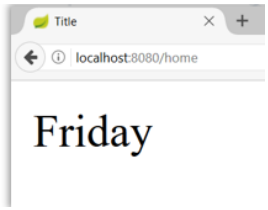
Date – Week Name of Day

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model){
    model.addAttribute("myDate", new Date());
    return "whiskey-home.html";
}
```

whiskey-home.html

```
<div th:text="${#dates.dayOfWeekName(myDate)}"></div>
```



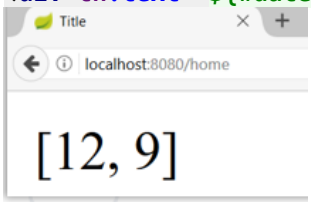
Date – List Days

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model){
    // List of dates -> 2016-12-12, 2017-04-09 -> yyyy-MM-dd
    model.addAttribute("myDates", myDates);
    return "whiskey-home";
}
```

whiskey-home.html

```
<div th:text="${#dates.listDay(myDates)}"></div>
```



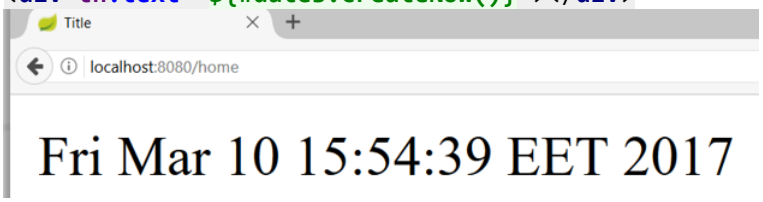
Date – Get Current Date

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage() {
    return "whiskey-home";
}
```

whiskey-home.html

```
<div th:text="${#dates.createNow()}"></div>
```



LocalDate and Thymeleaf

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model){
    model.addAttribute("myDate", LocalDate.now());
    return "whiskey-home";
}
```

whiskey-home.html

```
${#temporals.format(myDate, 'dd-MMM-yyyy')}|
```

- To use **LocalDate** we need to add new **dependency** (for old versions only needed)

```
<dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-java8time</artifactId>
        <version>3.0.4.RELEASE</version>
</dependency>
```

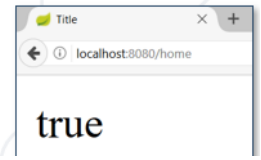
Strings – is Empty

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model) {
    String whiskeyNull = null;
    model.addAttribute("whiskey", whiskeyNull);
    return "whiskey-home";
}
```

whiskey-home.html

```
<div th:text="${#strings.isEmpty(whiskey)}"></div>
```



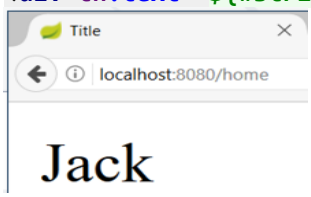
Strings – Substring

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model) {
    String whiskey = "Jack Daniels";
    model.addAttribute("whiskey", whiskey);
    return "whiskey-home";
}
```

whiskey-home.html

```
<div th:text="${#strings.substring(whiskey,0,4)}"></div>
```



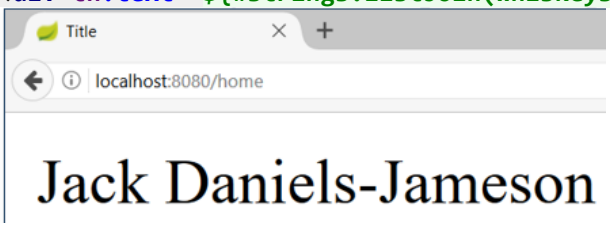
Strings – Join

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model) {
    List<String> whiskeys = List.of("Jack Daniels", "Jameson");
    model.addAttribute("whiskeys", whiskeys);
    return "whiskey-home";
}
```

whiskey-home.html

```
<div th:text="${#strings.listJoin(whiskeys,'-')}"></div>    раздели със запетая
```



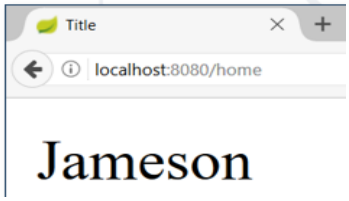
Strings – Capitalize

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model) {
    String whiskey = "jameson";
    model.addAttribute("whiskey", whiskey);
    return "whiskey-home";
}
```

whiskey-home.html

```
<div th:text="${#strings.capitalize(whiskey)}"></div>  -  само първата буква капитализира
```



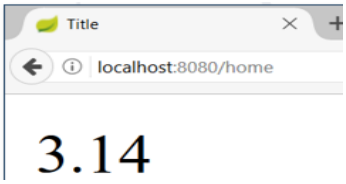
Numbers – Format

MathController.java

```
@GetMapping("/home")
public String getHomePage(Model model) {
    double num = 3.14159;
    model.addAttribute("num", num);
    return "home";
}
```

home.html

```
<div th:text="${#numbers.formatDecimal(num,1,2)}"></div>
```



Numbers – Sequence

MathController.java

```
@GetMapping("/home")
public String getHomePage(Model model) {
    return "home";
}
```

home.html

```
<span th:each="number: ${#numbers.sequence(0,2)}">
    <span th:text="${number}"></span>
</span>
```

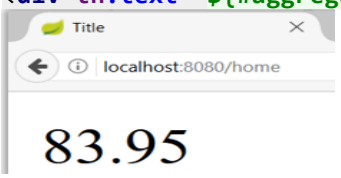
Aggregates – Sum

WhiskeyController.java

```
@GetMapping("/home")
public String getHomePage(Model model) {
    double[] whiskeyPrices = new double[]{29.23, 21.22, 33.50};
    model.addAttribute("whiskeyPrices", whiskeyPrices);
    return "whiskey-home";
}
```

whiskey-home.html

```
<div th:text="${#aggregates.sum(whiskeyPrices)}
```



Text inlining

Although the Standard Dialect allows us to do almost everything we might need by using tag attributes, there are situations in which we could prefer writing expressions directly into our HTML texts. For example, we could prefer writing this:

```
p>Hello, [[${session.user.name}]]!</p>
```

…instead of this:

```
p>Hello, <span th:text="${session.user.name}">Sebastian</span>!</p>
```

Expressions between `[[...]]` are considered expression inlining in Thymeleaf, and in them you can use any kind of expression that would also be valid in a `th:text` attribute.

In order for inlining to work, we must activate it by using the `th:inline` attribute, which has three possible values or modes (**`text`**, **`javascript`** and **`none`**). Let's try `text`:

```
<p th:inline="text">Hello, [[${session.user.name}]]!</p>
```

The tag holding the `th:inline` does not have to be the one containing the inlined expression/s, any parent tag would do:

```
<body th:inline="text">
...
<p>Hello, [[${session.user.name}]]!</p>
...
</body>
```

So you might now be asking: *Why aren't we doing this from the beginning? It's less code than all those `th:text` attributes!* Well, be careful there, because although you might find inlining quite interesting, you should always remember that inlined expressions will be displayed verbatim in your HTML files when you open them statically, so you probably won't be able to use them as prototypes anymore!

The difference between how a browser would statically display our fragment of code without using inlining…

```
Hello, Sebastian!
```

…and using it…

```
Hello, [[${session.user.name}]]!
```

…is quite clear.

When inlining, [[…]] corresponds to th:text and [(…)] corresponds to th:utext.

Thymeleaf in JavaScript - text inlining

JSController.java

```
@GetMapping("/js")
public String getMapPage(Model model){
    String message = "Hi JS!";
    model.addAttribute("message", message);
    return "page.html";
}
```

script.js

```
<script th:inline="javascript">
    let message = [[${message}]];
<script>
```

Text inlining

7.2. How to Validate?

Spring Validation & Thymeleaf

- Making a simple **Model validation** and **Error rendering**

Пример 1:

```
public class SomeModel {

    @NotNull
    @Size(min = 3, max = 10,
            message = "Invalid name")
    private String name;
}

@Controller
public class SomeController {
    @GetMapping("/add")
    public String getPage(Model model) {

        if(!model.containsAttribute("bindingModel"){
            model.addAttribute("bindingModel",
                    new BindingModel());
        }

        return "add";
    }
}
```

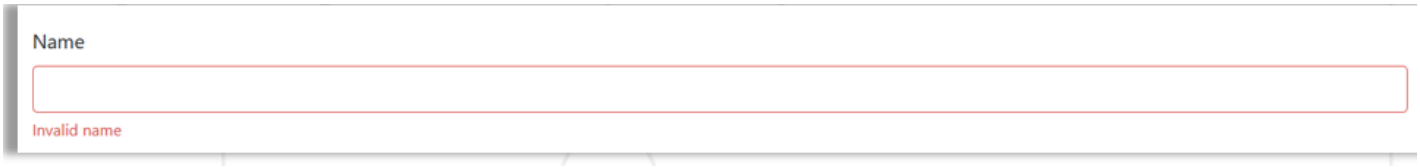
//поредността на параметрите на add метода са в строга последователност!!! – Spring inject-ва валидирания модел, който и веднага се bind-ва.

```
@PostMapping("/add")
public String add(@Valid @ModelAttribute("bindingModel") SomeModel bindingModel,
BindingResult bindingResult, RedirectAttributes rAtt) {
    if (bindingResult.hasErrors()) {
        rAtt.addFlashAttribute("bindingModel", bindingModel);  //flash – ще рече след
презареждане на URL-то на страницата да предадем данните на template. Нещо, което живее в
рамките на следващия един http request.
        rAtt.addFlashAttribute("org.springframework.validation.BindingResult.bindingModel",
bindingResult);
        return "redirect:/add";  //редиректваме към този URL, не към html страница
    }

    this.someService.save(bindingModel);
```

```
    return "redirect:/home"; //редиректваме към този URL, не към html страница
}
```

add.html
Когато имаме атрибут th:field, то нямаме нужда от атрибут name в html кода.

```
<div th:object="${ bindingModel }">
    <div class="justify-content-center">
        <label for="name" class="h4 mb-2 text-white">Name</label>
    </div>
    <input th:field="*{name}" th:errorclass="bg-danger" type="text" class="form-control"
id="name" name="name"/>
    <small th:if="${#fields.hasErrors('name')}"
           th:errors="*{name}" class="text-danger"> Name error</small>
</div>
```



Пример 2:

AuthController.java

```
@Controller
public class AuthController {

    //Ако го има вече инициализиран userToRegister, то го прескача и не го инициализира отново с
празни стойности
@ModelAttribute(name = "userToRegister")
public UserRegisterBindingModelDTO userRegisterBindingModelDTO(){
    return new UserRegisterBindingModelDTO();
}


    @GetMapping("/register")
    public String register(Model model) {
//        this.userRepo
        return "register.html";
    }

@PostMapping("/register")
public String registerConfirm(@Valid UserRegisterBindingModelDTO userRegisterBindingModelDTO,
                              BindingResult bindingResult, RedirectAttributes
redirectAttributes){

    System.out.println(userRegisterBindingModelDTO);

    if (bindingResult.hasErrors() ||

!userRegisterBindingModelDTO.getPassword().equals(userRegisterBindingModelDTO.getConfirmPassword
())) {
```

```
            //pass dto to template
        redirectAttributes.addFlashAttribute("userToRegister", userRegisterBindingModelDTO);

        //pass errors to template
        redirectAttributes.addFlashAttribute(
                "org.springframework.validation.BindingResult.userToRegister",
                bindingResult);

            //Добавяне на custom грешка чрез reject
            bindingResult.reject("password", new String[]{"passwords.matching"}, "Password id
not match");


        return "redirect:/users/register";
    }

    //TODO: save in db
        //check if passwords are the same
        // check if username/email exists in the db
        //insert in DB

    return "redirect:/users/login";
}
```

register.html

```
<form
        th:action="@{/register}"
        th:method="post"
        th:object="${userToRegister}"
        class="registration-form"
>
    <div>
        <div class="col-auto">
            <label for="inputUsername" class="col-form-label ">Username</label>
        </div>
        <div class="col-auto">
            <input name="username"
                   th:field="*{username}"
                   th:errorclass="is-invalid"   добавя се след това в class
атрибута чрез Bootstrap
                   type="text"
                   id="inputUsername"
                   class="form-control"
                   aria-describedby="usernameHelpInline">
            <small id="usernameError"
                   class="invalid-feedback bg-danger rounded">Username length must be more
than 3
                characters</small>

            <small id="usernameUniqueError"
                   class=" bg-danger rounded">Username is already occupied</small>

        </div>
    </div>
```

```
    <div>
        <div class="col-auto">
            <label for="inputFullName" class="col-form-label ">Full Name</label>
        </div>
        <div class="col-auto">
            <input name="fullname" //служи за формиране на FormData на браузъра
                   th:field="*{fullname}"
                      th:errorclass="is-invalid"
                   type="text"
                   id="inputFullName"
                   class="form-control"
                   aria-describedby="fullNameHelpInline">
            <small id="fullNameError"
                   class="invalid-feedback  form-text bg-danger rounded">Full name length
must be more than
                3 characters</small>
              <small th:if="${#fields.hasErrors('fullname')}"> Some custom errors
</small>
        </div>
    </div>

    <div>
        <div class="col-auto">
            <label for="inputEmail" class="col-form-label ">Email</label>
        </div>
        <div class="col-auto">
            <input name="email"
                   th:field="*{email}"
                 th:errorclass="is-invalid"
                   type="email"
                   id="inputEmail"
                   class="form-control"
                   aria-describedby="emailHelpInline">
            <small id="emailError" th:if="${#fields.hasErrors('email')}"
                   class="invalid-feedback  form-text bg-danger rounded">Must be valid
email</small>
        </div>
    </div>
    <div>
        <div class="col-auto">
            <label for="inputAge" class="col-form-label ">Age</label>
        </div>
        <div class="col-auto">
            <input name="age" required
                   th:field="*{age}"
                 th:errorclass="is-invalid"
                   type="number"
                   id="inputAge"
                   class="form-control"
                   min="0" max="90"
                   aria-describedby="ageHelpInline">
            <small id="ageError"
                   class="invalid-feedback  form-text bg-danger rounded">Must be valid
age</small>
        </div>
    </div>
```

```
    <div>
        <div class="col-auto">
            <label for="inputPassword" class="col-form-label ">Password</label>
        </div>
        <div class="col-auto">
            <input name="password"
                   th:field="*{password}"
                th:errorclass="is-invalid"
                   required minlength="5" maxlength="20"
                   type="password"
                   id="inputPassword"
                   class="form-control"
                   aria-describedby="passwordHelpInline">
            <small id="passwordError"
                   class="invalid-feedback  form-text bg-danger rounded">
                Password length must be between 5 and 20 characters and passwords should match.
            </small>
        </div>
    </div>


    <div class="d-flex justify-content-center mt-4">
        <button class="btn btn-primary btn-block w-50" type="submit">Register</button>
    </div>

</form>
```

List/render All Errors

add.html

```
<ul th:if="${#fields.hasErrors('*')}">
    <li th:each="err : ${#fields.errors('*')}" th:text="${err}">
        Input is incorrect
    </li>
</ul>
```

add.html

```
<ul th:if="${#fields.hasErrors('${someModel.*}')}">
    <li th:each="err : ${#fields.errors('${someModel.*}')}" th:text="${err}">
        Input is incorrect
    </li>
</ul>
```

- Invalid creator
- Invalid name
- Mutation cannot be null
- Invalid description
- Invalid hours
- You must select capitals

Пример

login.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head th:replace="fragments/commons::head"></head>
<body class="bg-secondary">
<header th:replace="fragments/commons::nav"></header>

<main role="main" class="bg-secondary">
    <div class="jumbotron">
        <div class="container text-light ">
            <h1 class="display-3">
                <strong class="bg-blur rounded border-left border-white pl-3 border-
bottom">Login</strong></h1>
            <h3 class="mt-5 text-center"><strong class="bg-blur rounded border-white pl-3
border-bottom">Enter valid
                username and
                password</strong>
            </h3>
        </div>

        <div class="container bg-blur rounded p-5 mt-5 w-75 ">
            <form th:action="@{/users/login}" method="POST" th:object="${userToLogin}"
                  class="text-center text-light">
                <h3 th:if="${isFound == false}" class="mt-5 text-center">
                    <strong class="bg-blur rounded text-danger">
                        Wrong username and password combination.
                    </strong>
                </h3>

                <div class="form-group row">
                    <label for="username" class="col-sm-2 col-form-label">Username</label>
                    <div class="col-sm-10">
                        <input type="text"
                               class="form-control"
                               th:field="*{username}"
                               th:errorclass="bg-danger"
                               id="username"
                               aria-describedby="usernameHelpInline" placeholder="Username">
                        <small id="usernameHelpInline"
                               th:if="${#fields.hasErrors('username')}"
                               th:errors="*{username}"
                               class="bg-danger text-light rounded">
                            Username length must be between 5 and 20 characters.
                        </small>
                    </div>
                </div>

                <div class="form-group row">
                    <label for="password" class="col-sm-2 col-form-label">Password</label>
                    <div class="col-sm-10">
                        <input type="password"
                               class="form-control"
                               th:field="*{password}"
                               th:errorclass="bg-danger"    from bootstrap - background-danger
                               id="password"
                               aria-describedby="passwordHelpInline" placeholder="Password">
```

```
                        <small id="passwordHelpInline"
                               th:if="${#fields.hasErrors('password')}"
                               th:errors="*{password}"
                               class="bg-danger text-light rounded">
                            Password length must be more than 3 characters.
                        </small>
                    </div>
                </div>
                <button type="submit" class="btn btn-info w-50">Login</button>
            </form>
            <hr class="bg-light">
        </div>
    </div>
</main>

<footer th:replace="fragments/commons::footer"></footer>

</body>
</html>
```

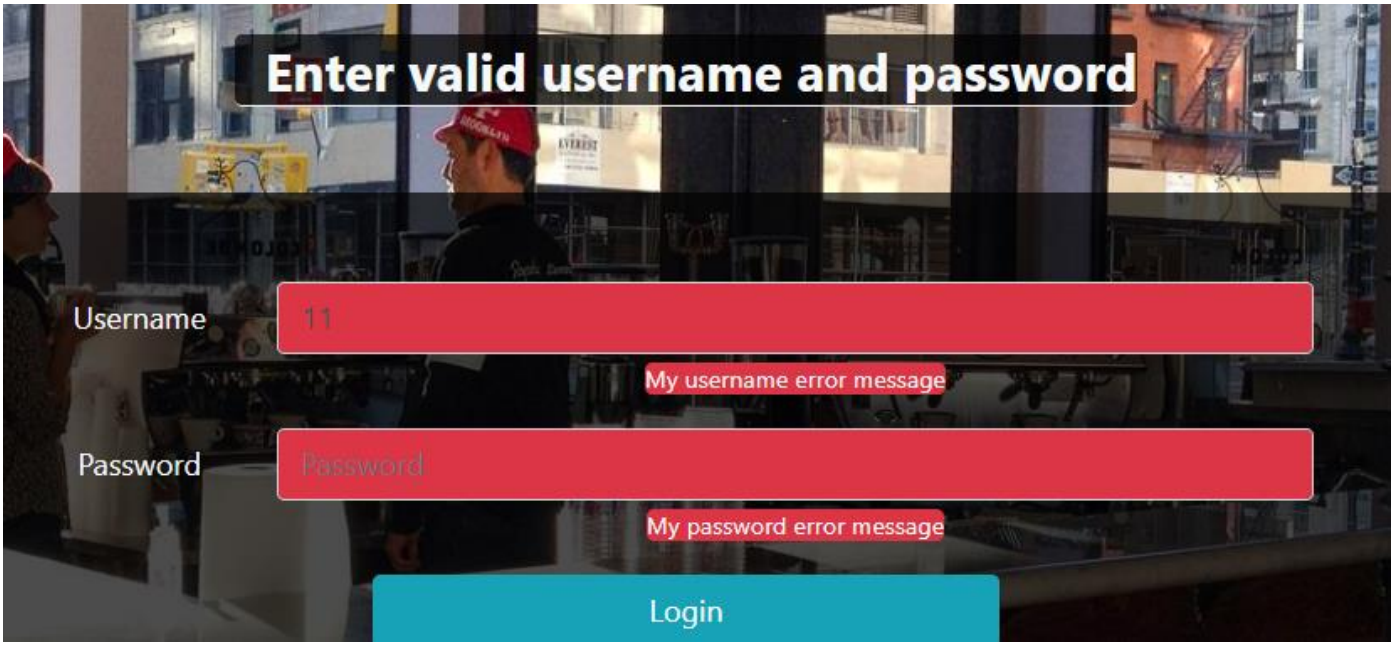
login.html – листване на грешките отгоре

```
<div th:each="e : ${#fields.errors()}" th:text="${e}"></div>
```

UserLoginBindingModelDTO.java

```
public class UserLoginBindingModelDTO {
    @Size(min = 5, max = 20, message = "My username error message")
    private String username;

    @Size(min = 3, message = "My password error message")
    private String password;

    public UserLoginBindingModelDTO() {
    }
```

The display result



Custom Annotations

- You can also implement **custom validation annotations**
  - Sometimes it is necessary due to complex validation functionality

Пример 1 – проверка сега или в бъдещето

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
@Constraint(validatedBy = PresentOrFutureValidator.class)
public @interface PresentOrFuture {

    String message() default "Invalid Date";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}
```

- You can also use the **@PresentOrFuture** validation annotations.

```
public class SomeModel {
    @NotNull
    @PresentOrFuture
    @DateTimeFormat(pattern = "dd/MM/yyyy")
@DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm")  за LocalDateTime
     private Date startDate;
}
```

- You will have to implement a **custom validator** too

```
public class PresentOrFutureValidator implements ConstraintValidator<PresentOrFuture, Date> {
    @Override
    public boolean isValid(Date date,
                           ConstraintValidatorContext constraintValidatorContext) {
```

```
        Date today = new Date();
        return date.after(today);   //True = No Error; False = Error

    }
}
```

Пример 2 – проверка на e-mail

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
@Constraint(validatedBy = UniqueUserEmailValidator.class)
public @interface UniqueUserEmail {
    String message() default "Invalid Email";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}


public class UniqueUserEmailValidator implements ConstraintValidator<UniqueUserEmail, String> {
    private UserRepository userRepository;

    public UniqueUserEmailValidator(UserRepository userRepository) {
        this.userRepository = userRepository;

    }

    //    @Override
//    public void initialize(UniqueUserName constraintAnnotation) {
//        ConstraintValidator.super.initialize(constraintAnnotation);
//    }

    @Override
    public boolean isValid(String value, ConstraintValidatorContext context) {
        return this.userRepository.findByEmail(value).isEmpty();
    }
}


public class UserRegisterDto {
    @NotEmpty(message = "User email should be provided") //we override here the default error
message
    @Email(message = "User email should be valid") //we override here the default error message
    @UniqueUserEmail(message = "User email should be unique")  //we override here the default
error message
    private String email;
```

Render errors

```
<form th:action="@{/users/register}"
      th:method="post"
      th:object="${userModel}">

<div class="form-group col-md-6 mb-3">
    <label for="email" class="text-white font-weight-bold">E-mail</label>
    <input id="email"
           th:field="*{email}"
```

```
            th:errorclass="is-invalid"
            type="text"
            class="form-control"
            placeholder="email"/>
```

B)

```
//валидира се от сложната custom анотация за паролите – работи само с таг div и дълбоко съобщение за грешка - колекции от грешки
<div class="invalid-feedback errors alert alert-danger">
       <  div th:each="err : ${#fields.errors('email')}" th:text="${err}"  /> искаме листа от всички грешки
    </div>
</div>
```

C)

Още един вариант за изкарване на дълбоко съобщение за грешка – колекции от грешки

```
<div class="form-group row">
    <label for="email" class="col-sm-2 col-form-label">Email</label>
    <div class="col-sm-10">
        <input type="text" class="form-control" id="email"
               th:field="*{email}"
               th:errorclass="bg-danger"
               aria-describedby="emailHelpInline" placeholder="email@example.com">
        <small id="emailHelpInline"
               th:if="${#fields.hasErrors('email')}"    //проверява за грешки
               th:errorclass="is-valid"   - можем и без този ред
               th:error ="*{email}"                    //показва всичките грешки
               class="bg-danger text-light rounded">
            Enter valid email address.
        </small>
    </div>
</div>
```

Пример 3 – сложна проверка на password и confirmPassword

Render errors

```
<form th:action="@{/users/register}"
      th:method="post"
      th:object="${userModel}">


<div class="form-group col-md-6 mb-3">
    <label for="password" class="text-white font-weight-bold">Password</label>
    <input id="password"
           th:field="*{password}"
           th:errorclass="is-invalid"
           type="password"
           class="form-control"
           placeholder="Password"/>
    <p class="invalid-feedback errors alert alert-danger">
        Password is required and should be at least 5 symbols long.    //валидира се от @Valid анотацията, и @Size(min = 5)
    </p>
</div>

<div class="form-group col-md-6 mb-3">
```

```
    <label for="confirmPassword" class="text-white font-weight-bold">Confirm Password</label>
    <input
                type="password"
                id="confirmPassword"
           name="confirmPassword"
           th:field="*{confirmPassword}"
           th:errorclass="is-invalid"

           class="form-control"
           placeholder="confirmPassword"/>
```

A)
Съобщение за грешка спрямо какво сме записали в HTML елемента p

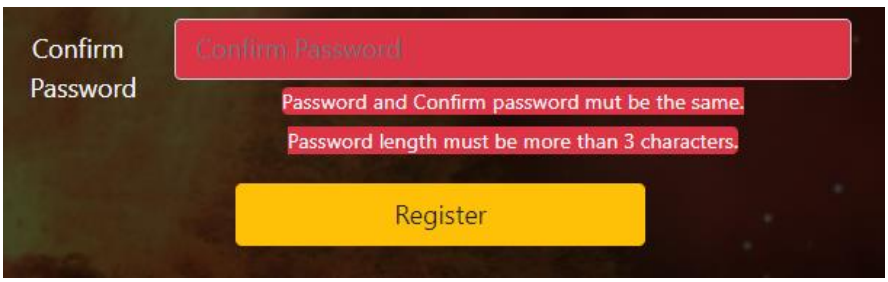
```
    <p class="invalid-feedback errors alert alert-danger">
        Passwords should match.   //валидира се от сложната custom анотация за паролите – с таг елемент p и без дълбоко съобщение за грешка/грешки
    </p>
```

B)
С използване на message-а от клас Анотацията @FieldMatch на UserRegisterDto

```
<div class="invalid-feedback errors alert alert-danger">
    <div th:each="err : ${#fields.errors('confirmPassword')}" th:text="${err}"/> //валидира се от сложната custom анотация за паролите – работи само с таг div и дълбоко съобщение за грешка - колекции от грешки   искаме листа от всички грешки

</div>
```

C)
Още един вариант за изкарване на дълбоко съобщение за грешка – колекции от грешки

```
<div class="col-sm-10">
    <input type="password"
           th:field="*{confirmPassword}"
           th:errorclass="bg-danger"
           class="form-control" id="confirmPassword"
           aria-describedby="confirmPasswordHelpInline" placeholder="Confirm Password">
    <small
            th:if="${#fields.hasErrors('confirmPassword')}"
            th:errorclass="is-valid" – можем и без този ред
            th:errors="*{confirmPassword}"          //показва всичките грешки
           id="confirmPasswordHelpInline" class="bg-danger text-light rounded">
        Password length must be more than 3 characters long.
    </small>
</div>
</div>
```



```java
@FieldMatch(firstField = "password",
        secondField = "confirmPassword",
        message = "Passwords do not match" //we override here the default error message
)
public class UserRegisterDto {
    @NotEmpty
    @Size(min = 5)
    private String password;

    private String confirmPassword;
}

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Constraint(validatedBy = FieldMatchValidator.class)
public @interface FieldMatch {
    String firstField();
    String secondField();

    String message() default "Some default message about password";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}


import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class FieldMatchValidator implements ConstraintValidator<FieldMatch, Object> {
    private String first;
    private String second;
    private String message;

    @Override
    public void initialize(FieldMatch constraintAnnotation) {
        this.first = constraintAnnotation.firstField();
        this.second = constraintAnnotation.secondField();
        this.message = constraintAnnotation.message();
    }

    @Override
    public boolean isValid(Object value, ConstraintValidatorContext context) {
        //В случая Object value ще е целия модел UserRegisterDTO
        BeanWrapper beanWrapper = PropertyAccessorFactory.forBeanPropertyAccess(value);
        Object firstValue = beanWrapper.getPropertyValue(this.first);
        Object secondValue = beanWrapper.getPropertyValue(this.second);

        boolean valid;

        if (firstValue == null) { //ако password от html формуляра е null
            //ако първото е null и второто е null, то не хвърляй грешка
            //ако първото е null а второто има стойност, то хвърли грешка
            valid = secondValue == null;
        } else { //има въведен password от html формуляра
            valid = firstValue.equals(secondValue);  //еднакви ли са password и confirmPassword
```

```
от html формуляра
        }

        if (!valid) {  //ако не са еднакви паролите, върни error message
            context
                    .buildConstraintViolationWithTemplate(message)  //error message-а от клас
Анотацията @FieldMatch на UserRegisterDto
                    .addPropertyNode(this.second) //задай грешката на второто поле от класа
UserRegisterDto
                    .addConstraintViolation()
                    .disableDefaultConstraintViolation();  //без default message при грешка
        }

        return valid;
    }
}
```

7.3. Mapstruct library (alternative to ModelMapper)

Готиното е, че MapStruct не ползва Reflection, И затова е по-бърз от ModelMapper.
https://mapstruct.org/

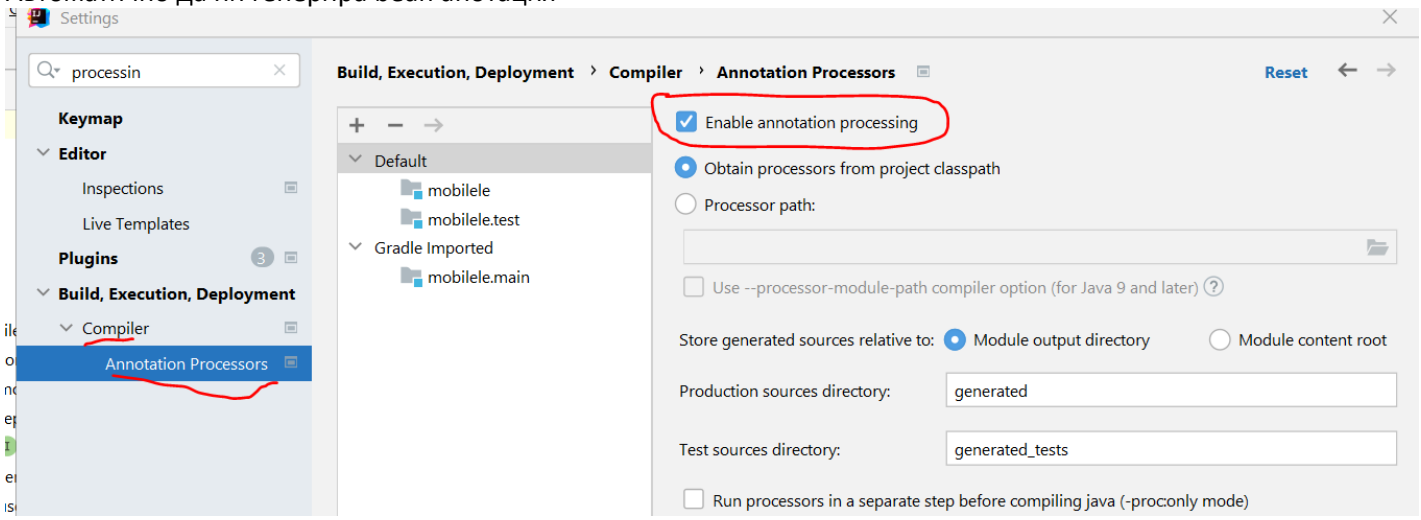
Download

Apache Maven
……..

Gradle
build.gradle

```
dependencies {
    implementation 'org.mapstruct:mapstruct:1.5.1.Final'
        #compileOnly 'org.mapstruct:mapstruct-processor:1.5.1.Final'
    annotationProcessor 'org.mapstruct:mapstruct-processor:1.5.1.Final'
```

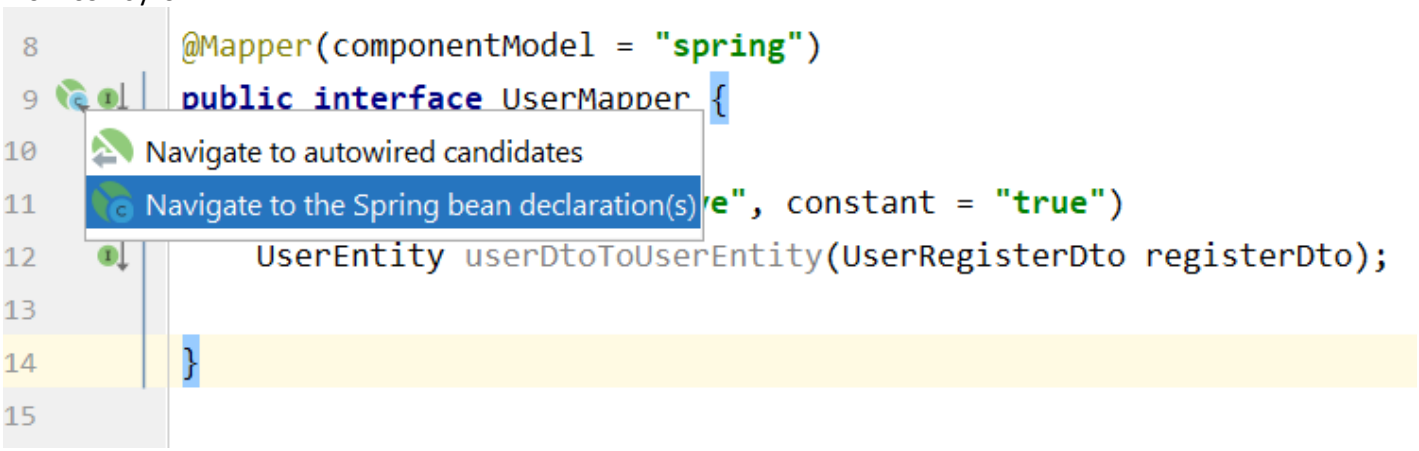
Автоматично да ни генерира bean анотация



Пишем следния interface

```
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;

@Mapper(componentModel = "spring")  //може да бъде expose-ната като Spring bean
public interface UserMapper {

    @Mapping(target = "active", constant = "true")
    UserEntity userDtoToUserEntity(UserRegisterDto registerDto);

}
```

Даваме Build project,
и от този бутон



може да ни се генерира следната имплементация за bean

UserMapperImpl.java

```
import javax.annotation.processing.Generated;
import org.springframework.stereotype.Component;

@Generated(
    value = "org.mapstruct.ap.MappingProcessor",
    date = "2022-06-13T20:05:09+0300",
    comments = "version: 1.5.1.Final, compiler: IncrementalProcessingEnvironment from gradle-
language-java-7.4.1.jar, environment: Java 17.0.2 (Oracle Corporation)"
)
@Component
public class UserMapperImpl implements UserMapper {

    @Override
    public UserEntity userDtoToUserEntity(UserRegisterDto registerDto) {
        if ( registerDto == null ) {
            return null;
        }

        UserEntity userEntity = new UserEntity();

        userEntity.setEmail( registerDto.getEmail() );
        userEntity.setPassword( registerDto.getPassword() );
        userEntity.setFirstName( registerDto.getFirstName() );
        userEntity.setLastName( registerDto.getLastName() );
```

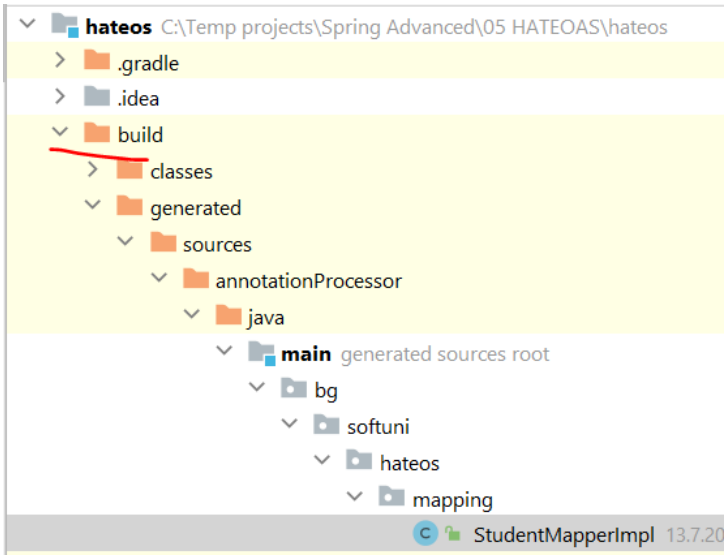
```
        userEntity.setActive( true );

        return userEntity;
    }
}
```

Реално работим само с интерфейси!!!
И да речем го използваме в UserService.java класа

```
@Service
public class UserService {
    private UserRepository userRepository;
    private CurrentUser currentUser;
    private PasswordEncoder passwordEncoder;
    private UserMapper userMapper;
    private Logger LOGGER = LoggerFactory.getLogger(UserService.class);

    public UserService(UserRepository userRepository,
                       CurrentUser currentUser,
                       PasswordEncoder passwordEncoder,
                       UserMapper userMapper) {
        this.userRepository = userRepository;
        this.currentUser = currentUser;
        this.passwordEncoder = passwordEncoder;
        this.userMapper = userMapper;
    }

    public void registerAndLogin(UserRegisterDto userRegisterDto) {
        UserEntity newUser = userMapper.userDtoToUserEntity(userRegisterDto);
        newUser.setPassword(passwordEncoder.encode(userRegisterDto.getPassword()));

//        UserEntity newUser = new UserEntity()
//                .setActive(true)
//                .setEmail(userRegisterDto.getEmail())
//                .setFirstName(userRegisterDto.getFirstName())
//                .setLastName(userRegisterDto.getLastName())
//                .setPassword(passwordEncoder.encode(userRegisterDto.getPassword()));

        newUser = userRepository.save(newUser);

        login(newUser);
    }
}
```

Можем да променяме името на даден field – в обекта/инстанцията car, която е source, имаме **"numberOfSeats"**, а в target-а можем да променим полето на **"seatCount"**

```
@Mapper
public interface CarMapper {

    CarMapper INSTANCE = Mappers.getMapper( CarMapper.class ); 3

    @Mapping(source = "numberOfSeats", target = "seatCount")
    CarDto carToCarDto(Car car); 2
}
```



declare/implement a mapping method

```
//using default mapper method
default String map(PictureEntity photo){
    return photo.getUrl();
}
```

```
//може по няколко мапинги да правим
@Mapping(source = "ivan", target = "petkan")
@Mapping(source = "photo", target = "photoUrl")
ComputerViewGeneralModel computerEntityToComputerSalesViewGeneralModel(ComputerEntity
computerEntity);
```

7.4. HTML Forms requests with Thymeleaf and Spring

Чистите HTML форми могат да имат само method GET и POST!!!

GET

Два варианта за препратка от HTML

```
<div class="card-body">
    <div class="row">
        <a class="btn btn-link" th:href="@{/offers/{id}/edit(id=*{id})}">Update</a>

        <form th:action="@{/offers/{id}(id=*{id})}"
              th:method="delete">
            <input type="submit" class="btn btn-link" value="Delete"></input>
        </form>
    </div>
</div>
```

POST

```
<form
        th:action="@{/offers/add}"
        th:method="POST"
        th:object="${addOfferModel}"
        class="main-form mx-auto col-md-8 d-flex flex-column justify-content-center">
```

```
@PostMapping("/offers/add")
public String addOffer(@Valid AddOfferDTO addOfferModel,
                       BindingResult bindingResult,
                       RedirectAttributes redirectAttributes){

    if (bindingResult.hasErrors()) {
        redirectAttributes.addFlashAttribute("addOfferModel", addOfferModel);

redirectAttributes.addFlashAttribute("org.springframework.validation.BindingResult.addOfferModel
", bindingResult);
        return "redirect:/offers/add";
    }

    //Save in the DB
    this.offerService.addOffer(addOfferModel);

    return "redirect:/offers/all";
}
```

```
    public void addOffer(AddOfferDTO addOfferDTO) {
        OfferEntity newOffer = this.offerMapper.addOfferDtoToOfferEntity(addOfferDTO);

        //TODO - current user should be logged in

//        UserEntity seller = userRepository.findByEmail(currentUser.getEmail()).orElseThrow();
        Optional<UserEntity> seller = userRepository.findById(1L);
        ModelEntity model = modelRepository.findById(addOfferDTO.getModelId()).orElseThrow();

        newOffer.setModel(model);
        newOffer.setSeller(seller.get());

        offerRepository.save(newOffer);
    }
```

DELETE

Настройка в application.yml

```
spring:
  datasource:
    driverClassName: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://localhost:3306/mobilele?allowPublicKeyRetrieval=true&useSSL=false&createDatabaseIfN
otExist=true&serverTimezone=UTC
    username: root
    password:
  mvc:
```

```
    hiddenmethod:
      filter:
        enabled: true
```

Какво ние пишем

```
<form th:action="@{/offers/{id}(id=*{id})}"
      th:method="delete">
    <input type="submit" class="btn btn-link" value="Delete"></input>
</form>
```

```
@Controller
public class OfferController {
    private final OfferService offerService;
    private final BrandService brandService;

    public OfferController(OfferService offerService, BrandService brandService) {
        this.offerService = offerService;
        this.brandService = brandService;
    }

    @GetMapping("/offers/{id}/details")
    public String showOfferDetail(@PathVariable Long id, Model model){
        model.addAttribute("currOfferDetail", offerService.findById(id));

        return "details";
    }

    @DeleteMapping("/offers/{id}")
    public String deleteOffer(@PathVariable Long id){
        offerService.deleteOffer(id);

        return "redirect:/offers/all";
    }

}
```

```
@Service
public class OfferService {
    private final OfferRepository offerRepository;
    private final OfferMapper offerMapper;
    private final ModelRepository modelRepository;
    private final UserRepository userRepository;
    private final CurrentUser currentUser;

    public OfferService(OfferRepository offerRepository, OfferMapper offerMapper,
ModelRepository modelRepository,
                        UserRepository userRepository, CurrentUser currentUser) {
        this.offerRepository = offerRepository;
        this.offerMapper = offerMapper;
        this.modelRepository = modelRepository;
        this.userRepository = userRepository;
        this.currentUser = currentUser;
    }
```

```
    public void deleteOffer(Long id){
        offerRepository.deleteById(id);
    }

}
```

Какво излиза като HTML код

```
<form action="/offers/1" method="post">
    <input type="hidden" name="_method" value="delete">
    <input type="submit" class="btn btn-link" value="Delete">
</form>
```

PUT – целия запис го изтриваме и записваме нов на негово място

Override-ваме всичко. Номерът на записа в базата се запазва, но ако подадем по-малко неща за записа, за тези полета, за които не сме дали инфо ще се запишат като null в базата данни.

PATCH – частичен Update

В случая не редактираме марка и модел на колата само…
Подход с едно ДТО за валидация на данни и едно DTO за презаписване на данните

Настройка в application.yml

```
spring:
  datasource:
    driverClassName: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://localhost:3306/mobilele?allowPublicKeyRetrieval=true&useSSL=false&createDatabaseIfN
otExist=true&serverTimezone=UTC
    username: root
    password:
  mvc:
    hiddenmethod:
      filter:
        enabled: true
```

Какво ние пишем

```
<div class="container">
    <h2 class="text-center text-white">Update Offer</h2>
    <form
            th:object="${offerModel}"
            th:action="@{/offers/{id}/edit(id=*{id})}"
            th:method="PATCH"
            class="main-form mx-auto col-md-8 d-flex flex-column justify-content-center">
        <div class="row">
            <div class="form-group col-md-6 mb-3">
                <label for="mileage" class="text-white font-weight-bold">Mileage</label>
                <input
                        id="mileage"
                        th:field="*{mileage}"
```

```
                        th:errorclass="is-invalid"
                        type="number"
                        min="0" max="900000" step="1000"
                        class="form-control"
                        placeholder="Mileage in kilometers"/>
                <p class="invalid-feedback errors alert alert-danger">
                    Mileage in kilometers is required.
                </p>
            </div>

            <div class="form-group col-md-6 mb-3">
                <label for="price" class="text-white font-weight-bold">Price</label>
                <input id="price"
                       th:field="*{price}"
                       th:errorclass="is-invalid"
                       type="number"
                       min="0" step="100" class="form-control"
                       placeholder="Suggested price"/>
                <p class="invalid-feedback errors alert alert-danger">
                    Suggested price is required.
                </p>
            </div>
        </div>

        <div class="row">
            <div class="form-group col-md-6 mb-3">
                <label class="text-center text-white font-weight-bold"
for="engine">Engine</label>
                <select id="engine"
                        name="engine"
                        th:errorclass="is-invalid"
                        class="form-control">
                    <option value="">- Select engine type -</option>
                    <!--                    We pre-select here what is saved until now - на
BindingDTO модела полето сравняваме дали е равно с текущото поле ${anEngine} *{engine}-->
                    <option th:each="anEngine : ${engines}"
                            th:value="${anEngine}"
                            th:text="${anEngine}"
                            th:selected="${anEngine} == *{engine}">- Select engine type -
                    </option>
                </select>
                <p class="invalid-feedback errors alert alert-danger">
                    Engine type is required.
                </p>
            </div>

            <div class="form-group col-md-6 mb-3">
                <label class="text-center text-white font-weight-bold"
for="transmission">Transmission</label>
                <select id="transmission"
                        name="transmission"
                        th:errorclass="is-invalid"
                        class="form-control">
                    <option value="">- Select transmission type -</option>
                    <!--                    We pre-select here what is saved until now - на
BindingDTO модела полето сравняваме дали е равно с текущото поле *{transmission}-->
                    <option th:each="aTransmission : ${transmissions}"
```

```
                            th:value="${aTransmission}"
                            th:text="${aTransmission}"
                            th:selected="${aTransmission} == *{transmission}">- Select engine
type -
                        </option>
                    </select>
                    <p class="invalid-feedback errors alert alert-danger">
                        Transmission type is required.
                    </p>
                </div>
            </div>

            <div class="row">
                <div class="form-group col-md-6 mb-3">
                    <label for="year" class="text-white font-weight-bold">Year</label>
                    <input id="year"
                           th:field="*{year}"
                           th:errorclass="is-invalid"
                           type="number"
                           min="1900" max="2099" step="1"
                           class="form-control"
                           placeholder="Manufacturing year"/>
                    <p class="invalid-feedback errors alert alert-danger">
                        Manufacturing year is required.
                    </p>
                </div>
            </div>

            <div class="form-group">
                <label class="text-white font-weight-bold" for="description">Description</label>
                <textarea id="description"
                          th:field="*{description}"
                          th:errorclass="is-invalid"
                          type="textarea"
                          class="form-control" rows="3"
                          placeholder="Description"></textarea>
                <p class="invalid-feedback errors alert alert-danger">
                    Description is required.
                </p>
            </div>
            <div class="form-group">
                <label class="text-white font-weight-bold" for="imageUrl">Image URL</label>
                <input id="imageUrl"
                       th:field="*{imageUrl}"
                       th:errorclass="is-invalid"
                       type="url"
                       class="form-control"
                       placeholder="Put vehicle image URL here">
                <p class="invalid-feedback errors alert alert-danger">
                    Vehicle image URL is required.
                </p>
            </div>

            <div class="row">
                <div class="col col-md-4">
                    <div class="button-holder d-flex">
                        <input type="submit" class="btn btn-info btn-lg" value="Update Offer"/>
```

```
                </div>
            </div>
        </div>
    </form>
</div>
```

```
@Controller
public class OfferController {
    private final OfferService offerService;
    private final BrandService brandService;
    private final ModelMapper modelMapper;

    public OfferController(OfferService offerService, BrandService brandService, ModelMapper
modelMapper) {
        this.offerService = offerService;
        this.brandService = brandService;
        this.modelMapper = modelMapper;
    }

    @GetMapping("/offers/{id}/edit")
    public String editOffer(@PathVariable Long id, Model model) {
        OfferDetailsViewDTO offerDetailsView = offerService.findById(id);
        OfferUpdateBindingFlashAttrModelDTO offerUpdateModelDTO =
modelMapper.map(offerDetailsView,
                OfferUpdateBindingFlashAttrModelDTO.class);

        model.addAttribute("engines", EngineEnum.values());
        model.addAttribute("transmissions", TransmissionEnum.values());
        model.addAttribute("offerModel", offerUpdateModelDTO);

        return "update";
    }

    @GetMapping("/offers/{id}/edit/errors") //за да не се пренамажат грешките добавяме /errors
    public String editOfferErrors(@PathVariable Long id, Model model) {
        model.addAttribute("engines", EngineEnum.values());
        model.addAttribute("transmissions", TransmissionEnum.values());

        return "update";
    }

    @PatchMapping("/offers/{id}/edit")
    public String editOffer(@PathVariable Long id,
                            @Valid OfferUpdateBindingFlashAttrModelDTO offerModel,
                            BindingResult bindingResult,
                            RedirectAttributes redirectAttributes) {

        //TODO validation of OfferUpdateBindingFlashAttrModelDTO object
Тук можем сега реално и redirectAttributes и AddFlashAttribute да използваме – когато
валидацията е грешна, то да помни какво потребителят е въвел до момента.

        //validation of OfferUpdateBindingFlashAttrModelDTO object
        if (bindingResult.hasErrors()) {
redirectAttributes.addFlashAttribute("offerModel", offerModel);

redirectAttributes.addFlashAttribute("org.springframework.validation.BindingResult.offerModel",
```

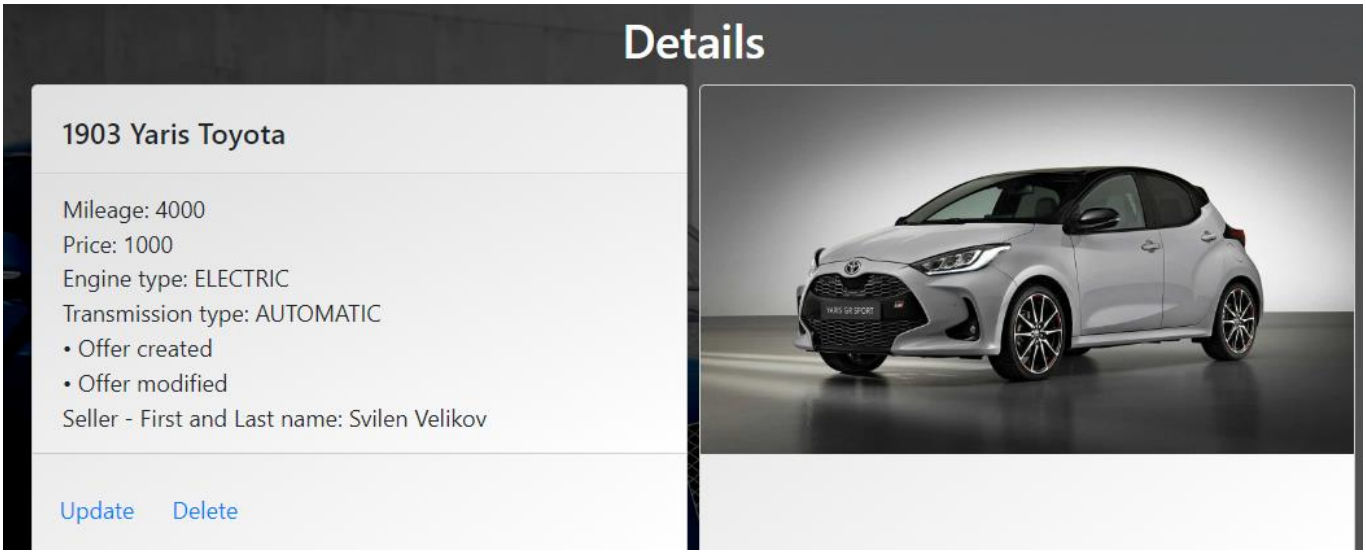
```
bindingResult);
            return "redirect:/offers/" + id + "/edit/errors"; //за да не се пренамажат грешките
добавяме наклонена /errors
        } else {
            // Валидирания модел инстанция на OfferUpdateBindingFlashAttrModelDTO  го записваме
като инстанция на OfferUpdateModelDTO
            OfferUpdateModelDTO modelUpdateService = modelMapper.map(offerModel,
OfferUpdateModelDTO.class);
            modelUpdateService.setId(id);
            offerService.updateOffer(modelUpdateService);

            return "redirect:/offers/" + id + "/details";
        }
    }
}
```

```
@Service
public class OfferService {
public void updateOffer(OfferUpdateModelDTO offerModel){
    OfferEntity offerEntity = offerRepository.findById(offerModel.getId()).orElseThrow(() ->
            new ObjectNotFoundException("Offer with id " + offerModel.getId() + " not found!"));

offerEntity.setPrice(offerModel.getPrice())
        .setDescription(offerModel.getDescription())
        .setEngine(offerModel.getEngine())
        .setImageUrl(offerModel.getImageUrl())
        .setMileage(offerModel.getMileage())
        .setTransmission(offerModel.getTransmission())
        .setYear(offerModel.getYear());

    offerRepository.save(offerEntity);
}
}
```

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class ObjectNotFoundException extends RuntimeException{
    public ObjectNotFoundException(String message) {
        super(message);
    }
}
```

Какво излиза като HTML код

```
<form action="/offers/2/edit" method="post"
class="main-form mx-auto col-md-8 d-flex flex-column justify-content-center">
<input type="hidden" name="_method" value="PATCH">
```



7.5. PasswordEncoder

build.gradle

```
dependencies {
    implementation 'org.springframework.security:spring-security-crypto:5.5.2'
```

pom.xml

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
```

```
@Configuration
public class ApplicationBeanConfiguration {

    @Bean
    public PasswordEncoder passwordEncoder(){
        return new Pbkdf2PasswordEncoder();
    }

    @Bean
    public ModelMapper modelMapper(){
        return new ModelMapper();
    }

}
```

8. Bootstrap, Front-End Basics

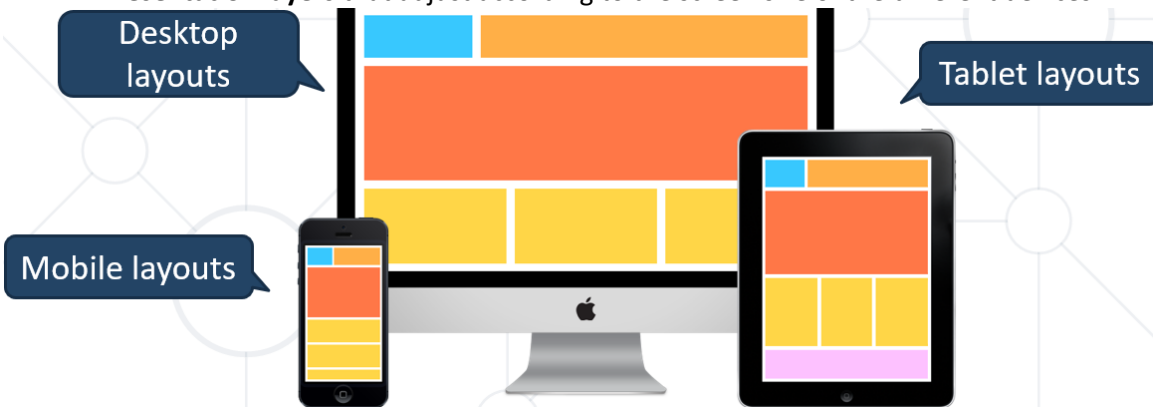
8.1. JavaScript

- **Undefined** - automatically assigned to variables
- **Null** - represents the **intentional absence** of any object value

8.2. Bootstrap

What is a Responsive Design?

- **Presentation layers** that adjust according to the screen size of the different devices



Bootstrap

- World's most popular front-end **component library**
- Open source toolkit for developing with **HTML**, **CSS**, and **JS**
- Works with
  - Responsive **grid system**
  - Extensive prebuilt **components**
  - Powerful plugins built on jQuery

Колекция от CSS

Include from a BootstrapCDN – JS

- Be sure to place **jQuery** and **Popper** first, as the Bootstrap code depends on them

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <!-- Първо зареждам Bootstrap CSS-а -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/css/bootstrap.min.css"
        rel="stylesheet"
        integrity="sha384-0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVor"
        crossorigin="anonymous" />
```

```
    <!-- След това зареждаме нашия CSS, за да override-нем-->
    <link rel="stylesheet" href="styles.css" />


</head>

<body>
```

// когато нямаме defer и async – то трябва да сложим в края на <body> то.

```
    <!-- Първо зареждаме JQuery и Popper, и последно Boostrap скрипта.
        Bootstrap скрипта ще чете от предходните 2 библиотеки -->
    <script src="https://code.jquery.com/jquery3.3.1.slim.min.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.5/dist/umd/popper.min.js"
    integrity="sha384-Xe+8cL9oJa6tN/veChSP7q+mnSPaj5Bcu9mPX5F5xIGE0DVittaqT5lorf0EI7Vk"
    crossorigin="anonymous"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-pprn3073KE6tl6bjs2QrFaJGz5/SUsLqktiwsUTF55Jfv3qYSDhgCecCxMW52nD2"
    crossorigin="anonymous"></script>

    <!-- И чак накрая нашия скрипт - каквото ние пишем това да стане, ако има съвпадение с
Bootstrap, да се изпълнява нашето -->
    <script src="js/main.js"></script>
</body>

</html>
```

```
<script src="demo_defer.js" defer></script>
<script src="demo_async.js" async></script>
```

The defer attribute is a boolean attribute.

If the defer attribute is set, it specifies that the script is downloaded in parallel to parsing the page, and executed after the page has finished parsing. Ако го има този атрибут то дори да го сложим в head-а, пак браузъра ще зареди този скрипт накрая.

Note: The defer attribute is only for external scripts (should only be used if the src attribute is present).

Note: There are several ways an external script can be executed:

- If `async` is present: The script is downloaded in parallel to parsing the page, and executed as soon as it is available (before parsing completes)
- If `defer` is present (and not `async`): The script is downloaded in parallel to parsing the page, and executed after the page has finished parsing
- If neither `async` or `defer` is present: The script is downloaded and executed immediately, blocking parsing until the script is completed – в този случай скрипта трябва да е сложен в края на <body> то.

https://getbootstrap.com/ - интегриран с JS
Минуси: само с div са дадени примерите.

https://tailwindcss.com/ - само надгражда CSS – да пишем CSS по различен начин

JQuery все по-малко се използва, защото все повече се ползват React, Vue, Angular. Някои неща в JQuery са тъпо написани, някои неща в React, Vue и Angular вземат/копират JQuery.

8.3. Bootstrap Grid System

Build Layouts with Grid – Twelve Column System

Bootstrap Grid System Demo

Use our powerful mobile-first flexbox grid to build layouts of all shapes and sizes thanks to a twelve column system, six default responsive tiers, Sass variables and mixins, and dozens of predefined classes.

```
    <div class="container">
        <div class="row">
            <div class="col-sm-6 col-md-4">Column one</div>  <!—6 колони при small, 4 колони при md т.е. за два екрана го правим сега, но реално автоматично работи за всеки екран ☺ -->
            <div class=" col-sm-6 col-md-4">Column two</div>
            <div class="col-xs m-3">Column three</div>
        </div>
    </div>
```

Bootstrap Containers

- Rows must be placed in **containers**
  - **.container** has one fixed width for each screen size in bootstrap (**xs**, **sm**, **md**, **lg**)
  - **.container-fluid** expands to fill the available width

https://getbootstrap.com/docs/5.2/layout/grid/

Column Classes

Extra small (xs)
Small (sm)
Medium (md)
Large (lg)
Extra large (xl)
Extra extra large (xxl)

- **.col-xs**: width less than 576px
- **.col-sm**: width between 576px and 768px
- **.col-md**: width between 768px and 992px
- **.col-lg**: width over 992px

	xs <576px	**sm** ≥576px	**md** ≥768px	**lg** ≥992px	**xl** ≥1200px	**xxl** ≥1400px
Container `max-width`	None (auto)	540px	720px	960px	1140px	1320px
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`	`.col-xxl-`
# of columns	12					
Gutter width	1.5rem (.75rem on left and right)					
Custom gutters	Yes					
Nestable	Yes					
Column ordering	Yes					

Columns and gutters

Gutters е разстоянието между колоните

Color

- Handful of **color utility classes**

```
<p class="text-primary">.text-primary</p>
<p class="text-secondary">.text-secondary</p>
<p class="text-success">.text-success</p>
<p class="text-danger">.text-danger</p>
<p class="text-warning">.text-warning</p>
<p class="text-info">.text-info</p>
<p class="text-light bg-dark">.text-light</p>
<p class="text-dark">.text-dark</p>
<p class="text-muted">.text-muted</p>
<p class="text-white bg-dark">.text-white</p>
```

Background Color

- Easily set the **background** of an element to any contextual **class**

```
<div class="bg-primary text-white">.bg-primary</div>
<div class="bg-secondary text-white">.bg-secondary</div>
<div class="bg-success text-white">.bg-success</div>
<div class="bg-danger text-white">.bg-danger</div>
<div class="bg-warning text-dark">.bg-warning</div>
<div class="bg-info text-white">.bg-info</div>
<div class="bg-light text-dark">.bg-light</div>
<div class="bg-dark text-white">.bg-dark</div>
<div class="bg-white text-dark">.bg-white</div>
```

8.4. CSS Grid System

```
.customGrid {
    max-width: 1140px;
    margin: 0 auto;
    padding: 0 15px;
    display: grid;
    grid-gap: 15px;
}

@media screen and (min-width: 560px){
    .customGrid{
        grid-gap: 15px;
        grid-template-columns: repeat(2, 1fr);
    }
}

@media screen and (min-width: 768px){
    .customGrid{
        grid-template-columns: repeat(3, 1fr);
    }
}
```

8.5. Bootstrap Components

Button Groups

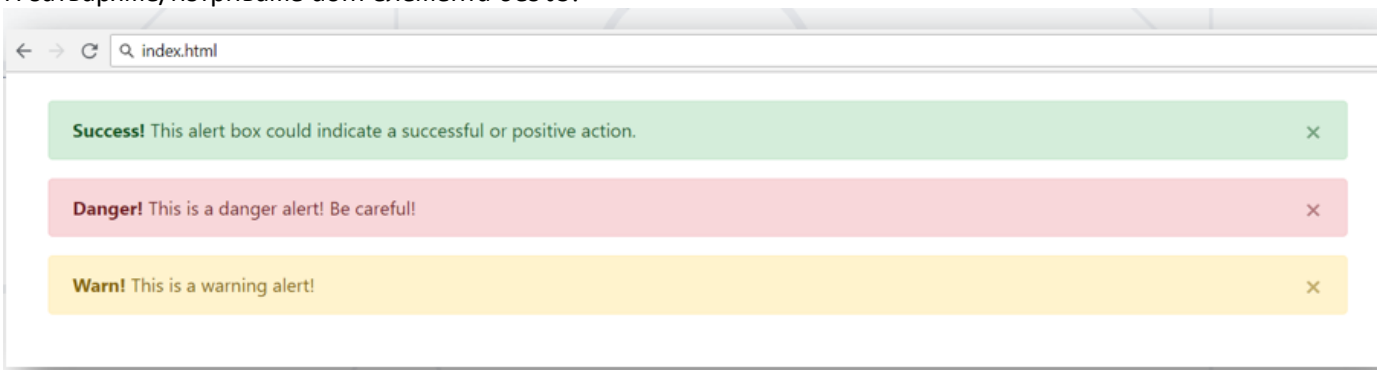
- Custom **button styles** with support for multiple sizes, states, and more


```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
```

Alerts

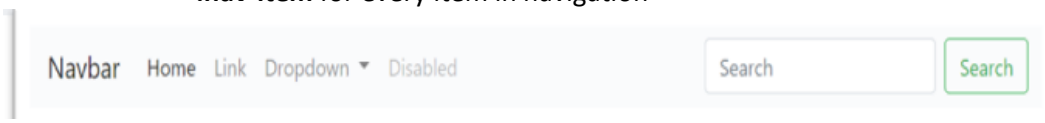
```
<div class="alert alert-success alert-dismissable">
     <a class="close" data-dismiss="alert" aria-label="close">×</a>
     <strong>Success!</strong> This alert box could indicate a successful or positive
action.
</div>
```

И затваряме/изтриваме dom елемента без JS!



Nav and Navbar

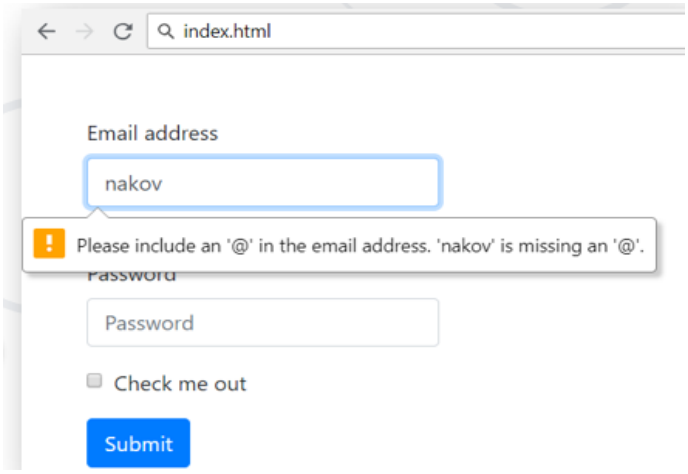
- Require a wrapping **.navbar**
- **Responsive** by default
- Come with built-in support for a handful of **sub-components**
  - **.navbar-brand** for your company, product, or project name
  - **.navbar-nav** for a full-height and lightweight navigation
  - **.nav-item** for every item in navigation



Forms

- Form **control styles**, **layout options** and custom **components** for creating a wide variety of forms
- Use **type** attribute on all inputs to take advantage of newer input controls

- Email **verification**
- Number **selection**



Jumbotron

- **Lightweight**, **flexible** component for showcasing hero unit style content

```
<div class="jumbotron">
  <h1 class="display-4">Hello,
world!</h1>
  <p class="lead">This is a ...</p>
  <hr class="my-4"><p>It uses ...</p>
  <p class="lead">
  <a class="btn btn-primary btn-lg">
  Learn more</a>
  </p>
</div>
```

See more at: https://getbootstrap.com/docs/4.0/components/jumbotron/

Carousel

A slideshow component for cycling through elements—images or slides of text—like a carousel.

```
<div id="carouselExampleCaptions" class="carousel slide" data-bs-ride="false">
  <div class="carousel-indicators">
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="0"
class="active"
      aria-current="true" aria-label="Slide 1"></button>
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="1"
aria-label="Slide 2"></button>
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="2"
aria-label="Slide 3"></button>
  </div>
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
```

```
      <div class="carousel-caption d-none d-md-block">
        <h5>First slide label</h5>
        <p>Some representative placeholder content for the first slide.</p>
      </div>
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
      <div class="carousel-caption d-none d-md-block">
        <h5>Second slide label</h5>
        <p>Some representative placeholder content for the second slide.</p>
      </div>
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
      <div class="carousel-caption d-none d-md-block">
        <h5>Third slide label</h5>
        <p>Some representative placeholder content for the third slide.</p>
      </div>
    </div>
  </div>
  <button class="carousel-control-prev" type="button" data-bs-
target="#carouselExampleCaptions" data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-bs-
target="#carouselExampleCaptions" data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
```

8.6. jQuery

What is jQuery?

Cross-browser JavaScript library

- Dramatically simplifies **DOM manipulation**
- jQuery is **more expressive/declarative and easy to read** than JS pure Vanilla
- Simplifies **AJAX calls** and working with RESTful services
- Free, open-source software: **https://jquery.com**

Web консорциум относно какви неща всеки един браузър трябва да поддържа. При това положение, няма нужда вече от ползването толкова много на jQuery.

```
$('li').css('background', '#DDD');  // Change the CSS for all <li> tags
```

Using jQuery locally

`npm install jquery`

Using jQuery from CDN

CDN:

```
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>

<script src="https://code.jquery.com/jquery-3.1.1.min.js"
integrity="sha256-hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8=
" crossorigin="anonymous"></script>


$(function () {
    $("a").on('click', (event) => {
        alert("Link forbidden!");
        event.preventDefault();
    });
});
```

jQuery demo

page.js

```
$(document).ready(function() {
    $("#my-button").click(function() {
        console.log("Button clicked");

        $("#my-text").text('My custom now text JQuery here');
        $("#my-text").addClass('text-secondary');
    })
});
```

Или така

```
function myCustomClick() {
    console.log("Custom button click");

    $("#my-text").text('My custom now text JQuery here');
    $("#my-text").addClass('text-secondary');
}
```

demoJQuery.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/css/bootstrap.min.css" </head>

<body>
    <button type="button" id="my-button" class="btn btn-primary">Button</button>
или така
    <button type="button" onclick="myCustomClick()" id="my-button" class="btn btn-
primary">Button</button>
    <p id="my-text">My static text here.</p>


    <script src="https://code.jquery.com/jquery-3.6.0.min.js" />
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.5/dist/umd/popper.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/js/bootstrap.bundle.min.js"></script>
    <script src="page.js"></script>

</body>

</html>
```

jQuery Selectors

jQuery selection is the same as the **querySelector** in Vanilla JS

- All selector `$('*') // Selects all elements`
- Class selector `$('.class') // Select by class name`
- Element selector `$('section') // Select by tag name`
- Id selector `$('#id') // Selects a element by given id`
- Multi-selector `$('selector1, selector2') // Combined`

```
$('.list .list-item a');  //<ul class="list">  <li class="list-item"> <a> </a></li></ul>
$('.list.list-item a');  //<ul class="list list-item"></ul>
```

Adding Elements with jQuery

Select the parent element, then use:

- **append() / prepend()**
- **appendTo() / prependTo()**

```
<div id="wrapper">
```

```
    <div>Hello, student!</div>
    <div>Goodbye, student!</div>
</div>

$('#wrapper div').append("<p>It's party time :)</p>");
$('<h1>Greetings</h1>').prependTo('body');
```

```
<h1>Greetings</h1>
<div id="wrapper">
  <div>
    "Hello, student!"
    <p>It's party time :)</p>
  </div>
  <div>
    "Goodbye, student!"
    <p>It's party time :)</p>
  </div>
</div>
```

Creating / Removing Elements

```
let div = $('<div>');
div.text('I am a new div.');
div.css('background', 'blue');
div.css('color', 'white');
$(document.body).append(div);
```

With JS

```
document.getElementById("btn-add")
    .addEventListener('click', () => {
        const listNode = document.createElement('li');
        listNode.innerHTML = 'NEW tehcnology';
        document.getElementById('technologies-list')
            .appendChild(listNode);
    });
```

With jQuery

```
$('#btn-add')
    .on('click', () => {
        $('#technologies-list')
        .append($('<li>New Technology with jQuery</li>'));
    });
```

Removing

```
$('#technologies-list')
    .on('click', 'li', function() {
        $(this).remove();
    });
```

jQuery Events: Attach / Remove

- Attaching events on certain elements

```
$('a.button').on('click', buttonClicked);
function buttonClicked() {
    $('.selected').removeClass('selected');
    $(this).addClass('selected');
    // "this" is the event source (the hyperlink clicked)
}
```

- Removing event handler from certain elements

```
$('a.button').off('click', buttonClicked);
```

//Mouse events

//Form events

//Keayboard events

jQuery Methods

- **text()** - reads and writes text

```
let text = $('#theElement').text();
$('#theElement').text('New text for element.');
```

- **html()** - returns the HTML of a given element

```
let html = $('#theElement').html();
$('#theElement').html('New text for element.');
```

- **val()** - gets and sets value

```
let theValue = $('#theFormField').val();
$('#theFormField').val('New value');
```

- **attr()** - reads and writes attributes of HTML elements. Also can take an object as parameter

```
let attrValue = $('#theFormField').attr('height');
$('#theFormField').attr({height : attrValue});
```

- **removeAttr()** - removes an attribute from an HTML element

```
$('#theFormField').removeAttr('height');
```

- **wrap()** - wraps the selected element in another HTML element

```
$('#someElement').wrap('<div style='border: 1px solid black;'></div>');
```

- **replaceWith()** - replaces the selected HTML element with a new one

```
$('#theElement').replaceWith('<div style='border: 1px solid black;'></div>');
```

- **remove()** - removes the selected HTML element from the DOM

```
$('#theElement').remove();
```

- **empty()** - removes all child elements of the selected HTML element

```
$('#theElement').empty();
```

```
var selectVal = $('option:selected', formSelect).val(); //we get here the selected option from
the <select> of the current form
```

9. Web-API-and-REST-Controllers

9.1. REST API

Изначало HTTP Protocol е бил stateless – backend-server-а не трябва да пази информацията от предходната заявка на предхония или същия клиент. След това, с времето, обаче се променят нещата.

- Statelessness – няма състояние, на практика не е така – има кукита, и .т.н.

REST работи на база на HTTP протокола

Rest работи по Default с данни формат JSON. Може и да е с XML също.
За разлика от HTTP, което ползва както JSON формат, така и други тип данни като html данни, и други.

REST протокола може да обменя информация:
- между 2 сървъра
- или между сървър и клиент (без браузър)
- между сървър и браузър клиент

REST не е различно от MVC (Model-View-Controller), но без view-то примерно.

Versioning in REST API – 1.1. за стари клиенти, и примерно v.2.2. за нови клиенти.

Пример:

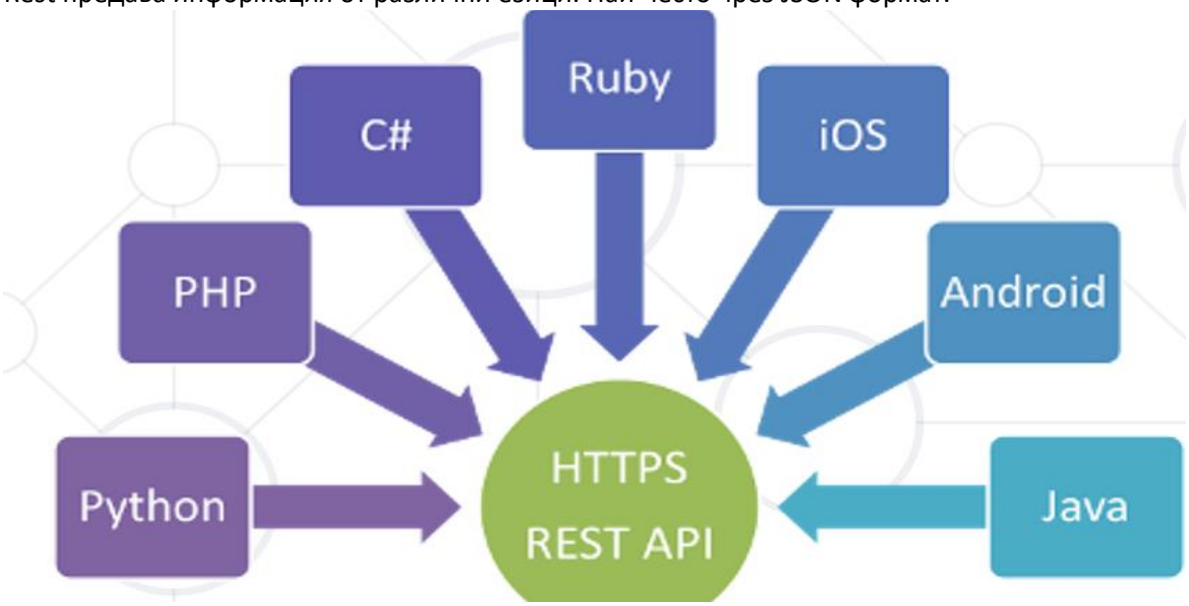
HomeController.java

```
@GetMapping('/')
public ModelAndView index(ModelAndView modelAndView) {
    modelAndView.setViewName('index');
    return modelAndView;
}

@GetMapping(value = '/fetch', produces = 'application/json')
```

```
@GetMapping(value = '/fetch', produces = 'application/xml')
@ResponseBody
public Object fetchData() {
    return new ArrayList<Product>() {{
        add(new Product() {{
            setName('Chewing Gum');
            setPrice(new BigDecimal(1.00));
            setBarcode('133242556222');
        }});
    ...
    }};
}
```

RESTful Design

Rest предава информация от различни езици. Най-често чрез JSON формат.



RESTful API - Representational state transfer API

True RESTful API, is a **web service** that must adhere to the following six **REST architectural constraints**

- Use of a **uniform interface** (UI)
- **Client-server based**
- **Stateless** operations – backend-server-а не помни предишната заявка каква е била и от кого
- RESTful **resource caching**
- **Layered system** – примерно web сървър-а на една машина, базите данни на една машина, аутентикация да се случва на трета машина.
- **Code on demand** – един response на http rest api може да върне код, който да се изпълни

SOAP and RPC

- **Simple Object Access Protocol** (SOAP)
  - Standardized protocol that **sends messages** using other protocols such as **HTTP** and **SMTP**
  - The SOAP specifications are official web standards, maintained and developed by the World Wide Web Consortium (W3C)
- **Remote Procedure Call** (RPC)

- A way to describe a mechanism that lets you **call a procedure in another process** and **exchange data by message passing**

HTTP GET

- Used to retrieve single data entities or data arrays

```
{
'id': 32,
'name': 'Read Book',
'deadline': 1362268800000, 'categoryName': 'Work', 'enabled': false
}

[{
    'id': 32,
    'name': 'Read Book',
    'deadline': 1362268800000,    'categoryName': 'Work',    'enabled': false
    },
    …
]
```

HTTP POST

HTTP PUT

HTTP PATCH

HTTP DELETE

9.2. REST (Representational state transfer) with Spring

Response Body On MVC Controller

- **Returning plain-text** in MVC controller:

```
@GetMapping('/info/{id}')     //MVC controller
@ResponseBody                 //Спри да търсиш Thymeleaf
public String getInfo(@PathVariable Long id) {
...
    return ‘Plain text’;
}
```

```
@GetMapping('/info/{id}')     //MVC controller
@ResponseBody                 //Спри да търсиш Thymeleaf
public Student getInfo(@PathVariable Long id) {
...
    return new Student().setName(“Joro”);
}
```

Response Status

- Setting the correct Response Code

```
@GetMapping('{id}/info')
@ResponseStatus(HttpStatus.OK)
public GameInfoView getInfo(@PathVariable Long id){
    GameInfoView gameInfo = this.gameService.getInfoById(id);

    return new Gson().toJson(gameInfo);
}
```

```
(HttpServletResponse reponse)
response.setStatus(HttpStatus.NotFound)
```

REST Controllers

- **@RestController** is essentially **@Controller** + **@ResponseBody**

```
@RestController /controller + response body/
public class OrderController {
    @GetMapping('{id}/info')
    public ResponseEntity<Game> getGame(@PathVariable Long id) {
        ...
    }
}
```

Response Entity

```
@GetMapping('{id}/title')
public ResponseEntity<Game> getTitle(...){
...
    return new ResponseEntity(gameService.getGame(id));
}
```

- The **ResponseEntity<>** object allows you **to change the response body**, response headers and response code

Spring Data REST

- Maven Dependency

```
Обикновено не се прави така. Понеже се expose-ва всичко.

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

- Spring Data REST **scans your project** and **provides REST API** for your application **using HAL** as media type

Configuring Repositories

По-добре бази данни да вземаме с обикновен @Controller. Голяма боза и допълнителни настройки е ако използваме @RepositoryRestResource

- You can configure repository settings using the **@RepositoryRestResource** annotation:

```
@RepositoryRestResource(path = 'gameIssues')
public interface IssueRepository extends JpaRepository<Issue, Long> {

    Issue getById(@Param('id') Long id);

    List<Issue> getAllByOrderByDateDesc();
}
```

Example 1

```
import bg.softuni.books.model.dto.BookDTO;
import bg.softuni.books.service.BookServiceImpl;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/books")
public class BooksController {
    private BookServiceImpl bookService;

    public BooksController(BookServiceImpl bookService) {
        this.bookService = bookService;
    }

    //called on http://localhost:8080/books
    @GetMapping
    public ResponseEntity<List<BookDTO>> getAllBooks(){
        List<BookDTO> allBooks = this.bookService.getAllBooks();

        return ResponseEntity.ok(allBooks);  //with body allBooks
    }

    @GetMapping("/{id}")
    public ResponseEntity<BookDTO> getBookById(@PathVariable("id") Long id) {
        Optional<BookDTO> bookOpt = this.bookService.getByBookId(id);
        if (bookOpt.isEmpty()) {
            return ResponseEntity.notFound().build();
        } else {
            return ResponseEntity.ok(bookOpt.get());
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<BookDTO> deleteBookById(@PathVariable("id") Long id) {
        bookService
                .deleteBook(id);

        return ResponseEntity.noContent().build();
    }
```

```
    //called on http://localhost:8080/books
//    @PutMapping("")
    @PostMapping("")    //В случая работи за добавяне на нова книга, и за промяна на съществуваща книга в базата  (bookRepository.save() или добавя нова книга или презаписва)
    public ResponseEntity<BookDTO> create(
            @RequestBody BookDTO bookDTO,   //десериализация на body-то до Java обект – пропъртитата на боди-то на нашата заявка ще бъдат популирани върху нашето bookDTO
            UriComponentsBuilder builder
    ) {
        //http://localhost:8080/books/id
        long bookId = bookService.createBook(bookDTO);
        URI location = builder.path("/books/{id}")
                .buildAndExpand(bookId)
                .toUri();

        return ResponseEntity.created(location).build();
    }

ResponseEntity.status(HttpStatus.OK).build();


//called on http://localhost:8080/books
@PutMapping("/{id}")
public ResponseEntity<BookDTO> update(
        @PathVariable("id") long bookId,
        @RequestBody BookDTO bookDTO){
    //todo
    throw new UnsupportedOperationException("coming soon");
}

}
```

9.3. Rest Template not reactive

Rest Template

- Accessing **a third-party REST service** inside a Spring application revolves around the use of the Spring **RestTemplate class**
- Class is **designed to call REST services**
- Its **main methods** are closely tied to **REST's underpinnings**, which are the **HTTP protocol's methods**: **HEAD**, **GET**, **POST**, **PUT**, **DELETE**
- **Recommended** to use the non-blocking, **reactive WebClient**.
- RestTemplate will be **deprecated in a future version**

Spring казват, че в бъдещите версии ще се deprecate-не Rest template, и ще се използва само Reactive WebClient.

Rest Template Demo – когато четем от един port 8080 като работим с втори порт 8000

Задаваме нов порт **8000**, защото TomCat е заел вече port 8080
application.yml

```
server:
  port: 8000
```

```java
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;
@Configuration
public class RestConfig {

    @Bean
    public RestTemplate create(RestTemplateBuilder restTemplateBuilder){
        return restTemplateBuilder.build();
    }
}
```

```java
import bg.softuni.restTemplate.model.dto.BookDTO;
import org.springframework.boot.CommandLineRunner;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
@Component
public class RestTemplateDemo implements CommandLineRunner {
    private RestTemplate restTemplate; //нашия Bean

    public RestTemplateDemo(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @Override
    public void run(String... args) throws Exception {
        ResponseEntity<BookDTO[]> allBooksResponse = restTemplate
                .getForEntity("http://localhost:8080/api/books", BookDTO[].class);

        if (allBooksResponse.hasBody()) {
            for (BookDTO book : allBooksResponse.getBody()) {
                System.out.println("Book: " + book);
            }
        }
    }
}
```

```java
@JsonIgnoreProperties(ignoreUnknown = true) //да игнорира непознато
@JsonInclude(Include.NON_NULL)
public class BookDTO {
    private AuthorDTO author;
    private Long id;
    private String title;
    private String isbn;
```

HTTP GET Method Examples

- **getForObject(url, classType)**
  - Retrieves a **representation by doing a GET on the URL**.
  - The response (if any) is unmarshalled to given class type and returned

- **getForEntity(url, responseType)**
  - Retrieve a **representation as ResponseEntity** by doing a GET on the URL

```
ResponseEntity<BookDTO[]> allBooksResponse = restTemplate
                .getForEntity("http://localhost:8080/api/books", BookDTO[].class);
```

- **exchange(requestEntity, responseType)**
  - **Executes** the specified **request** and **returns** the response as **ResponseEntity**

- **execute(url, httpMethod, requestCallback, responseExtractor)**
  - **Executes the httpMethod** to the given URI template and **preparing the request** with the **RequestCallback**

HTTP POST Method Examples

- **postForObject(url, request, classType)**
  - **POSTs** the given object **to the URL** and **returns the representation** found in the response **as given class type**

- **postForEntity(url, request, responseType)**
  - **POSTs** the given object **to the URL** and **returns the response as ResponseEntity**

```
@Component
public class AppInit implements CommandLineRunner {
    private final RimService rimService;
    private final RestTemplate restTemplate; //нашия Bean


    public AppInit(RimService rimService, RestTemplate restTemplate) {
        this.rimService = rimService;
        this.restTemplate = restTemplate;
    }

    @PostConstruct
    public void beginInit(){
        rimService.init();
    }

    @Override
    public void run(String... args) throws Exception {
        addOneRim();
    }


    private void addOneRim() {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);

//        MultiValueMap<String, String> map= new LinkedMultiValueMap<>();
//        //    {
```

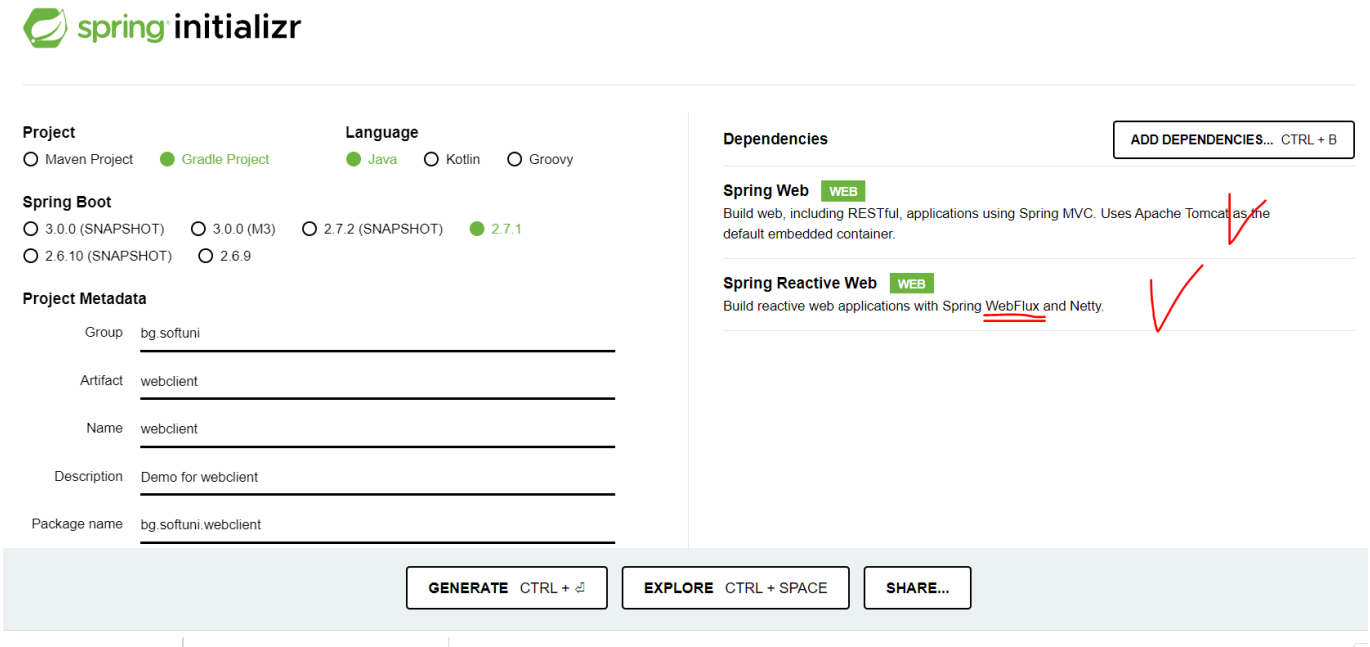
```
//        //        "metalKind": "bronze",
//        //        "inches": "15"
//        //    }
//        map.add("metalKind", "bronze");
//        map.add("inches", "15");

        RimJsonDTO newRimJsonToAdd =
                new RimJsonDTO()
                        .setMetalKind("bronze")
                        .setInches("15");

        RimEntity newRimEntity = new RimEntity()
                .setMetalKind(newRimJsonToAdd.getMetalKind())
                .setInches(newRimJsonToAdd.getInches());

        HttpEntity<RimEntity> request = new HttpEntity<>(newRimEntity, headers);

//caught by the @RestController :)
        ResponseEntity<RimEntity> oneRimPost = restTemplate
                .postForEntity("http://localhost:8000/spareparts/rims", request,
RimEntity.class);

        //If we do not have the @RestController with @PostMapping("/spareparts/rims") in class
RimController,
        // then we can update the db with the below line
//        rimService.addNewRim(newRimJsonToAdd);
    }
```

- **postForLocation(url, request, responseType)**
  - **POSTs** the given object **to the URL** and **returns** the value of the **Location header**
- **exchange(url, requestEntity, responseType)**
- **execute(url, httpMethod, requestCallback, responseExtractor)**

HTTP PUT and HTTP DELETE Method Examples

- **put(url, request)**
  - PUTs the given request object to URL
- **delete(url)**
  - Deletes the resource at the specified URL

9.4. Spring Reactive Webclient as a Rest Template



Spring Reactive е non-blocking - //нишката няма да се задържи и ще бъде release-вана и ше може тя да изчака, за да се expose-не за обслужването на някой друг request.

application.yml

```yaml
server:
  port: 8005
```

Rest Template Reactive Demo + Jackson parser (part of Spring MVC)

https://reqres.in/

GET this: https://reqres.in/api/users/2

```
{
  "data": {
    "id": 2,
    "email": "janet.weaver@reqres.in",
    "first_name": "Janet",
    "last_name": "Weaver",
    "avatar": "https://reqres.in/img/faces/2-image.jpg"
  },
  "support": {
    "url": "https://reqres.in/#support-heading",
    "text": "To keep ReqRes free, contributions towards server costs are appreciated!"
  }
}
```

```java
import bg.softuni.webclient.model.UserDTO;
import org.springframework.boot.CommandLineRunner;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Component;
import org.springframework.web.reactive.function.client.WebClient;

@Component
```

```java
public class WebclientDemo implements CommandLineRunner {
    @Override
    public void run(String... args) {

        WebClient client = WebClient.create("https://reqres.in/");

        UserDTO user = client.get()
                .uri("api/users/2")
                .accept(MediaType.APPLICATION_JSON)
                .retrieve()
//                .bodyToFlux(UserDTO[].class)   - масив от обекти
                .bodyToMono(UserDTO.class)//връща само 1 или 0 обекти
                .block(); //нишката няма да се задържи и ще бъде release-вана и ше може да
изчака, за да се expose-не за обслужването на някой друг request

        System.out.println(user);
    }
}


import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;

@JsonInclude(JsonInclude.Include.NON_NULL)
public class UserDTO {
    @JsonProperty("data") //ако името съвпада, то няма нужда
    private DataDTO data;
    @JsonProperty("support") //ако името съвпада, то няма нужда
    private SupportDTO support;

    public UserDTO() {
    }

    @JsonProperty("data") //ако името съвпада, то няма нужда
    public DataDTO getData() {
        return data;
    }

    @JsonProperty("data") //ако името съвпада, то няма нужда
    public void setData(DataDTO data) {
        this.data = data;
    }

    @JsonProperty("support") //ако името съвпада, то няма нужда
    public SupportDTO getSupport() {
        return support;
    }

    @JsonProperty("support") //ако името съвпада, то няма нужда
    public void setSupport(SupportDTO support) {
        this.support = support;
    }

    @Override
    public String toString() {
        return "User{" +
                "data=" + data +
```

```
                ", support=" + support +
                '}';
    }
}
```

```
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;

@JsonInclude(JsonInclude.Include.NON_NULL)
public class DataDTO {
    @JsonProperty("id") //ако името съвпада, то няма нужда
    private Integer id;
    @JsonProperty("email") //ако името съвпада, то няма нужда
    private String email;
    @JsonProperty("first_name") //ако името съвпада, то няма нужда
    private String firstName;
    @JsonProperty("last_name")  //ако името съвпада, то няма нужда
    private String lastName;
    @JsonProperty("avatar")   //ако името съвпада, то няма нужда
    private String avatar;

    public DataDTO() {
    }

    @JsonProperty("id")     //ако името съвпада, то няма нужда
    public Integer getId() {
        return id;
    }

    @JsonProperty("id")       //ако името съвпада, то няма нужда
    public void setId(Integer id) {
        this.id = id;
    }

    @JsonProperty("email")      //ако името съвпада, то няма нужда
    public String getEmail() {
        return email;
    }

    @JsonProperty("email")     //ако името съвпада, то няма нужда
    public void setEmail(String email) {
        this.email = email;
    }

    @JsonProperty("first_name")     //ако името съвпада, то няма нужда
    public String getFirstName() {
        return firstName;
    }

    @JsonProperty("first_name")      //ако името съвпада, то няма нужда
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    @JsonProperty("last_name")      //ако името съвпада, то няма нужда
```

```
    public String getLastName() {
        return lastName;
    }

    @JsonProperty("last_name")      //ако името съвпада, то няма нужда
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    @JsonProperty("avatar")      //ако името съвпада, то няма нужда
    public String getAvatar() {
        return avatar;
    }

    @JsonProperty("avatar")     //ако името съвпада, то няма нужда
    public void setAvatar(String avatar) {
        this.avatar = avatar;
    }

    @Override
    public String toString() {
        return "Data{" +
                "id=" + id +
                ", email='" + email + '\'' +
                ", firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", avatar='" + avatar + '\'' +
                '}';
    }
}
```

```
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;

@JsonInclude(JsonInclude.Include.NON_NULL)
public class SupportDTO {

    @JsonProperty("url")  //ако името съвпада, то няма нужда
    private String url;
    @JsonProperty("text")   //ако името съвпада, то няма нужда
    private String text;

    @JsonProperty("url")    //ако името съвпада, то няма нужда
    public String getUrl() {
        return url;
    }

    @JsonProperty("url")    //ако името съвпада, то няма нужда
    public void setUrl(String url) {
        this.url = url;
    }

    @JsonProperty("text")    //ако името съвпада, то няма нужда
    public String getText() {
        return text;
```

```
    }

    @JsonProperty("text")    //ако името съвпада, то няма нужда
    public void setText(String text) {
        this.text = text;
    }

    @Override
    public String toString() {
        return "Support{" +
                "url='" + url + '\'' +
                ", text='" + text + '\'' +
                '}';
    }
}
```

9.5. DOM Manipulations – plain Vanilla JS

Creating DOM Elements

- Create with document.createElement

```
let p = document.createElement('p');
```

- Append text to the <p> element

```
let text = document.createTextNode('Random Text');
p.appendChild(text);
```

- Text added to textContent will be escaped.
- Text added to innerHTML will be parsed and turned into actual HTML elements - beware of XSS attacks!

```
let list = document.createElement('ul');
let liPeter = document.createElement('li');
liPeter.textContent = 'Peter';
list.appendChild(liPeter);
let liMaria = document.createElement('li');
liMaria.innerHTML = '<b>Maria</b>';
list.appendChild(liMaria);

document.body.appendChild(list);
```

```
▼<ul>
    <li>Peter</li>
  ▼<li>
      <b>Maria</b>
    </li>
  </ul>
```

Deleting DOM Elements

- To remove an HTML element, you must know his parent

```
<div id='div1'>
    <p id='p1'>This is a paragraph.</p>
    <p id='p2'>This is another paragraph.</p>
</div>

let parent = document.getElementById('div1');
let child = document.getElementById('p1');
parent.removeChild(child);
```

9.6. Browser Events and DOM Events – plain Vanilla JS

Handling Events in JS

- Browsers send events to notify the JS code of interesting things that have taken place

```
<div id='text'>Some text</div>
```

```
let div = document.getElementById('text');
div.onmouseover = function(event) {
    event.target.style.border = '3px solid green';
}
div.onmouseout = function() {
    this.style.border = ''; // this === event.target
}
```

Event Types in DOM API

Focus event – когато имаме поле input и пишем в него

Attach / Remove Events

- Attach an event to an element.

```
let textbox = document.createElement('input');
textbox.type = 'text';
textbox.value = 'I am a text box';
document.body.appendChild(textbox);
textbox.addEventListener('focus', focusHandler);
```

- Remove an event.

```
function focusHandler(event) {
    textbox.value = 'Event handler removed';
    textbox.removeEventListener('focus', focusHandler);
}
```

Multiple Events

- The **addEventListener()** method also allows you to add many events to the same element, without overwriting existing events:

```
element.addEventListener('click', function);
element.addEventListener('click', myFunction);
element.addEventListener('mouseover', mySecondFunction);
element.addEventListener('mouseout', myThirdFunction);
```

- Note that you don't use the 'on' prefix for the event; use 'click' instead of 'onclick'.

9.7. Fetch API

Fetch API

- Fetch provides a generic definition of Request and Response objects
- Fetch API allows you to make network requests similar to **XMLHttpRequest** (XHR).
- The response of a **fetch()** is a Stream object.

Example:

HomeController.java

```
@GetMapping('/')
public ModelAndView index(ModelAndView modelAndView) {
    modelAndView.setViewName('index');
    return modelAndView;
}

@GetMapping(value = '/fetch', produces = 'application/json')
@ResponseBody
public Object fetchData() {
    return new ArrayList<Product>() {{
        add(new Product(){{
            setName('Chewing Gum');
            setPrice(new BigDecimal(1.00));
            setBarcode('133242556222');
        }});
    ...
    }};}


public class Product {
    private String name;
    private BigDecimal price;
    private String barcode;

    // Getters & Setters
    ...
}
```

- Now let's head to the view
  - There is no need for a separate .js file for one-time use

index.html

```
...
<div class='container-fluid'>
    <h1 class='text-center mt-5 display-1'>Data Fetch</h1>
    <div class='data-container mt-5'></div>
    <div class='button-holder mt-5'>
        <button id='fetch-button' class='btn btn-info'>Fetch Data</button>
        <button id='clear-button' class='btn btn-secondary'>Clear Data</button>
    </div>
</div>
<script>
    // jQuery Event handlers
    $('#fetch-button').click(() => {...}); // Fetch and render the data
    $('#clear-button').click(() => $('.data-container').empty()); // Clear the data
</script>
```

```
$('#fetch-button').click(() => {
    fetch('http://localhost:8000/fetch') // Fetch the data (GET request)
        .then((response) => response.json()) // Extract the JSON from the Response
        .then((json) => json.forEach((x, y) => { // Render the JSON data to the HTML
            if (y % 4 === 0) {
                $('.data-container').append('<div class='row d-flex justify-content-around mt-
4'>');
            }

            let divColumn =
                '<div class='col-md-3'>' +
            '<h3 class='text-center font-weight-bold'>' + x.name + '</h3>' +
            '<h4 class='text-center'>Price: $' + x.price + '</h4>' +
            '<h4 class='text-center'>Barcode: $' + x.barcode + '</h4>' +
            '</div>';

            $('.data-container .row:last-child').append(divColumn);
        }));
});
```

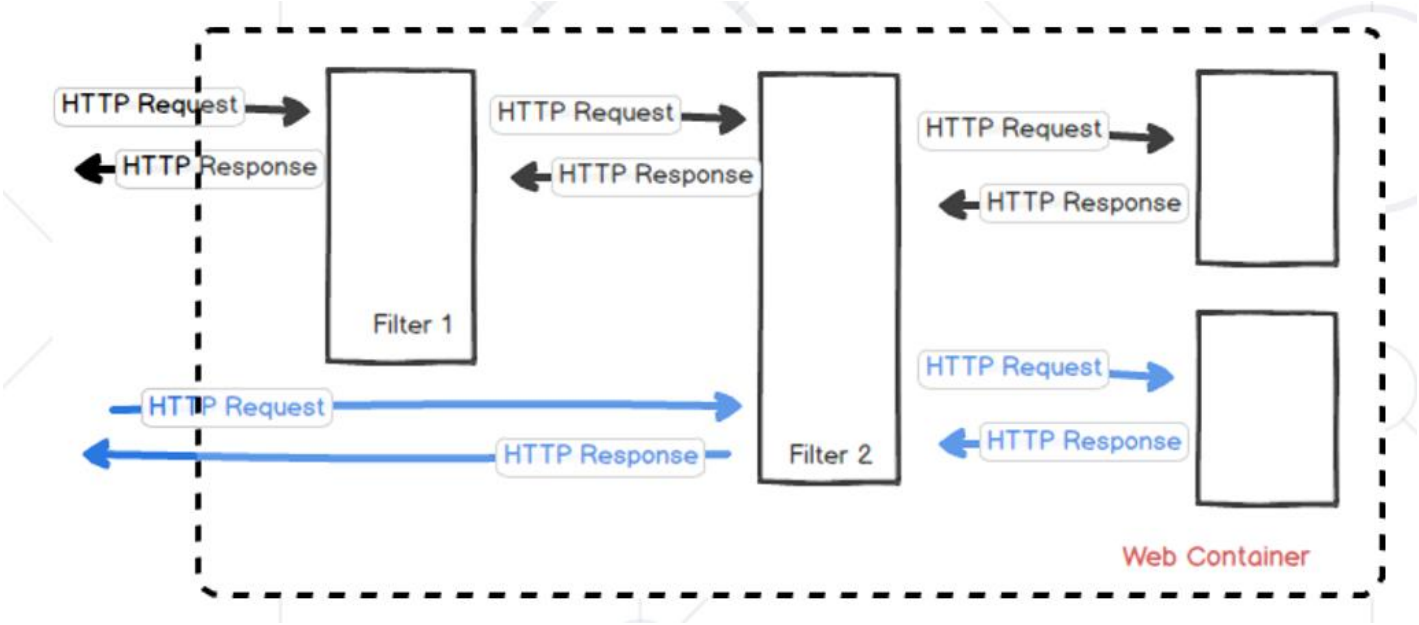
10. Spring Security

10.1. Filters and Interceptors

Filters

- A filter is an object used to **intercept** the HTTP **requests** and **responses** of your application
- We can perform two operations at two instances:
    - Before sending the **request** to the controller
    - Before sending a **response** to the client
    - Всеки по пътя филтър 1 - > филтър 2 може да променя нещо

Filters Diagram



Filter Example

```
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class GreetingFilter implements Filter {

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;

        request.getSession().setAttribute('name', 'Pesho');

        filterChain.doFilter(request, response);
    }
}

import javax.servlet.http.HttpSession;

@Controller
public class HomeController {

    @GetMapping('/')
    public ModelAndView index(ModelAndView modelAndView, HttpSession session) {
        modelAndView.setViewName('index');
        modelAndView.addObject('name', session.getAttribute('name'));

        return modelAndView;
```

```
    }
}
```

index.html

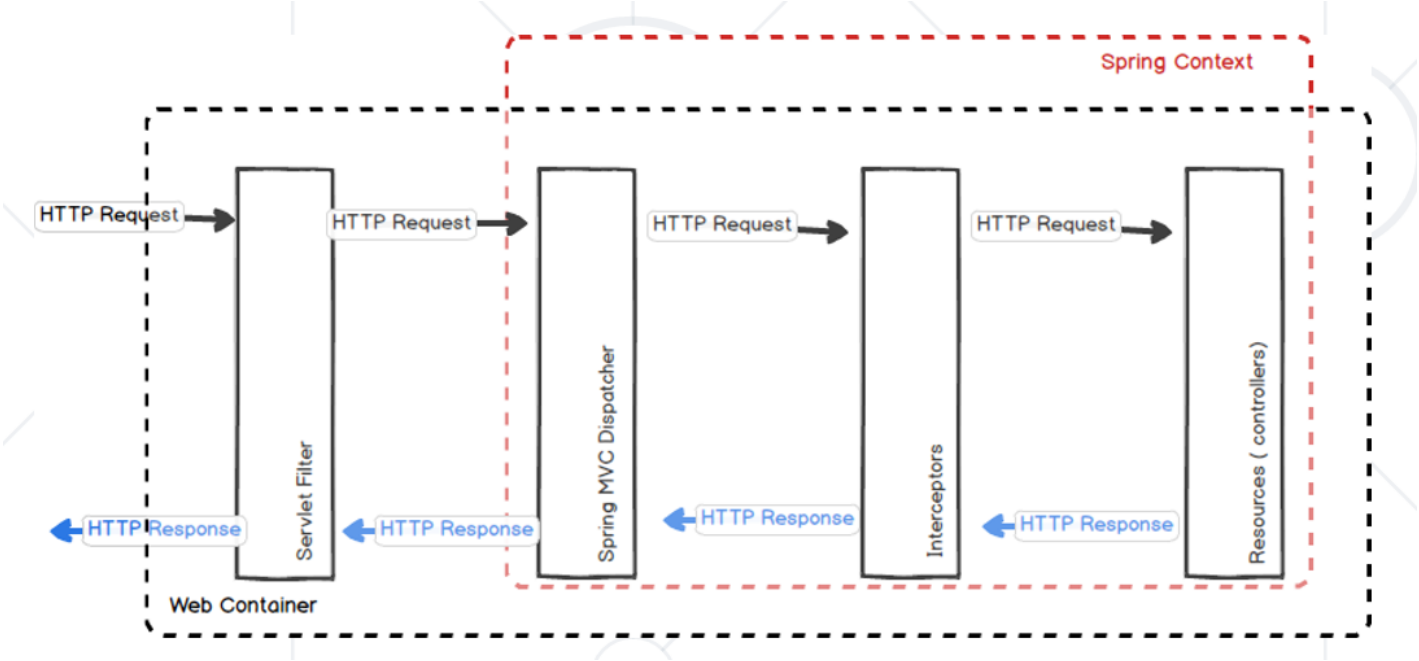
```
<!DOCTYPE html>
<html lang='en' xmlns='http://www.w3.org/1999/xhtml' xmlns:th='http://www.thymeleaf.org'>
<head>
    <meta charset='UTF-8'>
    <title>Filter Demo</title>
</head>
<body>
<h1 th:text='|Hello, ${name}!|'></h1>
</body>
</html>
```

Interceptor

- A **Filter** is used in the **web layer only** as it is defined in web.xml. We can not use it out of web context
- While Spring **Interceptors** are defined in the **Spring context за прихващане на неща**
- The **interceptor** include **three** main **methods**:
    - **preHandle**: executed before the execution of the target resource – преди да пристигне http заявката до контролера
    - **afterCompletion**: executed after the execution of the target resource (after rendering the view) – след като http заявката е минала през конттролера
    - **postHandle**: Intercept the execution of a handler – извиква се най-накрая

Interceptors можем да използваме за статистика, за поставяне на http headers, за филтриране на http headers

Interceptor Diagram



Interceptor Example

```
import org.springframework.web.servlet.HandlerInterceptor;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class LogingInteceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain, Object handler) throws IOException, ServletException {
        //Log some information ...
        //да си пазим статистика колко request-а и response-а са минали примерно.

        retrun true;
    }
}
```

Register Interceptor in Configuration

- To use interceptors we need to register them

```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfiguration implements WebMvcConfigurer {
```

```
    private final MyInterceptor myInterceptor;

    public WebConfiguration(MyInterceptor myInterceptor) {
        this.myInterceptor = myInterceptor;
    }

    @Override
    public void Spring Security (InterceptorRegistry registry) {
        registry.addInterceptor(myInterceptor);
    }
}
```

10.2. Spring Security

What is Spring Security?

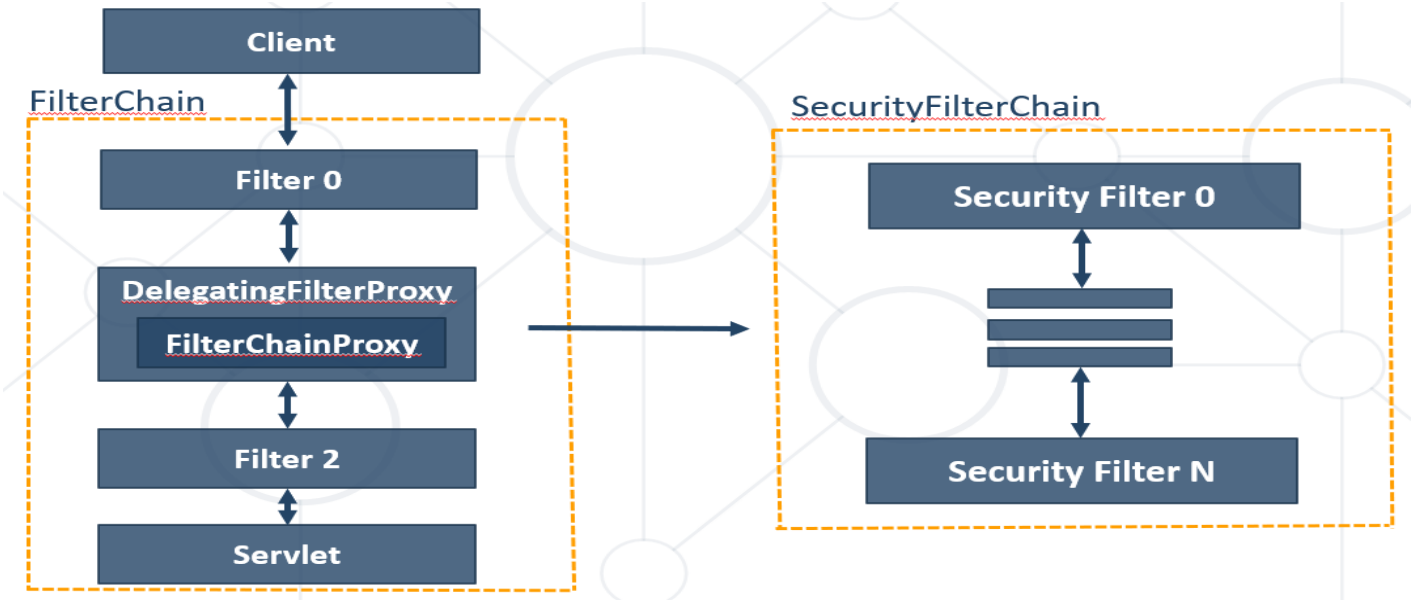
- A **powerful** and **highly customizable** authentication and access-control framework
- It is the de-facto **standard** for securing **Spring-based** applications
- Focuses on providing both **authentication** and **authorization** to Java applications

- **Authentication**
  - Who is logged in

- **Authorization**
  - What you are allowed to do

Spring Security Filter Chain

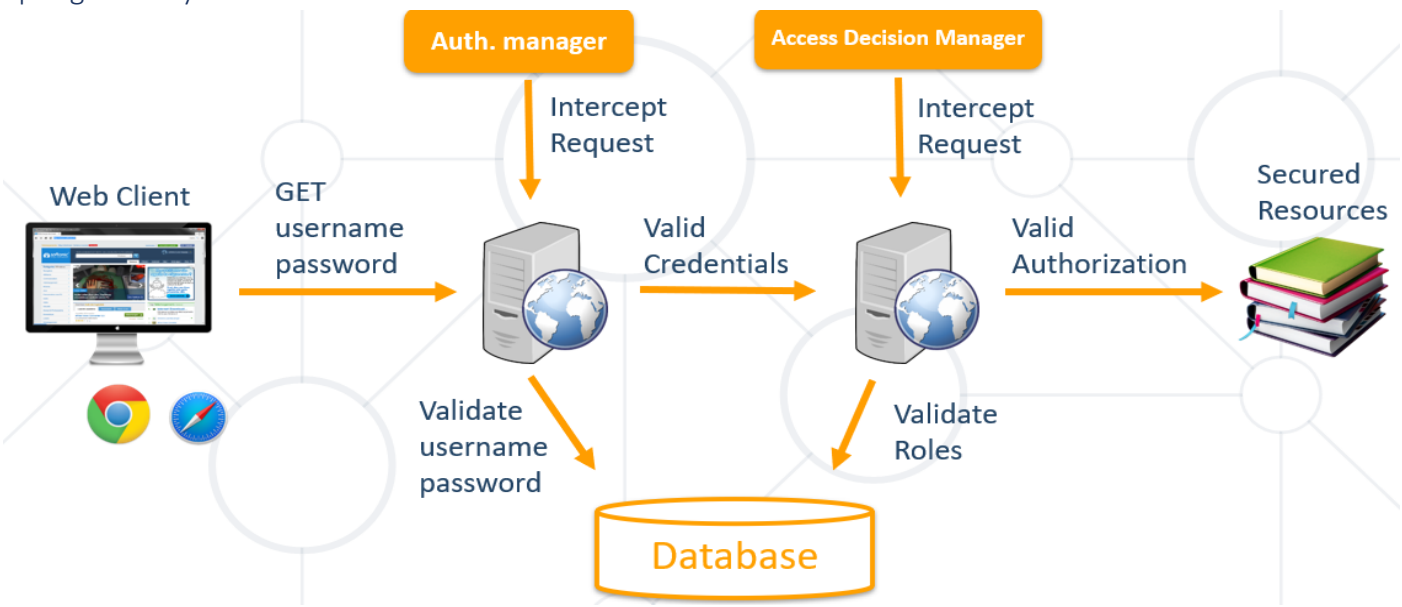
Spring Security реално е един филтър.



Security Context and Authentication

- At the heart of Spring Security's authentication model is the **SecurityContextHolder**
- It contains the **SecurityContext**

Spring Security Mechanism



Spring Security Maven/Gradle

build.gardle

```
implementation 'org.springframework.boot:spring-boot-starter-security'
implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity5'
```

pom.xml

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

Spring Security Configuration

Deprecated

- Extending the **WebSecurityConfigurerAdapter** class - deprecate-нат

```
@Configuration
@EnableWebSecurity //Can be omitted because of WebSecurityConfigurerAdapter
public class SecurityConfiguration extends ~~WebSecurityConfigurerAdapter~~ {
//Configuration goes here
```

- **Override configure**(HttpSecurity http)

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
```

```
            .authorizeRequests()    // Authorize Requests
            .antMatchers('/', '/register').permitAll()    //Permit Routes
            .anyRequest().authenticated()          //Require Authentication
}

}
```

Up-to-Date way

- Creating the **SecurityFilterChain** bean.

```
@Configuration
public class SecurityConfiguration {
        @Bean
        public SecurityFilterChain filterChain(HttpSecurity httpSecurity) {
            httpSecurity
                 .authorizeRequests()     // Authorize Requests
                    .antMatchers('/', '/register').permitAll()  //Permit Routes
                    .anyRequest().authenticated();  //Require Authentication
    …
            return http.build();
        }
}
```

Registration User

With interface

```
public interface UserDetails {
    Collection<? extends GrantedAuthority> getAuthorities();

    String getPassword();

    String getUsername();

    boolean isAccountNonExpired();

    boolean isAccountNonLocked();

    boolean isCredentialsNonExpired();

    boolean isEnabled();
}


@Entity
public class UserEntity implements UserDetails {
    private String username;
    private String password;
    private boolean isAccountNonExpired;
    private boolean isAccountNonLocked;
    private boolean isCredentialsNonExpired;
    private boolean isEnabled;
    private Set<Role> authorities;
}


UserDetails ud =
        User.
```

```
                withUsername(..).
    password(..).
    authorities(..).
    build();
```

Registration – Roles

- Implementing the **GrantedAuthority** interface.

```
import org.springframework.security.core.GrantedAuthority;

public class Role implements GrantedAuthority {
    private String authority;

    @Override
    public String getAuthority() {
        return null;
    }
}
```

SimpleGrantedAuthority

- If we want, we can use **SimpleGrantedAuthority** instead of creating Role class
- Is a basic concrete **implementation** of a **GrantedAuthority**
- Stores a **String representation** of an **authority** granted to the Authentication object

UserDetailsService

- Implementing the **UserDetailsService** interface.

```
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    public UserDetailsServiceImpl() {
     …
    }

    @Override
    public UserDetails loadUserByUserName(String userName) {
        // get the user and map to UserDetails
    }
```

```
@Autowired
private BCryptPasswordEncoder bCryptPasswordEncoder

public void register(RegisterModel registerModel) {
    bCryptPasswordEncoder.encode(password));
}
```

```
}
```

- Expose as a bean

```
@Configuration
public class SecurityConfig {

    @Bean
    public UserDetailsService userDetailsService(UserRepository
                                                        userRepository) {
        return new UserDetailsServiceImpl(userRepository);
    }

@Bean
public PasswordEncoder passwordEncoder() {
    return new Pbkdf2PasswordEncoder();
}



  …
}
```

Register User In memory with overriding configure

Едва ли ще се наложи в практиката да ги съхраняваме в паметта register и login

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth
            .inMemoryAuthentication(.withUser("user")
            .password(bCryptPasswordEncoder.encode("user"))
            .roles("USER")

.and(.withUser("admin").password(bCryptPasswordEncoder.encode("admin")).roles("ADMIN")
}…
```

Registration – Configuration

- Disabling **CSRF** protection temporarily.

```
@Configuration
public class SecurityConfiguration {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity httpSecurity){
        //Configuration goes here
        @Bean
        public SecurityFilterChain filterChain(HttpSecurity httpSecurity) {
            http.authorizeRequests()
                    .antMatchers('/', '/register').permitAll()
                    .anyRequest().authenticated();
    …
            return http.build();
        }

        @Override
```

```
        protected void configure(HttpSecurity http) throws Exception {
            http
                    .and()
                    .csrf()
                .disable();
        }
    }
}
```

Login Mechanism

Login – Configuration

SecurityConfiguration.java

```
.and()
        .formLogin().loginPage('/login').permitAll()
        .usernameParameter('username')
        .passwordParameter('password')
```

login.html

```
<input type='text' name='username'/>
<input type='text' name='password'/>
```

Login – UserService

```
@Service
public class UserServiceImpl implements UserDetailsService {
    // Some userServiceImpl logic
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //...
    }
}
```

Login – Controller

```
@Controller
public class LoginController {
    @GetMapping('/login')
    public String getLoginPage(@RequestParam(required = false) String error, Model model) {
        if (error != null) {
            model.addAttribute('error', 'Error');
        }

        return 'login';
    }
}
```

Logout

SecurityConfiguration.java

```
.and()
        .logout().logoutSuccessUrl('/login?logout').permitAll()
```

Remember Me

SecurityConfiguration.java

```
.and()
        .rememberMe()
        .rememberMeParameter('remember')
        .key('remember Me Encryption Key')
        .rememberMeCookieName('rememberMeCookieName')
        .tokenValiditySeconds(10000)
```

login.html

```
<input name='remember' type='checkbox' />
```

Principal

- This is the **currently logged user**

Можем да го инжектнем само в контролера

UserController.java

```
@GetMapping('/user')
public String getUser(Principal principal) {
    System.out.println(principal.getName());
    return 'user';
}
```

Demo using principal

```
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.User;

import java.util.Collection;

public class MobileleUser extends User {
```

```
    public MobileleUser(String username, String password, Collection<? extends GrantedAuthority>
authorities) {
        super(username, password, authorities);
    }

    public MobileleUser(String username, String password, boolean enabled, boolean
accountNonExpired, boolean credentialsNonExpired, boolean accountNonLocked, Collection<? extends
GrantedAuthority> authorities) {
        super(username, password, enabled, accountNonExpired, credentialsNonExpired,
accountNonLocked, authorities);
    }

    //some own methods
    public String getUserIdentifier(){
        return getUsername();
    }
}


import bg.softuni.mobilele.model.entity.UserEntity;
import bg.softuni.mobilele.repository.UserRepository;
import bg.softuni.mobilele.user.MobileleUser;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class MobileleUserServiceImpl implements UserDetailsService {
    private final UserRepository userRepository;

    public MobileleUserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UserEntity userEntity = this.userRepository
                .findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User with name " + username +
" not found."));

        return mapToUserDetails(userEntity);
    }

    //The purpose of this method is to map our user representation (UserEntity)
    // to the user representation in the Spring security world (UserDetails)
    // The only thing that Spring will provide to us is the username.
    // The username will come from the HTML login form.
    private static UserDetails mapToUserDetails(UserEntity userEntity){
        //Granted authority is the representation of a user role in the Spring world.
```

```
        // SimpleGrantedAuthority is an implementation of GrantedAuthority which Spring provides
for our convenience
        // Our representation of role is UserRoleEntity
        List<GrantedAuthority> authorities = userEntity
                .getUserRoles()
                .stream()
                .map(r -> new SimpleGrantedAuthority("ROLE_" + r.getUserRole().name()))
                .collect(Collectors.toList());

        //User is the Spring implementation of UserDetails interface.
//        return new User(
//                userEntity.getUsername(),
//                userEntity.getPassword(),
//                authorities
//        );
        return new MobileleUser(
                userEntity.getUsername(),
                userEntity.getPassword(),
                authorities
        );
    }
}



    @PostMapping("/offers/add")
    public String addOffer(@Valid AddOfferDTO addOfferModel,
                           BindingResult bindingResult,
                           RedirectAttributes redirectAttributes,
//                             Principal principal
                           @AuthenticationPrincipal MobileleUser user
    ) {

        if (bindingResult.hasErrors()) {
            redirectAttributes.addFlashAttribute("addOfferModel", addOfferModel);

redirectAttributes.addFlashAttribute("org.springframework.validation.BindingResult.addOfferModel
", bindingResult);
            return "redirect:/offers/add";
        }

        //Save in the DB
//        this.offerService.addOffer(addOfferModel, principal);
        this.offerService.addOffer(addOfferModel, user.getUserIdentifier());

        return "redirect:/offers/all";
    }



//    public void addOffer(AddOfferDTO addOfferDTO, Principal principal) {
    public void addOffer(AddOfferDTO addOfferDTO, String ownerId) {
//        OfferAddServiceModel offerAddServiceModel = modelMapper.map(addOfferDTO,
OfferAddServiceModel.class);
//        OfferEntity newOffer = modelMapper.map(offerAddServiceModel, OfferEntity.class);
//        newOffer.setCreated(Instant.now());
//        newOffer.setSeller(userRepository.findByUsername(principal.getName()).orElseThrow());
```

```
//        newOffer.setSeller(userRepository.findByUsername(ownerId).orElseThrow());
//        ModelEntity model = modelRepository.getById(addOfferDTO.getModelId());
//        newOffer.setModel(model);

        OfferEntity newOffer = this.offerMapper.addOfferDtoToOfferEntity(addOfferDTO);
        //TODO - current user should be logged in

        Optional<UserEntity> seller = userRepository.findById(1L);
        ModelEntity model = modelRepository.findById(addOfferDTO.getModelId()).orElseThrow();
//        ModelEntity model = modelRepository.findById(ownerId).orElseThrow();


//        OfferEntity newOffer = modelMapper.map();

        newOffer.setModel(model);
        newOffer.setSeller(seller.get());

        offerRepository.save(newOffer);
    }
```

Pre / Post Authorize – на ниво метод – както Controller-ите, така и Service-те и техните методи !!!

- Grant Access to specific methods – на ниво метод

За да активираме @PreAuhtorize, в класа със Spring security chain, извикваме
`@EnableGlobalMethodSecurity(prePostEnabled = true)`

SecurityConfiguration.java

```
@EnableGlobalMethodSecurity(prePostEnabled = true)   //Enables PreAuthorize
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

}
```

UserService.java

```
public interface UserService extends UserDetailsService {
    @PreAuthorize('hasRole('ADMIN')')        //Requires Admin Role to execute
    void delete();
}
```

No Access Handling

SecurityConfiguration.java

```
.and()
        .exceptionHandling().accessDeniedPage('/unauthorized')
```

AccessController.java

```
@GetMapping('/unauthorized')
public String unauthorized() {
    return 'unauthorized';
}
```

10.3. Cross-Site Request Forgery (подправяне/фалшификация)

Spring CSFR Protection

```
.csrf()
        .csrfTokenRepository(csrfTokenRepository())

private CsrfTokenRepository csrfTokenRepository() {
        HttpSessionCsrfTokenRepository repository = new HttpSessionCsrfTokenRepository();
        repository.setSessionAttributeName("_csrf");
        return repository;
        }
```

form.html

```
<input type='hidden' th:name='${_csrf.parameterName}' th:value='${_csrf.token}' />
```

Spring и Thymeleaf автоматично ни изсипва value csrf токен

```
<form action="/users/logout" method="post">
    <input type="hidden" name="_csrf" value="5c685dcc-6031-49bb-96aa-607344873b1e"> == $
    <input class="nav-link" type="submit" value="Logout">
</form>
```

При заявка POST, тогава се включва CSRF защитата. Хакера не знае стойността на value csrf токена.

Можем и ръчно да си го сложим, но реално няма смисъл. Spring сам си го търси при post заявки.

```
<input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}">
```

10.4. Thymeleaf Security

Functionality to Thymeleaf

pom.xml

```
<dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

build.gardle

```
implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity5'
```

Principal

index.html

```
<!DOCTYPE html>
<html lang='en'
      xmlns:th='http://www.thymeleaf.org'
      xmlns:sec='http://www.thymeleaf.org/extras/spring-security'>
<body>
<div sec:authentication='name'>
    The value of the 'name' property of the authentication object should appear here.
```

```
</div>
</body>
</html>
```

index.html

```
<!DOCTYPE html>
<html lang='en'
      xmlns:th='http://www.thymeleaf.org'
      xmlns:sec='http://www.thymeleaf.org/extras/spring-security'>
<body>
<div sec:authorize='hasRole('ADMIN')'>
This content is only shown to administrators.
</div>
</body>
</html>
```

10.5. Demo Spring Security 1 – with the deprecated WebSecurityConfigurerAdapter

```
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class MobileleUserServiceImpl implements UserDetailsService {
    private final UserRepository userRepository;

    public MobileleUserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UserEntity userEntity = this.userRepository
                .findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User with name " + username +
" not found."));

        return mapToUserDetails(userEntity);
    }

    //The purpose of this method is to map our user representation (UserEntity)
    // to the user representation in the Spring security world (UserDetails)
    // The only thing that Spring will provide to us is the username.
    // The username will come from the HTML login form.
    private static UserDetails mapToUserDetails(UserEntity userEntity){
        //Granted authority is the representation of a user role in the Spring world.
        // SimpleGrantedAuthority is an implementation of GrantedAuthority which Spring provides
for our convenience
        // Our representation of role is UserRoleEntity
        List<GrantedAuthority> authorities = userEntity
                .getUserRoles()
```

```
                .stream()
                .map(r -> new SimpleGrantedAuthority("ROLE_" + r.getUserRole().name()))
                .collect(Collectors.toList());

        //User is the Spring implementation of UserDetails interface.
        return new User(
                userEntity.getUsername(),
                userEntity.getPassword(),
                authorities
        );
    }
}


@Controller
@RequestMapping("/users")
public class UserLoginController {
    private UserService userService;

    public UserLoginController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping("/login")
    public String login() {
        return "auth-login";
    }
}



import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuil
der;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
public class ApplicationSecurityConfiguration extends WebSecurityConfigurerAdapter {
    private final UserDetailsService userDetailsService;
    private final PasswordEncoder passwordEncoder;

    public ApplicationSecurityConfiguration(UserDetailsService userDetailsService,
                                            PasswordEncoder passwordEncoder) {
        this.userDetailsService = userDetailsService;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
```

```
                .authorizeRequests()

                //with this line we allow access to all static resources

.requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()

                //the next line allows access to the home page, login page and register page
                .antMatchers("/", "/users/login", "/users/register").permitAll()

                //next, we forbid all other pages for unauthenticated users
                .antMatchers("/**").authenticated()

                .and()
тези код валидира Login само за @Controller анотация и само през html формуляр
                //configure login with login html form with two imput fields
                .formLogin()

                //our login page is located at http://<serveraddress>:<port>/users/login
                .loginPage("/users/login")

                //This is the name of the <input...> in the lofing form where the user enters
her e-mail/username
                // the value of this input will be presented to our User details service
                // those that want to nam ethe inout field differnetly =, e.g. email may change
the value below

.usernameParameter(UsernamePasswordAuthenticationFilter.SPRING_SECURITY_FORM_USERNAME_KEY)

                //The name of the field that keeps the password

.passwordParameter(UsernamePasswordAuthenticationFilter.SPRING_SECURITY_FORM_PASSWORD_KEY)

                //The place, where we should land in case that the login is successfull
                .defaultSuccessUrl("/")

                //the place where I should land if the login is NOT successfull
                .failureForwardUrl("/users/login-error")

                .and()
                .logout()
                //This is the URL which Spring will implement for me and will log the user put
                .logoutUrl("/users/logout")
                //where to go after logout
                .logoutSuccessUrl("/")

                //removes the session from the backend server
                .invalidateHttpSession(true)

                //delete the cookie that references my session
                .deleteCookies("JSESSIONID");
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // This gives Spring two important components.
        // 1. Our user details service that translates usernames/emails, phone numbers, etc to
UserDetails
```

```
        // 2. Password encoder - the component that can decide if the user password matches

        auth
                .userDetailsService(userDetailsService)
                .passwordEncoder(passwordEncoder);

        // registration
        //topsecretpass -> password encoder -> kwrnfwfewkfjkdkwqkdqwpqflsw (hashed_password)


        // login
        // (username, raw_password)
        // password_encoder.matches(raw_password, hashed_password)

    }
}
```

```
<div class="container-fluid">
    <h5 class="text-center text-white mt-5"> Welcome to MobiLeLeLe,
        <span
                sec:authorize="isAuthenticated()"
                sec:authentication='name'>
            Name of the logged user
        </span>

        <span
                sec:authorize="!isAuthenticated()"
                sec:authentication='name'>
            Anonymous
        </span>
    </h5>
</div>
```

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto col-12 justify-content-between">
        <li class="nav-item" sec:authorize="isAuthenticated()">
            <a class="nav-link" th:href="@{/brands/all}">All Brands</a>
        </li>
        <li class="nav-item" sec:authorize="isAuthenticated()">
            <a class="nav-link" th:href="@{/offers/add}">Add Offer</a>
        </li>
        <li class="nav-item" sec:authorize="isAuthenticated()">
            <a class="nav-link" th:href="@{/offers/all}">All Offers</a>
        </li>

        <li class="nav-item dropdown" sec:authorize="hasRole('ADMIN')">
            <a class="nav-link dropdown-toggle" href="/" id="navbarDropdown" role="button"
               data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                Admin
            </a>
            <div class="dropdown-menu" aria-labelledby="navbarDropdown">
                <a class="dropdown-item" href="/">Action</a>
                <a class="dropdown-item" href="/">Another action</a>
                <div class="dropdown-divider"></div>
```

```
                <a class="dropdown-item" href="/">Something else here</a>
            </div>
        </li>

        <!-- Logout start -->
        <li class="nav-item" sec:authorize="isAuthenticated()">
            <div class="form-inline my-2 my-lg-0 border px-3">
                <div class="text-white">Welcome, <th:block
sec:authentication='name'></th:block></div>
                <form th:action="@{/users/logout}" th:method="post">
                    <input class="nav-link" type="submit" value="Logout">
                </form>
            </div>
        </li>

        <li class="nav-item" sec:authorize="!isAuthenticated()">
            <a class="nav-link" th:href="@{/users/register}">Register</a>
        </li>

        <li class="nav-item" sec:authorize="!isAuthenticated()">
            <a class="nav-link" th:href="@{/users/login}">Login</a>
        </li>
    </ul>
</div>
```

10.6. Demo Spring Security 2 – with SecurityFilterChain

```
import bg.softuni.security.model.enums.UserRoleEnum;
import bg.softuni.security.repository.UserRepository;
import bg.softuni.security.service.AppUserDetailsService;
import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.crypto.password.Pbkdf2PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
public class SecurityConfiguration {

  //Here we have to expose 3 things:
  // 1. PasswordEncoder
  // 2. SecurityFilterChain
  // 3. UserDetailsService

  @Bean
  public PasswordEncoder passwordEncoder() {
    return new Pbkdf2PasswordEncoder();
  }

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.
```

```
        // define which requests are allowed and which not
        authorizeRequests().
        // everyone can download static resources (css, js, images)
        requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll().
        // everyone can login and register
        antMatchers("/", "/users/login", "/users/register").permitAll().
        // pages available onny for moderators
        antMatchers("/pages/moderators").hasRole(UserRoleEnum.MODERATOR.name()).
        // pages available only for admins
        antMatchers("/pages/admins").hasRole(UserRoleEnum.ADMIN.name()).
        // all other pages are available for logger in users
        anyRequest().
        authenticated().
    and().
```

този код надолу валидира Login само за @Controller анотация и само през html формуляр

```
        // configuration of form login
        formLogin().
        // the custom login form
        loginPage("/users/login").
        // the name of the username form field

usernameParameter(UsernamePasswordAuthenticationFilter.SPRING_SECURITY_FORM_USERNAME_KEY).
        // the name of the password form field

passwordParameter(UsernamePasswordAuthenticationFilter.SPRING_SECURITY_FORM_PASSWORD_KEY).
        // where to go in case that the login is successful
        defaultSuccessUrl("/").
        // where to go in case that the login failed
        failureForwardUrl("/users/login-error").
    and().
        // configure logut
        logout().
        // which is the logout url
        logoutUrl("/users/logout").
        // invalidate the session and delete the cookies
        invalidateHttpSession(true).
        deleteCookies("JSESSIONID");


    return http.build();
  }

  @Bean
  public UserDetailsService userDetailsService(UserRepository userRepository) {
    return new AppUserDetailsService(userRepository);
  }
}
```

```
import bg.softuni.security.model.entity.UserEntity;
import bg.softuni.security.model.entity.UserRoleEntity;
import bg.softuni.security.repository.UserRepository;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
```

```
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

// NOTE: This is not annotated as @Service, because we will return it as a bean.
public class AppUserDetailsService implements UserDetailsService {
  private final UserRepository userRepository;

  public AppUserDetailsService(UserRepository userRepository) {
    this.userRepository = userRepository;
  }

  @Override
  public UserDetails loadUserByUsername(String username)
      throws UsernameNotFoundException {
    return userRepository.
        findByEmail(username).
        map(this::map).
        orElseThrow(() -> new UsernameNotFoundException("User with email " + username + " not
found!"));
  }

  private UserDetails map(UserEntity userEntity) {
    return
        User.builder().
            username(userEntity.getEmail()).
            password(userEntity.getPassword()).
            authorities(userEntity.
                getUserRoles().
                stream().
                map(this::map).
                toList()).
            build();
  }

  private GrantedAuthority map(UserRoleEntity userRole) {
    return new SimpleGrantedAuthority("ROLE_" +
        userRole.
            getUserRole().name());
  }
}


@Controller
public class LoginController {

  @GetMapping("/users/login")
  public String login() {
    return "auth-login";
  }

  // NOTE: This should be post mapping!
  @PostMapping("/users/login-error")
  public String onFailedLogin(
      @ModelAttribute(UsernamePasswordAuthenticationFilter.SPRING_SECURITY_FORM_USERNAME_KEY)
String userName,
      RedirectAttributes redirectAttributes) {
```

```
redirectAttributes.addFlashAttribute(UsernamePasswordAuthenticationFilter.SPRING_SECURITY_FORM_U
SERNAME_KEY, userName);
    redirectAttributes.addFlashAttribute("bad_credentials",
        true);

    return "redirect:/users/login";
  }
}
```

10.7. Demo interceptor

```
@Configuration
public class ApplicationSecurityConfiguration extends ~~WebSecurityConfigurerAdapter~~ {
    private final UserDetailsService userDetailsService;
    private final PasswordEncoder passwordEncoder;

    public ApplicationSecurityConfiguration(UserDetailsService userDetailsService,
                                            PasswordEncoder passwordEncoder) {
        this.userDetailsService = userDetailsService;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()

                //with this line we allow access to all static resources

.requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()

                //the next line allows access to the home page, login page and register page
                .antMatchers("/", "/users/login", "/users/register").permitAll()
                //we permit the page below only only for admin users
                .antMatchers("/statistics").hasRole(UserRoleEnum.ADMIN.toString())


import bg.softuni.mobilele.web.interceptors.StatsInterceptor;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfiguration implements WebMvcConfigurer {
    private final StatsInterceptor statsInterceptor;

    public WebConfiguration(StatsInterceptor statsInterceptor){
        this.statsInterceptor = statsInterceptor;
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(statsInterceptor);
    }
}
```

```java
public class StatsView {
    private final int authRequests;
    private final int anonymousRequests;

    public StatsView(int authRequests, int anonymousRequests) {
        this.authRequests = authRequests;
        this.anonymousRequests = anonymousRequests;
    }

    public int getTotalRequests(){
        return authRequests + anonymousRequests;
    }

    public int getAuthRequests() {
        return authRequests;
    }

    public int getAnonymousRequests() {
        return anonymousRequests;
    }
}
```

```java
import bg.softuni.mobilele.model.view.StatsView;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

@Service
public class StatsService {
    private int anonymousRequests, authRequests;
    private StatsView statsView;

    public void onRequest(){
        Authentication authentication = SecurityContextHolder
                .getContext()
                .getAuthentication();

        if (authentication != null && (authentication.getPrincipal() instanceof UserDetails)) {
            authRequests++;
        } else {
            anonymousRequests++;
        }
    }

    public StatsView getStats(){
        return new StatsView(authRequests, anonymousRequests);
    }
}
```

```java
import bg.softuni.mobilele.service.StatsService;
import org.springframework.stereotype.Component;
```

```java
import org.springframework.web.servlet.HandlerInterceptor;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Component
public class StatsInterceptor implements HandlerInterceptor {
    private final StatsService statService;

    public StatsInterceptor(StatsService statService) {
        this.statService = statService;
    }

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler) throws Exception {
        statService.onRequest();
        return true;
    }
}
```

```java
@Controller
public class StatsController {
    private final StatsService statsService;

    public StatsController(StatsService statsService){
        this.statsService = statsService;
    }

    @GetMapping("/statistics")
    public ModelAndView statistics(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("stats", statsService.getStats());
        modelAndView.setViewName("stats");
        return modelAndView;
    }
}
```

```html
<div class="container-fluid">
    <h1 class="text-center text-white mt-5"> Welcome to Stats!</h1>
    <h3 class="text-center text-white mt-5">Total requests <th:block
th:text="${stats.getTotalRequests()}"></th:block></h3>
    <h3 class="text-center text-white mt-5">Anonymous requests <th:block
th:text="${stats.getAnonymousRequests()}"></th:block></h3>
    <h3 class="text-center text-white mt-5">Authorized requests <th:block
th:text="${stats.getAuthRequests()}"></th:block></h3>
</div>
```

10.8. SecurityExpressionRoot and MethodSecurityExpressionOperations

Watch lection of October 2021 – Events for this specific security customed configuration and usage
Implemented also in one of the diploma's project of other students.

public class OwnerSecurityExpressionRoot extends SecurityExpressionRoot implements MethodSecurityExpressionOperations
В този клас се задават Boolean методи, които се използват в @PreAuthorize("isOwner(#id)") като аргументи

public class MyMethodSecurityExpressionHandler extends DefaultMethodSecurityExpressionHandler

10.9. @Secured анотацията

11. (HATEOAS) Hypermedia As the Engine of Application State

11.1. What is HATEOAS

Hypermedia As the Engine of Application State

- **HATEOAS** is a constraint of the REST application architecture
- Keeps the RESTful style architecture **unique from most other network application** architectures
- Uses **hypermedia** to describe what future actions are available to the client
- Allowable actions are derived in the API based on the current application state and returned to the client as a **collection of links**

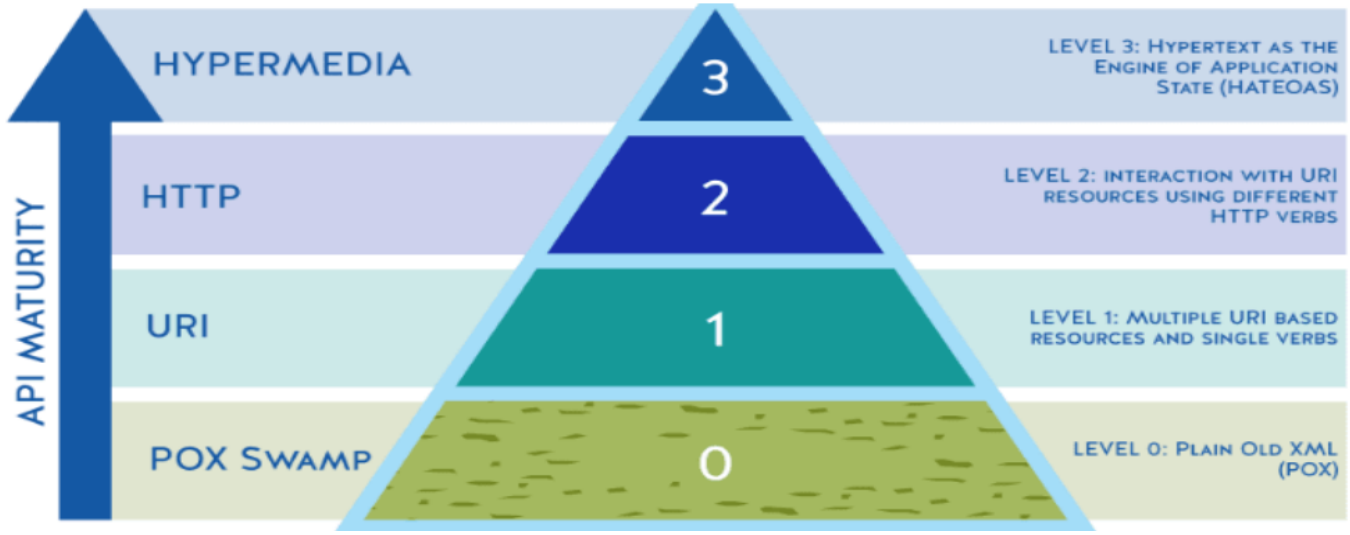
- Client uses these **links to drive further** interactions with the API

- Tells the client what **options** are **available** at a given point in time.
  - Doesn't tell them how each link should be used or exactly what information should be sent
- It is conceptually the same as a **web user browsing** through web pages by clicking the **relevant hyperlinks** to achieve a final goal

https://dev.to/ragrag/rest-api-maturity-towards-the-glory-of-rest-5cm3

The **Richardson Maturity Model** developed by Leonard Richardson is a heuristic that is used to indicate how mature a web service is in regards to its REST Architectural style.

The model identifies Level 3 as the **Glory of REST**



Пример за HATEOS в PayPal системата
https://developer.paypal.com/api/rest/responses/

11.2. HATEOAS Example

- Simple response **without** using **HATEOAS**
  - We have a simple REST controller that returns entity in JSON format to the client

```
{ "id" :2, "name": "Pesho", "age":12 }
```

- Using HATEOAS

```
{"id":2,
        "name":"Pesho",
        "age":12,
        " _links":{
                "self":{"href":"http://localhost:8080/students/2"},
                "delete":{"href":"http://localhost:8080/students/delete/2"},
                "update":{"href":"http://localhost:8080/students/update/2"},
                "orders":{"href":"http://localhost:8080/orders/allByStudentId/2"}
        }}
```

Rel & Href

- **rel** - describes the **relationship** between the Student resource and the URL
  - In example above **rel** is– self, update, delete …
  - **describes** the **action** that's performed with the link
  - It's important that this **value** is intuitive as it **describes** the purpose of the link
- **href** - the **URL** used to perform the action described in rel

11.3. Implement HATEOAS in Spring

Using HATEOAS in Spring Framework

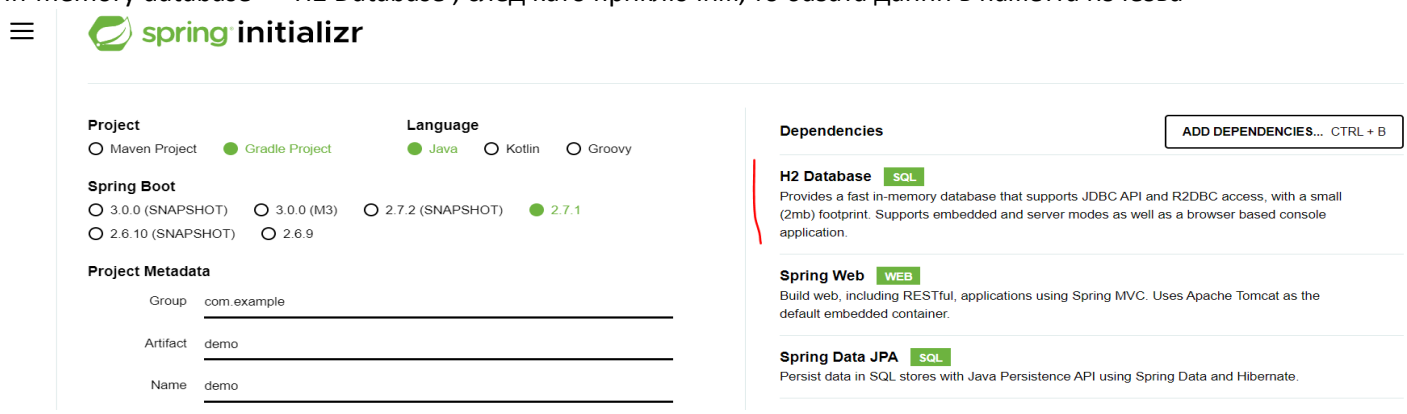
- Adding hypermedia links to RESTful responses is something you could implement on your own, but …
- **Spring HATEOAS** makes it **very easy (actually not so easy 😊)**

```
<dependency>
        <groupId>org.springframework.hateoas</groupId>
        <artifactId>spring-hateoas</artifactId>
        <version>1.1.0.RELEASE</version>
</dependency>
```

```
implementation 'org.springframework.hateoas:spring-hateoas'
```

Example with H2 In-Memory Database App DB

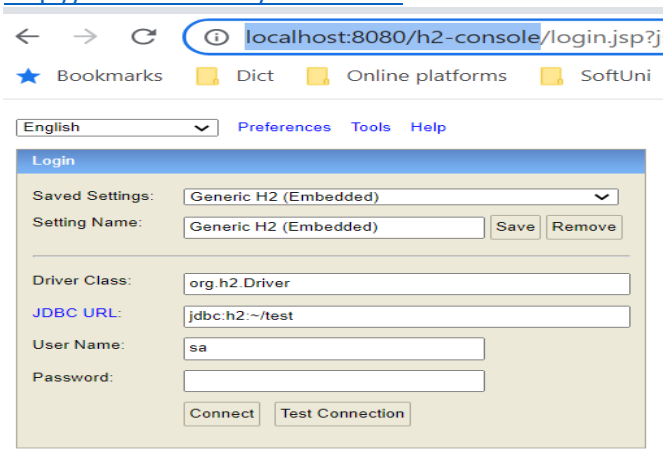
In-memory database == H2 Database ; след като приключим, то базата данни в паметта изчезва

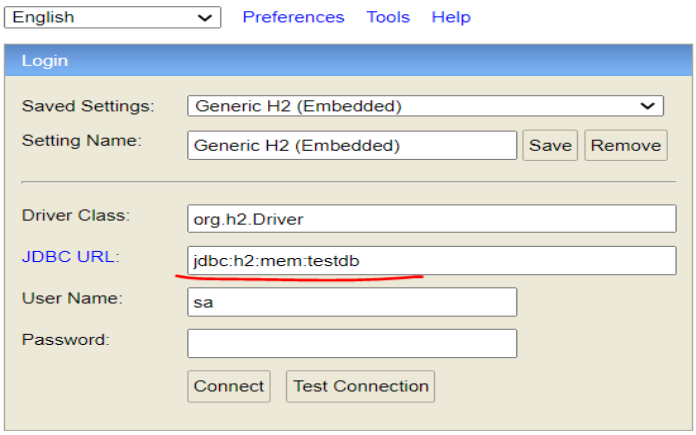


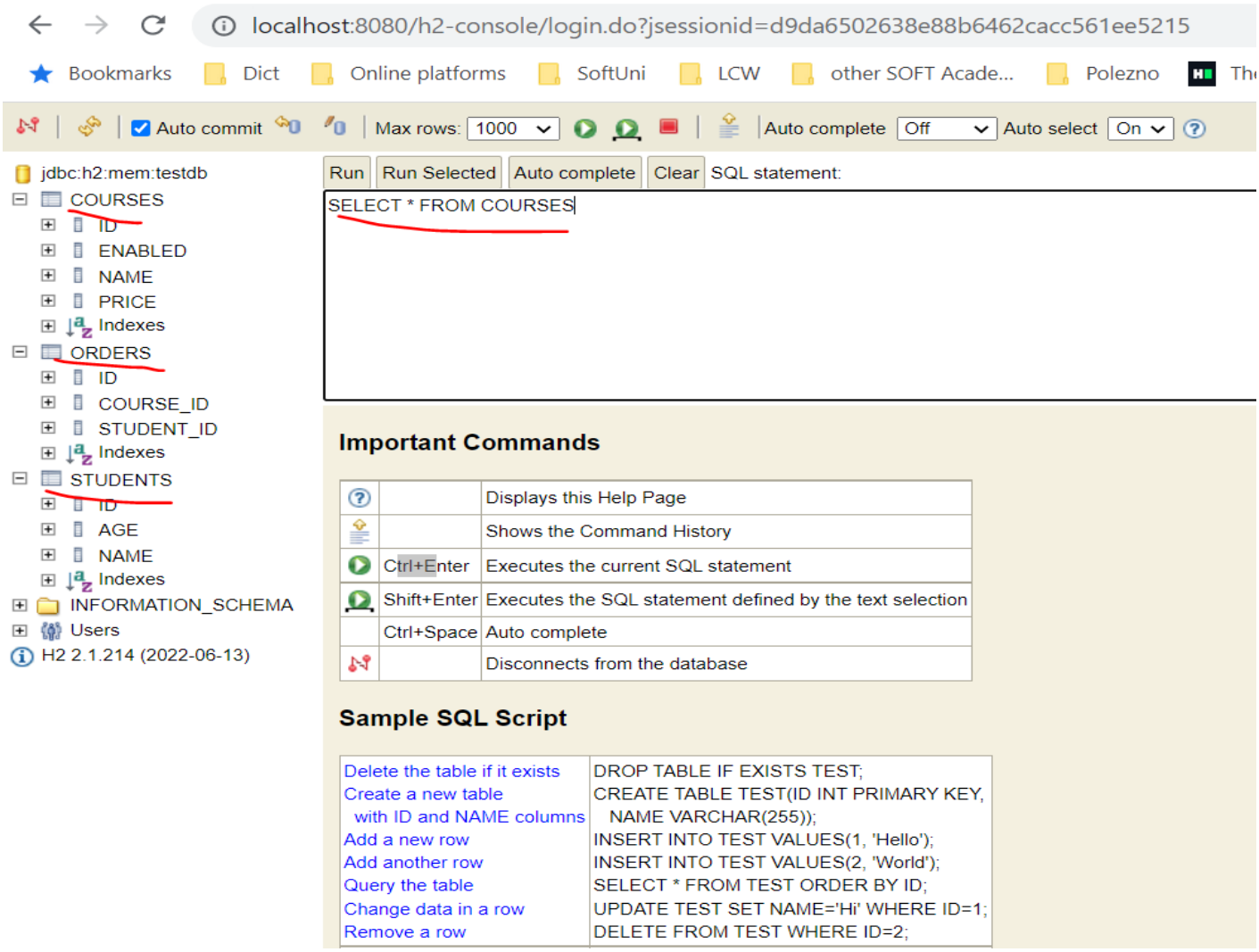

```
spring:
  datasource:
    driver-class-name: org.h2.Driver
    username: sa
    password: password
    url: jdbc:h2:mem:testdb

  h2:
    console:
      enabled: true

  jpa:
```

```
defer-datasource-initialization: true
database-platform: org.hibernate.dialect.H2Dialect
hibernate:
  ddl-auto: create
```

http://localhost:8080/h2-console







Prepare Controllers to Work

- If we implementing **RepresentationModel <T>** we can added links directly to our entity
- We need two methods from **WebMvcLinkBuilder**, that's why we must import them

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;

Main Work in Controller

Without implementing RepresentationModel<T>

```
private Link[] createStudentLinks(StudentDTO studentDTO) {
        List<Link> result = new ArrayList<>();

//        import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
        Link selfLink = linkTo(methodOn(StudentsController.class)
```

```
                .getStudentsById(studentDTO.getId())).withSelfRel();
        result.add(selfLink);

        Link updateLink = linkTo(methodOn(StudentsController.class).
                update(studentDTO.getId(), studentDTO)).withRel("update");
        result.add(updateLink);

        Link orderLink = linkTo(methodOn(StudentsController.class).
                getOrders(studentDTO.getId())).withRel("orders");
        result.add(orderLink);

        return result.toArray(new Link[0]);
    }
```

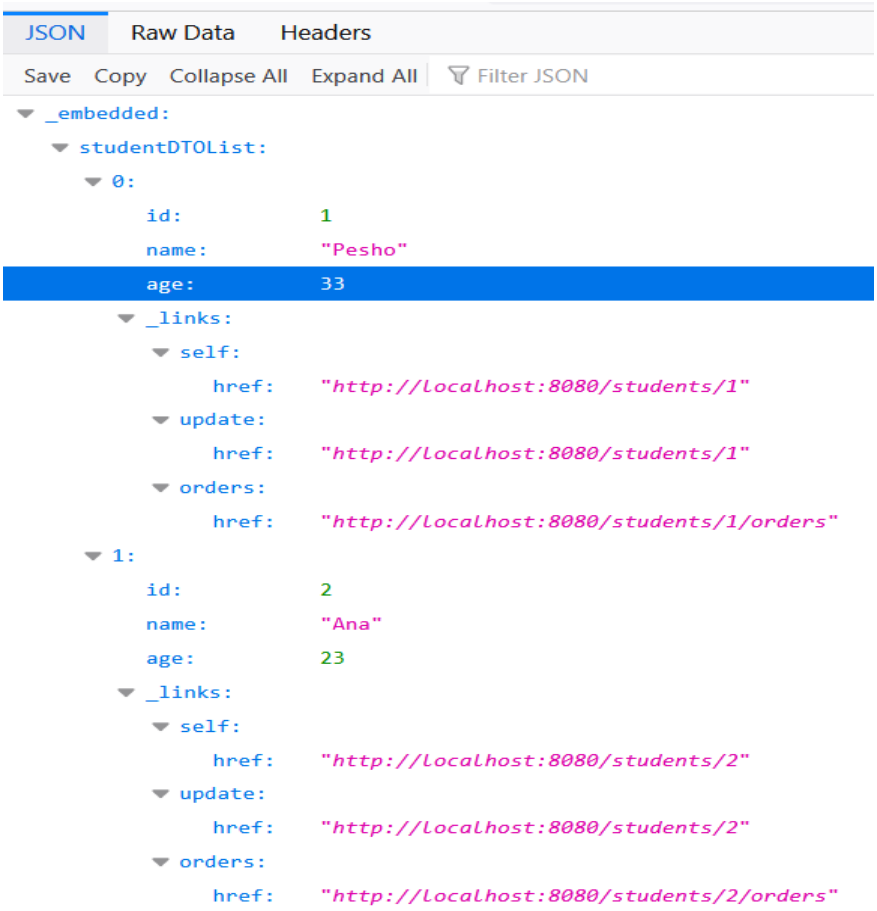
HATEOS Demo

```
import bg.softuni.hateos.mapping.StudentMapper;
import bg.softuni.hateos.model.dto.OrderDTO;
import bg.softuni.hateos.model.dto.StudentDTO;
import bg.softuni.hateos.model.entity.OrderEntity;
import bg.softuni.hateos.model.entity.StudentEntity;
import bg.softuni.hateos.repository.StudentsRepository;
import org.springframework.hateoas.CollectionModel;
import org.springframework.hateoas.EntityModel;
import org.springframework.hateoas.Link;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;

@RestController
@RequestMapping("/students")
public class StudentsController {
    private final StudentsRepository studentsRepository;
    private final StudentMapper studentMapper;

    //WARNING - Normally we never inject repos in the controllers, we do it now only - for fun
and quicker work
    public StudentsController(StudentsRepository studentsRepository, StudentMapper
studentMapper) {
        this.studentsRepository = studentsRepository;
        this.studentMapper = studentMapper;
    }

    //колекции в HATEOS - използваме CollectionModel<EntityModel<DTO>>
    @GetMapping
    public ResponseEntity<CollectionModel<EntityModel<StudentDTO>>> getStudents() {
        List<EntityModel<StudentDTO>> allStudents = studentsRepository.findAll()
                .stream()
                .map(s -> studentMapper.mapEntityToDTO(s))
                .map(dto -> EntityModel.of(dto, createStudentLinks(dto)))
```

```
                .collect(Collectors.toList());

        return ResponseEntity.ok(CollectionModel.of(allStudents));
    }

    @GetMapping("/{id}/orders")
    public ResponseEntity<CollectionModel<EntityModel<OrderDTO>>> getOrders(@PathVariable("id")
Long studentId) {
        StudentEntity studentEntity = studentsRepository
                .findById(studentId).orElseThrow();

        List<EntityModel<OrderDTO>> orders = studentEntity.getOrders()
                .stream()
                .map(o -> mapFromOrderEntityToOrderDTO(o))
                .map(dto -> EntityModel.of(dto))
                .collect(Collectors.toList());

        return ResponseEntity.ok(CollectionModel.of(orders));
    }

    private OrderDTO mapFromOrderEntityToOrderDTO(OrderEntity orderEntity) {
        return new OrderDTO().setId(orderEntity.getId())
                .setStudentId(orderEntity.getId())
                .setCourseId(orderEntity.getCourse().getId());
    }

    @PutMapping("/{id}")
    public ResponseEntity<EntityModel<StudentDTO>> update(@PathVariable("id") Long studentId,
StudentDTO studentDTO) {
        //IMPLEMENTATION NOT IMPORTAMT
        return ResponseEntity.ok().build();
    }


    @GetMapping("/{id}")
    public ResponseEntity<EntityModel<StudentDTO>> getStudentsById(@PathVariable("id") Long
studentId) {
        StudentDTO studentDTO = studentsRepository.findById(studentId)
                .map(s -> studentMapper.mapEntityToDTO(s))
                .orElseThrow();

        return ResponseEntity.ok(
                EntityModel.of(studentDTO, createStudentLinks(studentDTO))
        );
    }

    private Link[] createStudentLinks(StudentDTO studentDTO) {
        List<Link> result = new ArrayList<>();

//        import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
        Link selfLink = linkTo(methodOn(StudentsController.class)
                .getStudentsById(studentDTO.getId())).withSelfRel();
        result.add(selfLink);

        Link updateLink = linkTo(methodOn(StudentsController.class).
                update(studentDTO.getId(), studentDTO)).withRel("update");
        result.add(updateLink);
```

```
        Link orderLink = linkTo(methodOn(StudentsController.class).
                getOrders(studentDTO.getId())).withRel("orders");
        result.add(orderLink);

        return result.toArray(new Link[0]);
    }
}
```

Клиента винаги ще си вика следните URLs



Implementing RepresentationModel<T>

```
   Student student = this.studentService.findById(id);

student.add(linkTo(methodOn(StudentsController.class)
       .getStudent(student.getId())).withSelfRel());

student.add(linkTo(methodOn(StudentsController.class)
       .deleteStudent(student.getId())).withRel("delete"));

student.add(linkTo(methodOn(StudentsController.class)
        .updateStudent(student.getId(),student)).withRel("update"));
```

```
student.add(linkTo(methodOn(OrdersController.class)
        .findAllOrdersByUserId(student.getId()))
        .withRel("orders"));

        return ResponseEntity.ok(student);
```

Benefits of Using HATEOAS

- **URL structure** of the API can be **changed without affecting** clients
  - If the URL structure is changed in the service, clients will automatically pick up the new URL structure via hypermedia
- Hypermedia APIs are **explorable**
- Guiding clients toward the next step in the workflow by **providing** only the **links** that are **relevant** based on the current application state

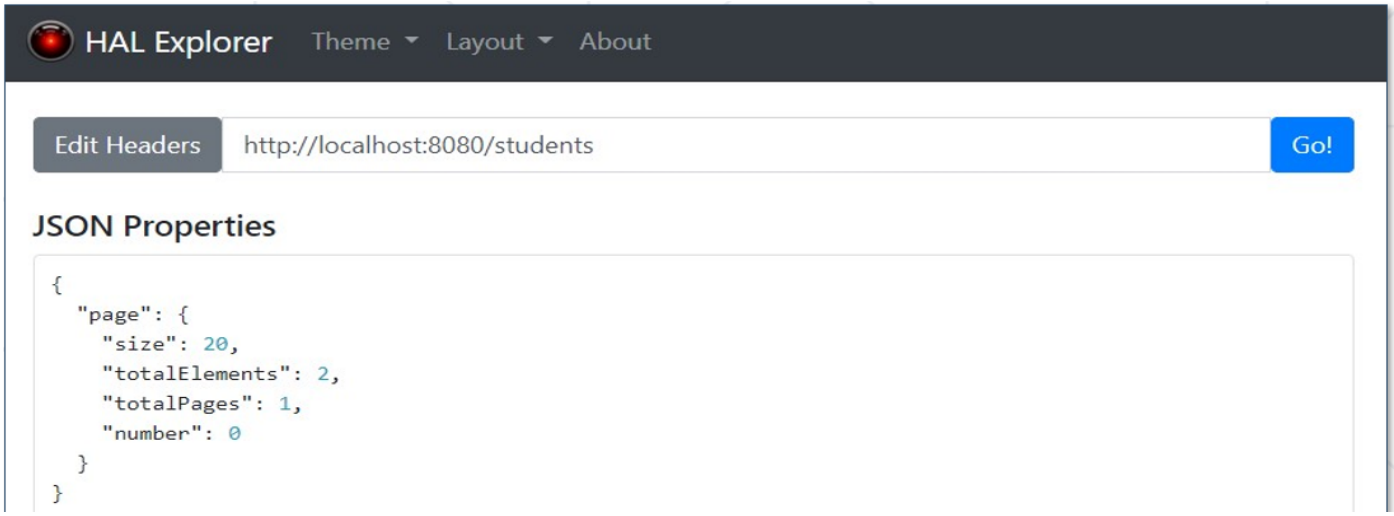
Negatives of Using HATEOAS

- Adds **extra complexity** to the API, which affects to:
  - **developer** needs to handle the **extra work** of adding links to each response
  - **more complex** to **build** and **test** than a vanilla CRUD REST API
  - **clients** also have to deal with the **extra complexity** of **hypermedia**

11.4. HAL Explorer

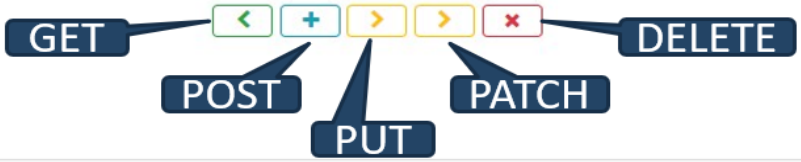
- To use **HAL Explorer** we need to add **dependency**

```
<dependency>
   <groupId>org.springframework.data</groupId>
   <artifactId>spring-data-rest-hal-explorer</artifactId>
</dependency>
```



Links

Relation	Name	Title	HTTP	Doc
self				
profile				



Embedded Resources

students

students [0]

students [1]

Response Status

200 (OK)

Response Headers

connection	keep-alive
content-type	application/hal+json
date	Mon, 06 Jul 2020 07:16:19 GMT
keep-alive	timeout=60
transfer-encoding	chunked
vary	Origin, Access-Control-Request-Method, Access-Control-Request-Headers

Response Body

```
{
  "_embedded": {
    "students": [
      {
        "name": "Gosho",
        "age": 1,
        "_links": {
          "self": {
            "href": "http://localhost:8080/students/1"
          },
          "student": {
            "href": "http://localhost:8080/students/1"
          },
          "orders": {
            "href": "http://localhost:8080/students/1/orders"
          }
        }
      },
      {
        "name": "Pesho",
        "age": 12,
```

12. Error Handling

Правим error handling-а само по един от начините. Не е добре да го смесваме.

12.1. Error Handling

Error Handling

- **Error handling** refers to:
  - The **anticipation**, **detection** and **resolution** of programming errors
  - The response & recovery procedures from error conditions
- Error handling is necessary!
  - Improves **user experience**
  - **Optimizes** debugging
  - Facilitates **code maintenance**
  - Ensures **product quality**

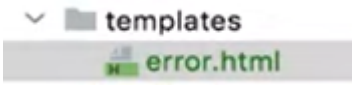
Error Handling in Spring

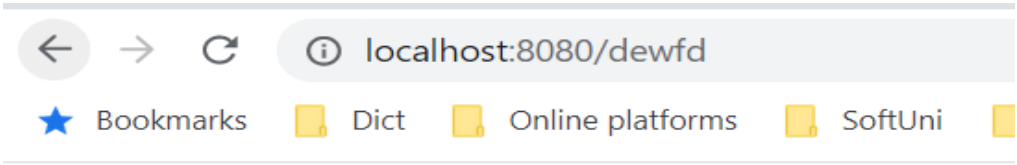
- Spring MVC offers **no default** (fall-back) **error page** out of the box, however Spring Boot does
- At start-up, Spring Boot **tries to find** a mapping for **/error**
- Spring MVC provides **several approaches** to error handling
  - Per exception
  - Per controller
  - Globally

  - Each option has its own use cases and circumstances
  - You can use:
    - **Response-annotated** custom exceptions
    - **Controller-based** handlers on specified actions
    - **@ControllerAdvise** annotated classes for global handlers

Custom error page

- To disable the default **Whitelabel error page** for a Spring Boot application:
  - We must save **error.html** file in resources/templates directory, it'll automatically be picked up by the default Spring **BasicErrorController** – Spring сам си намира default-ната страница за грешки **error.html**





My own error page

Something went wrong

ErrorController Interface – да не го правим

- Spring Boot maps **/error** to **BasicErrorController** which populates model with error attributes and then returns **'error'** as the view name
- To replace BasicErrorController with our own custom controller which can map to /error, we need to **implement ErrorController** interface

```
@Controller
public class MyErrorController implements ErrorController {
   @RequestMapping
   @ResponseBody
   public String handle(HttpServletRequest request){
      //Some code ...
   }
}
```

ErrorController е интерфейс, който няма методи няма нищо в него. Това се наричан **markup интерфейс.**

Другият вариант е да не използваме стандартния spring /error И да си направим наш:

```
server:
  error:
    whitelabel:
      enabled: false
    path: /error
```

```java
import org.springframework.boot.web.servlet.error.ErrorController;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.RequestDispatcher;
import javax.servlet.http.HttpServletRequest;

@Controller
public class ComputerStoreErrorHandler implements ErrorController {
    @RequestMapping("/error")
    public String handleError(HttpServletRequest request) {
        Object status =
                request.getAttribute(RequestDispatcher.ERROR_STATUS_CODE);

        if (status != null) {
            int statusCode = Integer.parseInt(status.toString());

            if (statusCode == HttpStatus.NOT_FOUND.value()) {
                return "errors/error-404";
            } else if (statusCode == HttpStatus.FORBIDDEN.value()) {
                return "errors/error-403";
            } else if (statusCode == HttpStatus.INTERNAL_SERVER_ERROR.value()) {
                return "errors/error-500";
            }
            //we can implement here more cases if needed
        }

        return "errors/default-error";
    }
}
```

12.2. HTTP Status Codes - Annotated Custom Exceptions

HTTP Status Codes

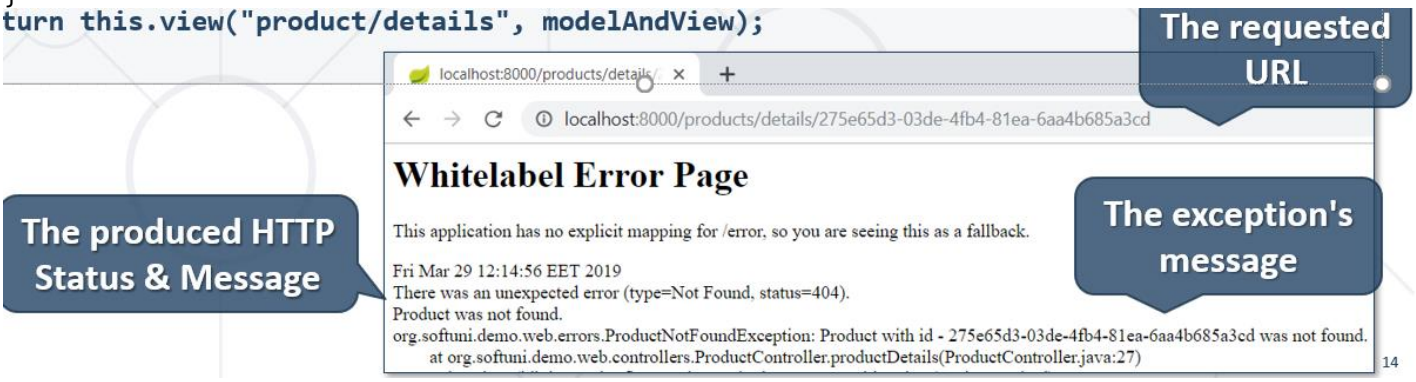
- Unhandled exceptions during a request produce HTTP 500 response
- Any custom exception can be annotated with **@ResponseStatus**
  - Supports all HTTP status codes
  - **When thrown and unhandled** – produces error page with appropriate response
  - **Било уж важно и тях да ги анотираме, дори ако ползваме @ControllerAdvice**– защото ако попадне този метод директно в Rest JSON заявка, трябвало да бъде анотиран…

```java
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Product was not found.")
public class ProductNotFoundException extends RuntimeException {
   // Exception definition
}
```

- And the controller action, throwing the exception

```java
@GetMapping("/products/details/{id}")
public ModelAndView productDetails(@PathVariable String id, ModelAndView modelAndView) {
```

```
    Product product = this.productRepository.findProductById(id);

    if(product == null) throw new ProductNotFoundException(id);

    modelAndView.addObject("product", product);
    return this.view("product/details", modelAndView);
}
```



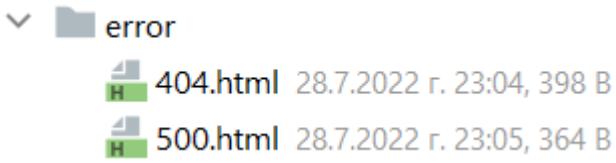
Не е добре да хвърляме stacktrace – защото издаваме информация на външни лица за нашето приложение.

```
server:
  error:
    include-stacktrace: always
```

Demo:

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Product not found!")
public class ProductNotFoundException extends RuntimeException{

}
```

```
//with Annotated Custom Exceptions
@Controller
public class ProductController1 {
    //Error 404
    @GetMapping("/products/{id}/details")
    public String showProductDetails(@PathVariable("id") String productId){
        //retreive product from repository
        //productRepository.findBy(productId).orElseThrow(new ProductNotFoundException());
        throw new ProductNotFoundException();
    }

    //Error 500
    @GetMapping("/products/{id}/error")
    public String boom(@PathVariable("id") String productid){
        //something wrong here
        //productRepository.findBy(productId).orElseThrow(new ProductNotFoundException());
        throw new NullPointerException();
    }
}
```



12.3. Controller-Based Error Handling – още един начин за handle-ване на exceptions

Controller-Based Error Handling

- You can define Controller-specific Exception Handlers
  - Annotated with **@ExceptionHandler** annotation
  - They work **only** for the **Controller** they are defined in - да
  - Can be annotated with **@ResponseStatus** to convert HTTP status
  - Can accept the **caught exception** as a **parameter**
  - Can return **ModelAndView** or **String** (view name)
  - Can catch **multiple** exception types

```
@ExceptionHandler({PersistenceException.class, TransactionException.class})
public ModelAndView handleDbExceptions(DatabaseException e) {    //Parent Exception
    ModelAndView modelAndView = new ModelAndView("error");
    modelAndView.addObject("message", e.getMessage());

    return modelAndView;
}
```

```
<html>
<head>...</head>
<body>
<h1>An error occurred while processing your request!</h1>
<p th:text="|Error: ${message}|"></p>
</body>

</html>
```

- Handler methods have **flexible signatures**
  - You can pass in servlet-related objects as parameters
    - `Exception public ModelAndView handleDbExceptions(DatabaseException e)`
    - **HttpServletRequest**
    - **HttpServletResponse**
    - **HttpSession**
    - **Principal**

- The **Model** or **ModelAndView** cannot be a parameter though
  - Instead of passing it, you have to setup it inside the method
  - Nevertheless, this is not an issue because the **IoC container** would have done the same (pass an **empty instance**)

- It is not a good practice for full error **stacktraces** to be exposed

- Your users don't need to see ugly exception web-pages
- You may even have security policies which **strictly forbid** any public exception info
- Hide as **much information** as **possible** and present **User-friendly** error pages
- For **testing** purposes you may view details
    - This may need an **environment** setup ${STACK_TARCE: never}

Demo:

```
//@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Product not found!") - когато
използваме Controller-based error handling този ред не важи
public class ObjectNotFoundException extends RuntimeException{
    private final Long productId;

    public ObjectNotFoundException(Long productId) {
        super("Cannot find object with id " + productId);
        this.productId = productId;
    }

    public Long getProductId() {
        return productId;
    }
}

//Controller-Based Error Handling
@Controller
public class ProductController2 {
    @GetMapping("/products/{id}/details")
    public String showProductDetails(@PathVariable("id") Long productId){
        //retreive product from repository
        //productRepository.findBy(productId).orElseThrow(new ProductNotFoundException());
        throw new ObjectNotFoundException(productId);
    }

    //В ProductController2   е   дефиниран @ExceptionHandler, и затова работи
    @ExceptionHandler({ObjectNotFoundException.class})
    public ModelAndView handleDbExceptions(ObjectNotFoundException e) {    //Parent Exception
        ModelAndView modelAndView = new ModelAndView("product-not-found");
        modelAndView.addObject("productId", e.getProductId());
        //@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Product not found!") - когато
използваме Controller-based error handling този ред не важи
        //затова задаваме тук http статуса на отговора
        modelAndView.setStatus(HttpStatus.NOT_FOUND);

        return modelAndView;
    }
}

//Controller-Based Error Handling
@Controller
public class OrdersController {
    @GetMapping("/orders/{id}/details")
    public String showProductDetails(@PathVariable("id") Long orderId){
        throw new ObjectNotFoundException(orderId);
    }

    //В OrdersController не е дефиниран @ExceptionHandler, и затова не работи handle-ването на
```

грешката
}

12.4. Global Application Exception Handling - @ControllerAdvice Classes

Менаджиране на грешки от много на брой контролери

Global Exception Handling

- There is a way to achieve Global exception handling in Spring
  - This is done through the **@ControllerAdvise** annotation
- Any class annotated with **@ControllerAdvise** turns into an **interceptor-like** controller:
  - Enables **global exception handling**
  - Enables **model enhancement** methods

- In **@ControllerAdvice** classes you still use **@ExceptionHandler**
  - However, this time it refers to the whole application
  - The error handling is not limited only to a specific controller

```
@ControllerAdvice
public class GlobalExceptionHandler {
public class ErrorInfo {
    public final String url;
    public final String ex;
    public ErrorInfo(String url, Exception ex) {
        this.url = url;
        this.ex = ex.getLocalizedMessage();
    }
}
}
```

Demo:

```
//Global exception handling
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler()
    public ModelAndView handleDbExceptions(ObjectNotFoundException e) {
        ModelAndView modelAndView = new ModelAndView("object-not-found");
        modelAndView.addObject("objectId", e.getObjectId());
         modelAndView.setStatus(HttpStatus.NOT_FOUND);

//         modelAndView.addObject("stack", {...} /* Formatted Stack Trace */);

        return modelAndView;
    }
}

//@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Product not found!") - когато
използваме Global controller error handling този ред също не важи
public class ObjectNotFoundException extends RuntimeException{
    private final Long objectId;
```

```
    public ObjectNotFoundException(Long objectId) {
        super("Object with id " + objectId + " not found!");
        this.objectId = objectId;
    }

    public Long getObjectId() {
        return objectId;
    }
}


@Controller
public class ProductsController {
    @GetMapping("/products/{id}/details")
    public String showProductDetails(@PathVariable("id") Long objectId){
        throw new ObjectNotFoundException(objectId);
    }
}

@Controller
public class OrdersController {
    @GetMapping("/orders/{id}/details")
    public String showProductDetails(@PathVariable("id") Long orderId){
        throw new ObjectNotFoundException(orderId);
    }
}
```

Global Exception Handling (REST)

- **RESTful requests** may also generate unexpected exceptions
  - HTTP Error response codes are a good choice
  - However sometimes you might need more than just a status
    - **Customized Error Object**, which can be presented on the Client
    - **Limited Information** returned to the Client

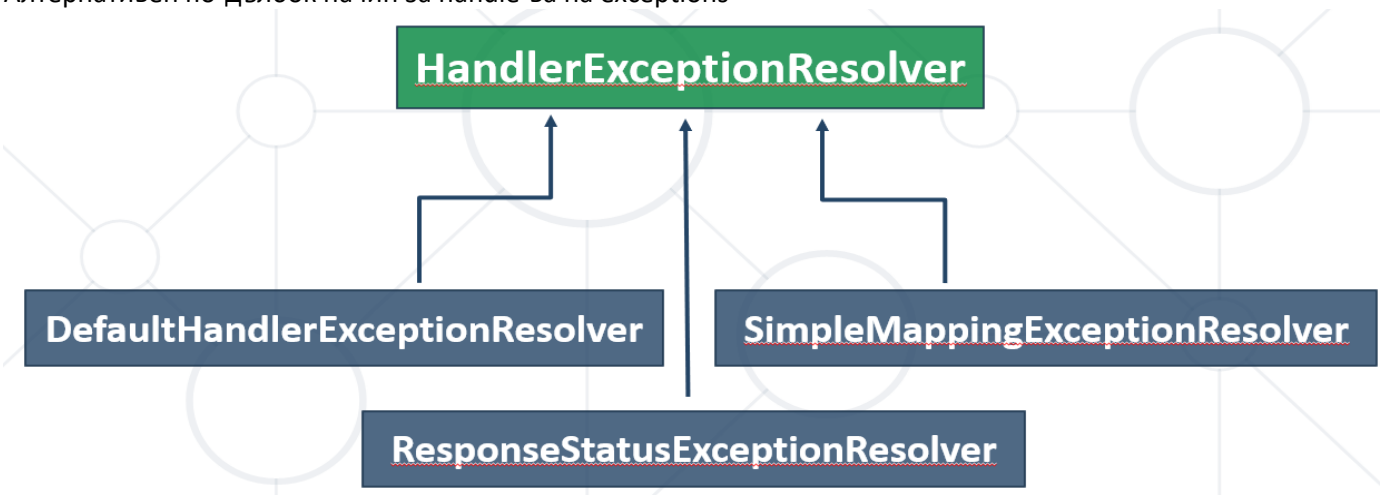
- You can customize the **Error Response** by introducing a class
  - The **Error Handler** itself remains the same as in casual web apps

```
public class ErrorInfo {
    public final String url;
    public final String ex;
    public ErrorInfo(String url, Exception ex) {
        this.url = url;
        this.ex = ex.getLocalizedMessage();
    }
}

@ControllerAdvice
public class GlobalRESTExceptionHandler {
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler({TransactionException.class, PersistenceException.class})
    public @ResponseBody
    ErrorInfo handleRESTErrors(HttpServletRequest req, DbException e) {
```

```
        return new ErrorInfo(req.getRequestURL(), ex);
    }
}
```

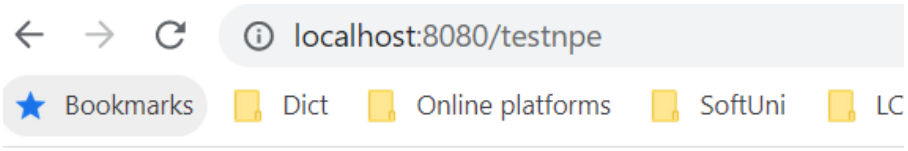
HandlerExceptionResolver Interface

Алтернативен по-дълбок начин за handle-ва на exceptions



Demo SimpleMappingExceptionResolver

```
@Configuration
public class ErrorConfig {

  @Bean
  public SimpleMappingExceptionResolver simpleMappingExceptionResolver() {
    SimpleMappingExceptionResolver resolver = new SimpleMappingExceptionResolver();

    Properties properties = new Properties();
    properties.setProperty(NullPointerException.class.getSimpleName(), "npe");

    resolver.setExceptionMappings(properties);

    return resolver;
  }
}
@Controller
public class TestController {

  @GetMapping("/testnpe")
  public String testNPE() {
    if (true) {
      throw new NullPointerException("npe");
    }
    return "npe";
  }
}
```



Ups, we threw NPE!

12.5. Exception Techniques Use Cases

What to Use When?

- Spring offers **many** choices, when it comes to **error** handling
- Be **careful mixing** too **many** of these – хващаме единия подход в целия проект и само с него
  - You may not get the behavior you wanted
- There are some semantics, that should be followed, though

Exception techniques use cases

- For custom exceptions, consider adding **@ResponseStatus** to then
- For Controller-specific exceptions, **@ExceptionHandler** methods should be added alongside the actions
- For all other exceptions, **@ExceptionHandler** methods in **@ControllerAdvice** classes should be implemented

13. Events in Spring

13.1. Scheduling Tasks

Scheduling Tasks

Нещо се тригерира като е включен app-а, без да е необходимо някой user да се е логнал. Изпълнява се анонимно, без да има Principal и логнат user.

- **Scheduling** is a process of executing the tasks for the **specific time** period
- Spring Boot provides a good support to write a scheduler on the Spring applications
- We can specify the time period by different ways:
  - Using **Cron**
  - Using **Fixed Rate**
  - Using **Fixed Delay**

Enable Scheduling

- The **@EnableScheduling** annotation is used to **enable** the **scheduler** for your application.
- This annotation should be added into the main Spring Boot application class file.

```
@SpringBootApplication
@EnableScheduling
public class MyApp {
  public static void main(String[] args) {
    SpringApplication.run(MyApp.class, args); } }
```

Scheduled Task Using Cron

- Java **Cron expressions** are used to configure the instances of **CronTrigger**
- The *cron* expression consists of **six fields**:

<second><minute><hour><day-of-month><month><day-of-week>

```
@Scheduled(cron = "0 5 * * * ?")    //each hour on the 5th minute
public void showTimeWithCron(){
  System.out.println(LocalDateTime.now());
}
```

Scheduled Task Using Fixed Rate

- **Fixed Rate** scheduler is used to execute the tasks at the **specific time**
- It **does not wait** for the completion of **previous task**
- The values should be in **milliseconds**

```
@Scheduled(fixedRate = 5000)
public void showTimeWithFixedRate() {
  System.out.println(LocalDateTime.now());
}
```

Scheduled Task Using Fixed Delay

- **FixedDelay** is the time between tasks
- The **initialDelay** is the time after which the task will be executed the first time after the initial delay value
- It **wait** for the completion of **previous task**

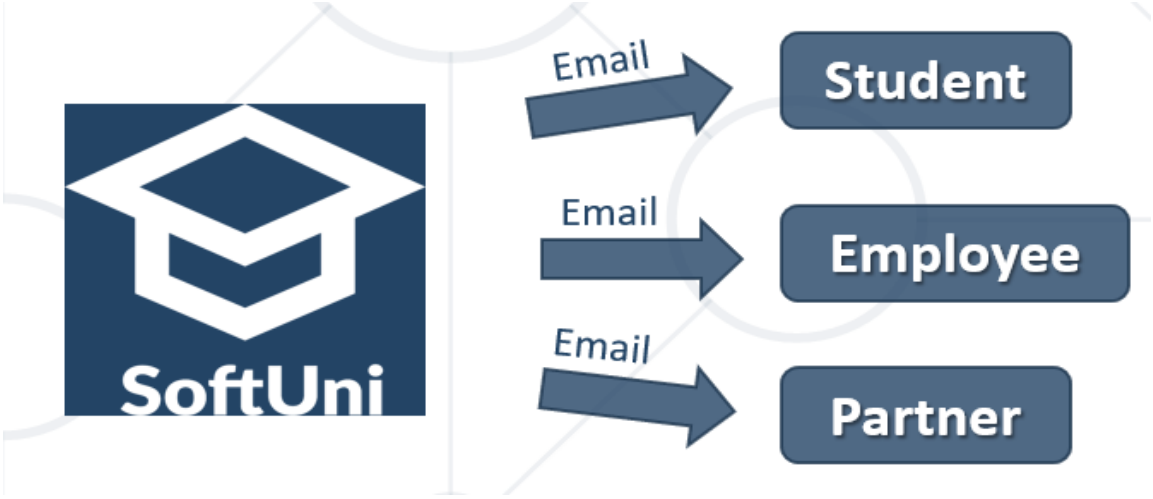
```
@Scheduled(fixedDelay = 5000, initialDelay = 10000)
public void showTimeWithFixedDelay() {
  System.out.println(LocalDateTime.now());
}
```

13.2. What Are the Events

Observer Pattern in JAVA

- Observer pattern is a **behavioral pattern**
- Provides **one object** with a loosely coupled method of **informing multiple objects** of property changes

СофтУни fire-ва event, реализирайки loose coupling – единия студент не знае дали/какво е получил другия студент от СофтУни.



Events in Spring

- The core of Spring is the **ApplicationContext (all beans, components, services, repositories)**, which manages the complete **life cycle** of the beans
- The ApplicationContext **publishes** certain types of **events** when **loading** the beans
- Spring's event handling is **single-threaded** so if an event is published, until all the receivers get the message, the **processes** are **blocked** and the flow will not continue

По време на start-up или shut-down на сървъра, то бихме могли да прихванем тези събития.

Spring Build-in Events – които се публикуват от ApplicationContext-а

- **ContextRefreshedEvent**
  - published when the ApplicationContext is either initialized/refreshed
- **ContextStartedEvent**
  - published when the ApplicationContext is started using the **start()**
- **ContextStoppedEvent**
  - published when the ApplicationContext is stopped using the **stop()**
- **ContextClosedEvent**
  - published when the ApplicationContext is closed using the **close()**
- **RequestHandledEvent**
  - Web-specific event telling all beans that an HTTP request has been serviced

13.3. Listening for Events

Listening for Events

- There are ways to listens for events in Spring:
  - **Implement** the **ApplicationListener** interface
    - Which has just one method **onApplicationEvent()**
  - Use **@EventListener()**
    - Annotate on a method

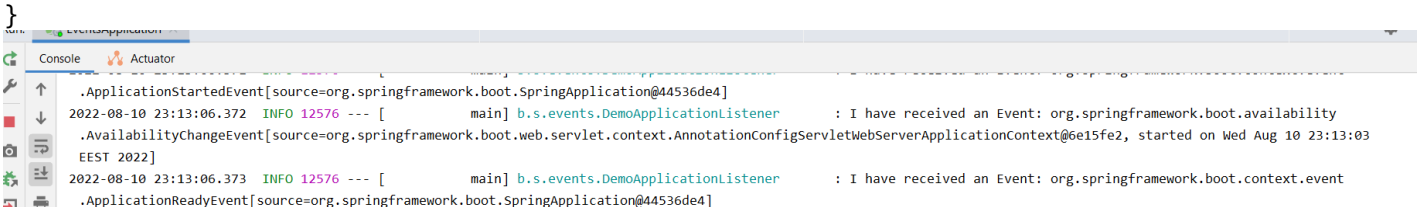
- Some of the events are **published too early** for a listener to be found via annotations and the application context. Then you must **register them** in the Spring Application instance

Demo using ApplicationListeners

- Implementing **ApplicationListener** interface

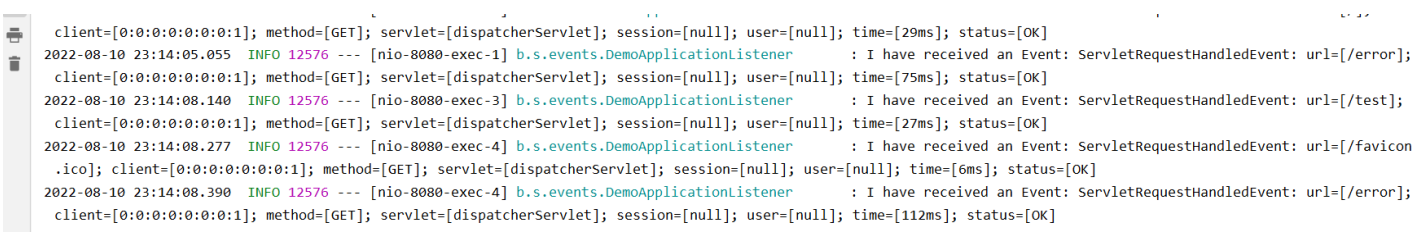
Catching all events

```
@Component
public class DemoApplicationListener implements ApplicationListener {
    private Logger LOGGER = LoggerFactory.getLogger(DemoApplicationListener.class);

    @Override
    public void onApplicationEvent(ApplicationEvent event) {
        LOGGER.info("I have received an Event: {}", event);
    }
}
```



Catching specific events – for the RestController for example

```
@RestController
public class DemoController {

    @GetMapping("/test")
    public String test(){
        return "test";
    }
}

@Component
public class DemoApplicationListener implements ApplicationListener<ServletRequestHandledEvent>
{
    private Logger LOGGER = LoggerFactory.getLogger(DemoApplicationListener.class);

    @Override
    public void onApplicationEvent(ServletRequestHandledEvent event) {
        LOGGER.info("I have received an Event: {}", event);
    }
}
```



Demo using @EventListener annotation

- Use **@EventListener()** with specific event class

```
@RestController
public class DemoController {

    @GetMapping("/test")
    public String test(){
        return "test";
    }
}

@Component
public class Demo2ApplicationListener {
    private Logger LOGGER = LoggerFactory.getLogger(Demo2ApplicationListener.class);

@Order(1) – ако имаме няколко event-listener-и, то контролираме кой event-listener да се изпълни първи, кой втори и т.н.
    @EventListener(ServletRequestHandledEvent.class)
    public void onApplicationEvent(ServletRequestHandledEvent event) {
        LOGGER.info("I have received an Event: {}", event);
    }
}
```

Listening for Multiple Events using @EventListener annotation

- Use **@EventListener(classes = {EventOne.class, EventTwo.class})** to listen for multiple events

```
@EventListener(classes = {MyEventOne.class, MyEventTwo.class})
public void handleTwoEvents(){
    System.out.println("Listens for two events!");
}
```

Register Events in Spring Application

- **Remember** that for some event is published too early for a listener to be found and needs to **be registered/added**

```
@SpringBootApplication
SpringApplication springApp = new SpringApplication(DemoForCustomEventsApplication.class);

        springApp.addListeners(new MyEventsClass());
        springApp.run(args);
        ...
```

Using Lambda When Registering Listener

- Using **lambda expressions** with specific event class

```
@SpringBootApplication
public class DemoForCustomEventsApplication {
    public static void main(String[] args) {
      SpringApplication springApp = new SpringApplication(DemoForCustomEventsApplication.class);
```

```
      springApp.addListeners((ApplicationContextInitializedEvent e) -> {
System.out.println("App context init event"); });
        springApp.run(args);
    }
}
```

Transaction Bound Events

Events-и, които са интегрирани със SpringData:

- The listener of an event to a **phase** of the **transaction**
- Transaction **phases :**
  - **AFTER_COMMIT** - The default, used to fire the event if the transaction has **completed successfully**
  - **AFTER_ROLLBACK** - when transaction has **rolled back**
  - **AFTER_COMPLETION** - when transaction has **completed**
  - **BEFORE_COMMIT** - used to fire the event right **before** transaction **commit**

- An example of Transaction Bound Event, that will fire before transaction commit

```
@TransactionalEventListener(phase = TransactionPhase.BEFORE_COMMIT)
public void transactionEventListener (MyCustomEvent event) {
    System.out.println("Hit before transaction commit!");
}
```

13.4. Creating Custom Event

Най-често можем да си създадем custom event, който да си го прихванем

Creating Custom Event

- To create and publish our custom event, there is some steps that we need to follow:
  - **Create** our custom **event class** that **extends ApplicationEvent** class
  - **Create publisher**, that publish our new event
  - **Add event listener**, that listens for our new event

Create Our Custom Event Class

- Create our event class, that extends ApplicationEvent

```
public class OrderCreatedEvent extends ApplicationEvent {
    private final String orderId;

    public OrderCreatedEvent(Object source, String orderId) {
        super(source);
        this.orderId = orderId;
    }

    public String getOrderId() {
        return orderId;
    }
}
```

Create Publisher

- Create a publisher that publish our custom event and inject in him the ApplicationEventPublisher object

```
@Component
public class MyPublisher {
    @Autowired // It is better to inject in constructor
    private ApplicationEventPublisher appEventPublisher;

    public void publishEvent(String message) {
        MyCustomEvent myEvent = new MyCustomEvent(this, message);
        appEventPublisher.publishEvent(myCustomEvent);
    }
};
```

Или само може да го inject-нем, като можем да го inject-нем и в Service класа

```
@Service
public class OrderService {
    private static final Logger LOGGER = LoggerFactory.getLogger(OrderService.class);
    private final ApplicationEventPublisher eventPublisher;

    @Autowired
    public OrderService(ApplicationEventPublisher eventPublisher) {
        this.eventPublisher = eventPublisher;
    }

    public void createOrder(String productId, int quantity) {
        LOGGER.info("Creating order for product {} with quantity {}.", productId, quantity);

        // TODO: do some work

        //first creating the event
        OrderCreatedEvent orderCreatedEvent = new OrderCreatedEvent(
                OrderService.class.getSimpleName(),
                productId
        );

        //then we publish the event
        eventPublisher.publishEvent(orderCreatedEvent);
        //we publish the event
        eventPublisher.publishEvent(orderCreatedEvent);
    }
}
```

```
@Controller
public class OrderController {
  private OrderService orderService;

  public OrderController(OrderService orderService) {
    this.orderService = orderService;
  }

  @GetMapping("/dummy/order/create")
  public String createOrder() {
    orderService.createOrder("3", 33);
    return "Hello!";
```

```
  }
}
```

Create Listener

- Create listeners, already explain the different ways

```
@Component
public class Listeners {
    @EventListener(MyCustomEvent.class)
    public void listener(MyCustomEvent myCustomEvent) {
        System.out.printf("Custom event listeners with message -%s!%n", myCustomEvent.getMsg());
    }
}
```

```
@Service
public class BonusPointsService {

    private static final Logger LOGGER = LoggerFactory.getLogger(BonusPointsService.class);

    @EventListener(OrderCreatedEvent.class)
    public void onOrderCreated(OrderCreatedEvent evt) {
        LOGGER.info("Adding bonus points to user for order {}", evt.getOrderId());
    }
}
```

```
@Service
public class EmailService {
    private static final Logger LOGGER = LoggerFactory.getLogger(EmailService.class);

    @EventListener(OrderCreatedEvent.class)
    public void onOrderCreated(OrderCreatedEvent evt) {
        LOGGER.info("Sending email to user for order {}", evt.getOrderId());
    }
}
```

```
2022-08-11 00:01:34.113  INFO 1396 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 4 ms
2022-08-11 00:01:34.151  INFO 1396 --- [nio-8080-exec-1] bg.softuni.events.service.OrderService   : Creating order for product 3 with quantity 33.
2022-08-11 00:01:34.152  INFO 1396 --- [nio-8080-exec-1] b.s.events.service.BonusPointsService    : Adding bonus points to user for order 3
2022-08-11 00:01:34.152  INFO 1396 --- [nio-8080-exec-1] bg.softuni.events.service.EmailService   : Sending email to user for order 3
2022-08-11 00:01:34.152  INFO 1396 --- [nio-8080-exec-1] b.s.events.service.InventoryService      : Decreasing inventory for order 3
```

13.5. Caching Data

Caching

- If you using Spring Boot, then simply use the **spring-boot-starter-cache** dependency
- Under the hood, the starter brings the spring-context-support module
- Използваме при обекти, които се създават сравнително бавно, но се използват сравнително често

```
implementation 'org.springframework.boot:spring-boot-starter-cache'
```

```
<dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

Enable Caching

- When using Spring Boot, the **@EnableCaching** annotation would register the **ConcurrentMapCacheManager**
- No need for separate Bean declaration
- Simply adding the **@EnableCaching** annotation to any of the configuration classes

```
@SpringBootApplication
@EnableCaching
public class CacheApplication {
   public static void main(String[] args) {
      SpringApplication.run(CacheApplication.class, args);
   }
}
```

```
@Configuration
@EnableCaching
public class MyConfig {
// Some configurations }}
```

@Cacheable

- Use **@Cacheable** to demarcate/разграничавам methods that are cacheable
- Result is **stored** in the **cache** and on subsequent invocations (with the same arguments), the value in the cache is returned **without** having to actually **execute** the method
- the **findAllStudents** method is associated with the cache named **students**

```
@Cacheable("students")
public List<Student> findAllStudents() { //...  }
```

Като го извикаме втори път този метод findAllStudents, то резултата го връща вече веднага, защото той вече е в cache-а на cacheManager-а на Spring.

- Custom Cache Resolution

```
@Cacheable("students", cacheManager = "myCacheManager")
public List<Student> findAllStudents() { //...  }
```

- Conditional Caching

```
@Cacheable("student", condition = "#avg > 4")
public List<Student> findStudentsByAvgScore(Double avg) {
//...
}
```

@CachePut

- When the **cache** needs to be **updated** without interfering with the method execution
- The **method** is **always executed** and its result is placed into the cache
- It supports the same options as **@Cacheable**
- **Заменя старите данни за findAll с новите данни за findAll**

```
@CachePut("students")
public List<Student> findAll() {
   //...   }
```

@CacheEvict

- This process is useful for **removing** stale or unused data from the cache
- Using the **allEntries** attribute to evict/изваждам all entries from the cache

```
@CacheEvict(cacheNames="books", allEntries=true)
public void loadStudents() {
   //...
}
```

Customize The auto-configured CacheManager

- To customize the CacheManager we must implement **CacheManagerCustomizer<ConcurentMapCacheManager>**
- Create Bean CacheManager that returns new **ConcurentMapCacheManager**

```
@Component
public class MyCacheCustomizer implements CacheManagerCustomizer<ConcurrentMapCacheManager> {
    @Override
    public void customize(ConcurrentMapCacheManager cacheM) {
        cacheM.setCacheNames(asList("students", "courses"));
    }
}
```

14. Aspect Oriented Programming (AOP)

14.1. What is AOP

Aspect Oriented Programming AOP

- **AOP** breaks the program logic into distinct parts (called **concerns**)
- **Cross-cutting concern**
    - Concern that can affect the whole application and **should be centralized in one location**, such as transaction management, authentication, logging, security etc.

Filter е само върху Web Controller-а, Interceptor е върху Spring context, а concerns AOP е само върху определен метод.
Един AOP може да се използва само в рамките на един JVM (Java Virtual Machine)!

14.2. Why We Use AOP

Why Use AOP

- To **dynamically add the additional concern** before, after or around the actual logic
- Suppose that we have to maintain methods and need to do actions before or after they are called
- We can solve the problem **with** or **without AOP**

Problem Example

- **Student class** with some methods whose activity we want to track

```
public class Student{
    public void actionOne(){...};
    public void actionTwo(){...};
    public void actionThree(){...};
    public void actionFour(){...};
    public void actionFive(){...};
}
```

Problem Solution

- **Solution without AOP**
  - If we need to log all activity of student, we need to write additional code in all tracked methods
  - It leads to the maintenance problem.

- **Solution with AOP**
  - We can define the additional concern like maintaining log, sending notification, etc. in the method of a class
  - Maintenance is easy in AOP

14.3. AOP Concepts and Terminology

Terminologies

- The AOP **concepts** and **terminologies** are
  - Join point
  - Advice
  - Pointcut
  - Introduction
  - Target Object
  - Aspect
  - **Interceptor**
  - AOP Proxy
  - Weaving

Join Point

- **Join point**
  - A Join point is **any point in your program** such as method execution, exception handling, field access etc.
  - We can have many Join points
  - Spring supports **only the method** execution join point

Advices and Types

- Represents an action taken by an aspect at a join point
  - **Before Advice**: it executes before a join point
  - **After Returning Advice**: it executes after a joint point completes normally
  - **After Throwing Advice**: it executes if method exits by throwing an exception
  - **After Advice**: it executes after a join point regardless of join point exit whether normally or exceptional return
  - **Around Advice**: It executes before and after a join point

Pointcut, Introduction, Target Object

- **Pointcut**
  - It is an expression language of AOP that matches join points – мога да опиша някакъв метод
- **Introduction**
  - Introduction of additional method and fields for a type – действието, при което се вкарват допълнителни методи в някакъв клас
- **Target Object**
  - The object i.e. being advised by one or more aspects
  - Also known as **Proxied Object**

Aspect, Interceptor, AOP Proxy, Weaving

- **Aspect**
  - A class that contains advices
- **Interceptor**
  - An aspect that contains only one advice
- **AOP Proxy**
  - Used to implement aspect contracts, created by AOP framework
- **Weaving**
  - The process of linking aspect with other application types or objects to create an advised object.

14.4. Spring AOP AspectJ Annotations

Gradle or Maven

```
implementation 'org.springframework.boot:spring-boot-starter-aop'
```

Spring AOP AspectJ

- The 3 ways to use spring AOP are
  - By Spring 1.2 old style
  - By AspectJ annotation-style
    - The widely used approach is Spring AspectJ Annotation Style
  - By Spring XML configuration-style(schema based)

- There are two ways to use Spring AOP AspectJ implementation
  - By annotation – метода ще се изпълни на който и да е метод от класа Student

```
@Aspect
public class LoggingAspect {
    @Before("execution(* Student.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        ...
}}
```

  - By XML Configuration

```
<!-- Aspect -->
<bean id="logAspect" class="" />
<aop:config>
 <aop:aspect id="aspectLoggging" ref="logAspect" >
  <!-- @Before -->
  <aop:pointcut id="pointCutBefore"
    expression="execution(* Student.*(..))" />
  <aop:before method="logBefore" pointcut-ref="pointCutBefore" />
 </aop:aspect>
</aop:config>
```

AspectJ Annotations in Spring

- **@Aspect**
  - Declares the class as aspect
- **@Pointcut**
  - Declares the pointcut expression
- **@Before**
  - Declares the before advice
  - Applied before calling the actual method
- **@After**
  - Declares the after advice
  - Applied after calling the actual method and before returning result
- **@AfterReturning**
  - Declares the after returning advice
  - Applied after calling the actual method and before returning result, can get the result value in the advice
- **@Around**
  - Declares the around advice
  - Applied before and after calling the actual method
- **@AfterThrowing**
  - Declares the throws advice
  - Applied if actual method throws exception

@Pointcut

- Pointcut is an **expression language** of Spring AOP
- **@Pointcut** annotation is used to define the pointcut
- We can also **refer the pointcut expression by name**

Pointcut Expressions – език за описване на сигнатури на методи

- Applied on all the public methods

```
@Pointcut("execution(public * *(..))")
```

- Applied on all methods of Student class

```
@Pointcut("execution(* Student.*(..))")
```

- Applied on all setter methods of Student class

```
@Pointcut("execution(* Student.set*(..))")
```

- Applied on all methods of class that returns an int value

```
@Pointcut("execution(int Student. *(..))")
```

14.5. Examples

Prepare for AOP

```
public class Student{
    public void actionOne(){...};
    public void actionTwo(){...};
    public void actionThree(){...};
    public void actionFour(){...};
    public void actionFive(){...};
}
```

Create Aspect Class

- We need to create a class with **@Aspect**, that contains all advices

```
@Aspect
@Configuration
public class TrackStudent{
    @Pointcut("execution(* Student.*(..))")
    public track(){}

    //Can have more than one pointcuts
    //Here place all advices
}
```

@Before Example

- Add **before advice** to our TrackStudent class

```
@Aspect
@Configuration
public class TrackStudent {
```

```
    @Pointcut("execution(* Student.*(..))")
    public track(){}
    @Before("track()") // Execute before track pointcut
    public void beforeAdvice(JoinPoint joinPoint){
        System.out.println("Before advice executed");
    }
}
```

@After Example

- Add after advice to our TrackStudent class

```
@Aspect
@Configuration
public class TrackStudent {
    @Pointcut("execution(* Student.*(..))")
    public track(){} //for reuse below

    @After("track()") // Execute after track pointcut
    public void afterAdvice(JoinPoint joinPoint){
        System.out.println("After advice executed");
    }
}
```

@AfterReturning Example

- Add after returning advice to our TrackStudent class – при успешно изпълнение на метода

```
...
@AfterReturning(pointcut = "execution(* Student.action())", returning = "result")
public void afterReturning(JoinPoint joinPoint, Object result) {
        System.out.println("AfterReturning advice executed");
        //In AfterReturning we can get the result of pointcut
        }
...
```

@Around Example

- Add **around advice** to our TrackStudent class - //around - и преди и след изпълнението на метода

```
@Around("track()")
public Object aroundAdvices(ProceedingJoinPoint pjp) throws Throwable {
    System.out.println("Before calling");
    Object obj = pjp.proceed();   //We need to pass the pjp references in the advice
method, so that we can proceed the request by calling the proceed method
    System.out.println("After calling");
}
```

@AfterThrowing Example

- Add after throwing advice to our TrackStudent class

```
@AfterThrowing(pointcut = "execution(* Student.action())", throwing = "error")
Public void afterReturning(JoinPoint joinPoint, Throwable error){
        System.out.println("AfterReturning advice executed");
        System.out.println("Exception is: " + error);
        //In AfterThrowing we can get the exception
}
```

Specifying Aspects Ordering

- There are two ways:
  - By annotation

```
@Aspect
@Order(0)
public class TrackStudent{//...}
```

  - By implementing interface

```
@Aspect
public class TrackStudent implements Ordered {
    //Override this method
    public int getOrder(){ return 0; }
}
```

14.6. Demos

Demo1 – става извън класа IncredibleMachine

За сменяне на кой Aspect файл да ни е активен ако имаме testing и production например.
Application.yml

```
sample1:
  enabled: false
sample2:
  enabled: true
```

```
@Aspect
@Component
@ConditionalOnProperty(value = "sample1.enabled", havingValue = "true")
public class Sample1Aspect {
    private static final Logger LOGGER = LoggerFactory.getLogger(Sample1Aspect.class);

    @Pointcut("execution(* bg.softuni.aop.IncredibleMachine.*(..))")
    void onAllIncredibleMachineMethods() {}   //служи за последващо използване

    @Pointcut("execution(* bg.softuni.aop.IncredibleMachine.echo(..))")
    void onEchoCalled() {
    }

    @AfterThrowing(pointcut = "execution(* bg.softuni.aop.IncredibleMachine.boom())",
            throwing = "error")
    public void afterThrowing(JoinPoint joinPoint, Throwable error) {
        LOGGER.error("Ups, I think that the method {} threw Exception and the exception is...",
                joinPoint.getSignature(), error);
    }

    @Before("onAllIncredibleMachineMethods()")
    public void beforeEachMethod(JoinPoint joinPoint) {
        LOGGER.info("Before calling method {} with arguments {}",
                joinPoint.getSignature(),
```

```
                Arrays.asList(joinPoint.getArgs()));
    }
}


@Component
@ConditionalOnProperty(value = "sample1.enabled", havingValue = "true")
public class Sample1AspectDemo implements CommandLineRunner {
    private static final Logger LOGGER = LoggerFactory.getLogger(Sample1AspectDemo.class);

    private final IncredibleMachine incredibleMachine;

    public Sample1AspectDemo(IncredibleMachine incredibleMachine) {

        this.incredibleMachine = incredibleMachine;
    }

    @Override
    public void run(String... args) throws Exception {
        incredibleMachine.saySomething();
        incredibleMachine.echo("AOP!!!!");

        try {
            incredibleMachine.boom();
        } catch (Exception exc) {
            LOGGER.info("Exception from boom called!");
        }

        LOGGER.info(incredibleMachine.concat("Hello, ", "world!"));
    }
}


@Component
public class IncredibleMachine {
    private static final Logger LOGGER = LoggerFactory.getLogger(IncredibleMachine.class);

    public void saySomething() {
        LOGGER.info("I'm saying something!");
    }

    public void boom() {
        throw new NullPointerException("Ups, I did something wrong!");
    }

    public void echo(String whatToEcho) {
        LOGGER.info("I'm echoing this {}", whatToEcho);
    }

    public String concat(String a, String b) {
        return a + b;
    }
}
```

Demo2

```
@Aspect
@Component
@ConditionalOnProperty(value = "sample2.enabled", havingValue = "true")
public class Sample2Aspect {
    private static final Logger LOGGER = LoggerFactory.getLogger(Sample2Aspect.class);

    @Pointcut("execution(* bg.softuni.aop.IncredibleMachine.concat(..))")
    void onConcat() {
    }

    @Around(value = "onConcat() && args(a, b)") //around - и преди и след изпълнението на метода
    public String onConcat(ProceedingJoinPoint pjp, String a, String b) throws Throwable {
        // Before the execution of concat
        LOGGER.info("The on concat method was called with arguments [{}] and [{}].", a, b);

        var modifiedA = "(" + a + ")";
        var modifiedB = "(" + b + ")";

        // execute the method
        var methodResult = pjp.proceed(new Object[]{modifiedA, modifiedB});

        // modify the result
        return "[" + methodResult + "]";
    }
}

@Component
@ConditionalOnProperty(value = "sample2.enabled", havingValue = "true")
public class Sample2AspectDemo implements CommandLineRunner {
    private static final Logger LOGGER = LoggerFactory.getLogger(Sample2AspectDemo.class);

    private final IncredibleMachine incredibleMachine;

    public Sample2AspectDemo(IncredibleMachine incredibleMachine) {
        this.incredibleMachine = incredibleMachine;
    }

    @Override
    public void run(String... args) throws Exception {
        LOGGER.info(incredibleMachine.concat("Hello, ", "world!"));
    }
}
```

Demo3

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface TrackLatency {
    String latency();
}


@Aspect
@Component
public class Latency {
```

```
    @Around(value = "@annotation(TrackLatency)") //around - и преди и след изпълнението на метода
    public Object trackLatency(ProceedingJoinPoint pjp, TrackLatency TrackLatency) throws
Throwable {
        String latencyId = TrackLatency.latency();
        DateTimeFormatter formatterToString = DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm:ss");

        StopWatch stopWatch = new StopWatch();
        stopWatch.start();
        Object obj = pjp.proceed();
        stopWatch.stop();

        long actualLatency = stopWatch.getLastTaskTimeMillis();
        FileWriter myWriter = new FileWriter("src/main/java/mainPackage/logs/logFile.log",
true);
        myWriter.write(String.format("%s The latency for %s is: %dms%n",
LocalDateTime.now().format(formatterToString), latencyId, actualLatency));
        myWriter.close();
        return obj;   //for rest json consuming scenarios
    }
}
```

```
@TrackLatency(latency = "fix order")
@PostMapping("/fix/{id}")
@PreAuthorize("hasRole('ROLE_BACK_OFFICE')")
public String fixOrderConfirm(@PathVariable Long id, @Valid
@ModelAttribute("orderFixBindingModel") OrderFixBindingModel orderFixBindingModel,
                              BindingResult bindingResult, RedirectAttributes
redirectAttributes) throws IOException {

}
```

15. Unit testing and integration testing

Не трябва да тестваме passwordencodep - идва от външна библиотека и самата външна библиотека е тествана вече от други хора.
Методите от repository-то не трябва да се тестват.
Database init unit тестове няма нужда също!!!
Column definition = “TEXT” не му харесва на in-memory базата данни – дали се разрешава с @Lob???
Spring менъджира с коя версия на in-memory базата данни HyperSQL Database или H2 ще работи!!!
Затова нарочно не слагаме версия на in-memory базата данни!!!

```
Important notes before starting testing:
  - first, disable in the class AppSeedInit.java  the @PostConstrict annotated method
beginInit()
  - second, copy the real CLOUDINARY_SECRET in the application.yml in the test section // or
other option is to set Enviromental Variables for every test class manually

For testing - do not use columnDefinition @Column(name = "more_info", columnDefinition = "TEXT")
- (in the ItemEntity class for field moreInfo, I removed the columnDefinition so that the in-
memory HyperSQL grammar is satisfied)
```

```
<!-- https://mvnrepository.com/artifact/org.hsqldb/hsqldb -->
<dependency>
    <groupId>org.hsqldb</groupId>
    <artifactId>hsqldb</artifactId>
    <scope>test</scope>
</dependency>
```

15.1. Unit Testing

Unit Testing

- **Unit Testing**
  - A level of software testing where **individual components are tested**
  - The purpose is to validate that **each unit performs as designed**
  - The **lowest level of software testing**
  - Often isolated in order to ensure individual testing

Mocking

- Software practice, primarily used in **Unit Testing**
  - An object under test may have **dependencies** on other objects
  - To **isolate** the behavior, the other objects are replaced
    - The replacements are **mocked objects**
    - The mocked objects **simulate** the behavior of the **real objects**

Benefits

- Unit testing **increases confidence** in **changing** / **maintaining code**
- Development is faster:
  - Verifying the correctness of new functionality is not manual
  - Localizing bugs, introduced in development is much faster
- The code is modular and reusable (necessary for Unit testing)

15.2. Unit Testing a Web Application

Unit Testing

- **Unit Testing** for web apps is similar to the unit tests we've done
  - Writing test methods to test classes and methods (functionalities)
    - Testing individual code components (**units**)
    - Independently from the **infrastructure**
  - You still use the same testing frameworks as in casual unit testing

- When using a web frameworks such as **Spring MVC**
  - Built-in logic does not need to be tested
    - It is already tested during the development of the framework itself
  - You still need to test your custom functionality

- **Web applications** also need testing for:
  - Controllers

    - Services
    - Custom Components etc.

- Different **components** of the application are tested differently
    - They are tested on different levels
        - **Unit** testing
        - **Integration** testing
        - **End-to-End** testing – тестваме директно в Browser-а – например със Selenium или с Playwright Chromium
- Every component of the application must be tested

15.3. Unit Testing the Service layer

- Testing a simple service with mocking in an **Spring MVC** app

```
@ExtendWith(MockitoExtension.class)
@Mock
```

JUnit5

```
@ExtendWith(MockitoExtension.class)
public class AppUserDetailsServiceTest {
    private UserEntity testUser;
    private UserRoleEntity adminRole, customerRole;
    private AppUserDetailsService testAppUserDetailsService;

    @Mock
    private UserRepository mockUserRepository;

    @BeforeEach
    void init() {
        testAppUserDetailsService = new AppUserDetailsService(mockUserRepository);

        adminRole = new UserRoleEntity().setUserRole(UserRoleEnum.ADMIN);
        customerRole = new UserRoleEntity().setUserRole(UserRoleEnum.CUSTOMER);

        testUser = new UserEntity()
                .setFirstName("Svilen")
                .setLastName("Velikov")
                .setUsername("admin")
                .setEmail("svilkata@abv.bg")
```

```
                .setPassword("11111")
                .setUserRoles(Set.of(adminRole, customerRole));
    }

    @Test
    void testUserNotFound() {
        Assertions.assertThrows(
                UsernameNotFoundException.class, //expected error class exception
                () -> testAppUserDetailsService.loadUserByUsername("invalid_username"));
    }

    @Test
    void testUserFound() {
        //Arrange
        Mockito.when(mockUserRepository.findByUsername(testUser.getUsername()))
                .thenReturn(Optional.of(testUser));

        //Act
        UserDetails actual =
testAppUserDetailsService.loadUserByUsername(testUser.getUsername());

        //Assert
        Assertions.assertEquals(actual.getUsername(), testUser.getUsername());

        String actualRoles = actual.getAuthorities().stream().map(ga -> ga.getAuthority())
                .collect(Collectors.joining(", "));
        String expectedRoles = "ROLE_ADMIN, ROLE_CUSTOMER";
        Assertions.assertEquals(expectedRoles, actualRoles);
    }
}
```

15.4. Integration Testing the Web Layer / the Controller

Testing Controller Examples

With Integration Tests!!!

@SpringBootTest
@AutoConfigureMockMvc

@Autowire
@WithMockUser
@MockBean

MockMvcResultMatchers Methods

- **request()**
  - Access to request-related assertions
- **handler()**

        - Access to assertions for the handler that handled the request
    - **model()**
        - Access to model-related assertions
    - **view()**
        - Access to assertions on the selected view
    - **flash()**
        - Access to flash attribute assertions
    - **status()**
        - Access to response status assertions
    - **header()**
        - Access to response header assertions
    - **content()**
        - Access to response body assertions

!!! Simple test examples

```
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerTests {
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void when_getOneStudents_returnFirst() throws Exception {
        mockMvc
                .perform(MockMvcRequestBuilders.get("/users/1"))
                .andExpect(status().isOk())
                .andExpect(view().name("one"))
                .andExpect(model().attributeExists("user"));
```

```
@SpringBootTest
@AutoConfigureMockMvc
public class AuthorsControllerTest {
    // @Autowired MockMvc and AuthorRepository
    @BeforeEach
    public void setUp() { // Add two test authors in repository }
        @AfterEach
        public void tearDown() { authorRepository.deleteAll(); }

        @Test
        public void testGetAuthorsCorrect() throws Exception {
            this.mockMvc.perform(get("/authors")).
                    andExpect(status().isOk()).
                    andExpect(jsonPath("$", hasSize(2))).
                    andExpect(jsonPath("$.[0].name", is(author1Name))).
                    andExpect(jsonPath("$.[1].name", is(author2Name)));  }
```

```
mockMvc.perform(get( urlTemplate: "/api/" + routeId + "/comments")).
    andExpect(status().isOk()).
    andExpect(jsonPath( expression: "$", hasSize(2)));
}
```

- Testing with MockUser

Кой user има право да достъпва дадената операция
Можем и целият клас да го анотираме с @WithMockUser

```
@Test
@WithMockUser(username = "Plamen", roles = {"FRONT_OFFICE"})
void orderReceive() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get(
            FRONT_OFFICE_CONTROLLER_PREFIX + "/receive", orderId))
            .andExpect(status().isOk())
            .andExpect(view().name("/orders/orders-receive"))
            .andExpect(model().attributeExists("OrderReceiveBindingModel"));
}
```

15.5. Testing Essentials

Testing

- There are also different concepts and practices of test development
  - **Code-first** approach (The usual Development)
  - **Test-first** approach (Test-Driven Development)

- Each has its own **advantages** and **disadvantages**
  - The **Code-first** approach ensures **flexibility** & **fast** development
  - The **Code-first** approach requires **additional refactoring**
  - The **Test-first** approach ensures **quality** and **edge case coverage**
  - The **Test-first** approach is **complicated** and is an **"initial delay"**

Common levels of Software Testing

- Some of the most common levels of Software Testing

Testing Level	Description
Unit Testing	Tests Individual components of code, independent from the infrastructure
Component Unit Testing	Testing of multiple functionalities (a single component)
Integration Testing	Testing of all integrated modules to verify the combined functionality
System Testing	Tests the system as a whole, once all the components are integrated
Regression Testing	Testing that recent program or code change has not adversely affected existing features.
Acceptance Testing	Tests if the product meets the client's requirements. Purely done by QAs

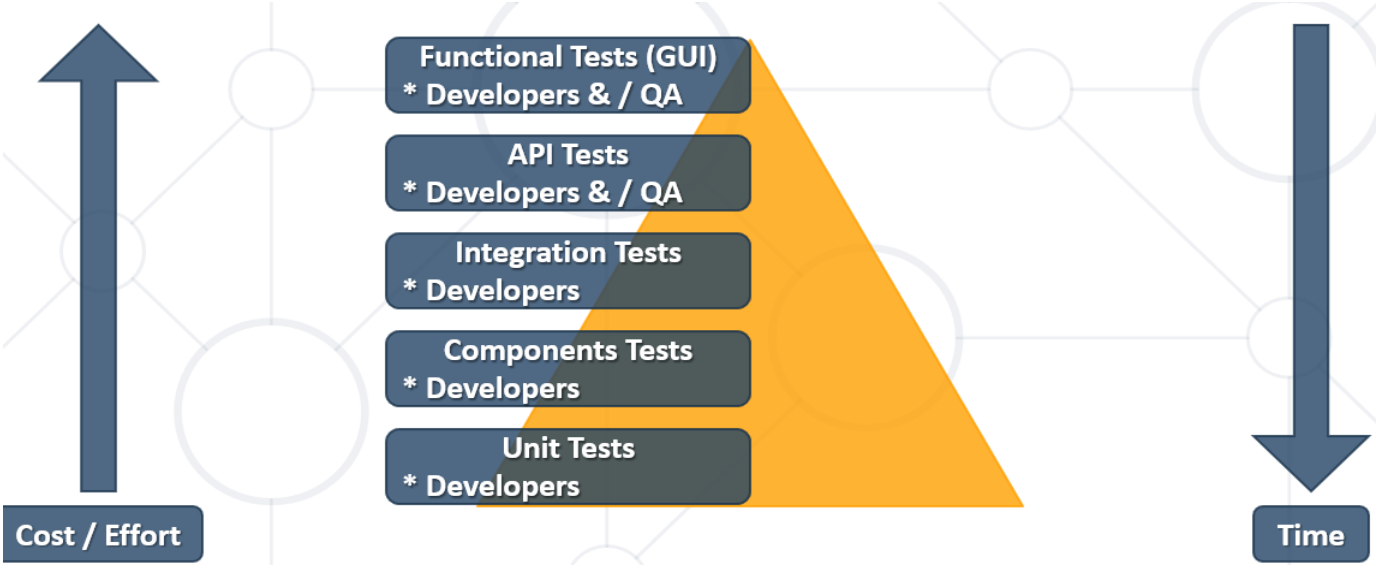
Load / Stress Testing	Test the application's limits by attempting large data processing and introducting abnormal circumstances and conditions (edge cases)
Security Testing	Test if the application has any security flaws and vulnerabilities
Other Types of Testing	Manual, automation, UI, performance, black box, end-to-end testing, etc.

Testing

- Unit testing ensures the correctness of a particular unit
  - Not testing all components may lead to false results
    - A single unit may function correctly, independent of the infrastructure
  - Combining components and testing them collectively is necessary
  - Every level of testing is essential to an application's lifecycle

Different Testing levels

- Different Testing levels require different time and resources



15.6. Advanced Testing examples

Като има много методи за тестване, и когато пуснем целия клас да се тества, то се помни контекста, и трябва или да изтриваме базата с **@AfterEach (ако успеем да я изтрием разбира се)**, или в нов тестов клас да добавяме някои тестове.

Винаги да използваме **@BeforeEach – за да се инициализира преди всеки тестметод от тесткласа!!! Иначе дава бъгове!!!**

Страшно много се работи с Reflection при тестването – когато не можем да заложим цели обекти в теста, то залагаме имена на полетата на този клас, имената/полетата на ProductItemTypeBindingDTO например.

В тестовете за CloudinaryServiceTest и PictureServiceTest съм използвал само @Mock, но мокнатите неща ги бутам в конструктора на по-горния service//компонент/bean. И тествам на по-горния service някой от методите.

А в случая с PictureControllerTest, аз не използвам на PictureController метод, който да тествам, т.е. не използвам и конструктор PictureController. И за да инжектираме мокнатия service, използваме този път @MockBean

@mock е част от Mockito фреймуъркът и няма нищо общо със спринг. С `@mock` създаваш един мокнат клас и толкоз. В случая го ползваш в някакъв локален незиползваем контролер, който също не бива извикан от mockMvc.

`@MockBean` от своя страна е спринг анотация. Като я използваш, създаваш един мок клас и с него заменяш съответния компонент в инициализирания спринг аппликейшън контекст (това е мястото, където "живеят" всички спринг бийнове - компоненти, сървиси, репозиторита и т.н.). Т.е. вместо истинския Bean имаш мокнат бийн.

```
import bg.softuni.computerStore.exception.MyFileDestroyFromCloudinaryException;
import bg.softuni.computerStore.exception.MyFileUploadException;
import bg.softuni.computerStore.model.entity.picture.CloudinaryImage;
import com.cloudinary.Cloudinary;
import com.cloudinary.Uploader;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.mock.web.MockMultipartFile;

import java.io.File;
import java.io.IOException;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.Mockito.when;

@ExtendWith(MockitoExtension.class)
class CloudinaryServiceTest {
    @Mock
    private Cloudinary cloudinary;

    @Test
    public void upload_must_return_correct_cloudirary_image() throws IOException {
        // Arrange
        Uploader mockUploader = Mockito.mock(Uploader.class);
        when(cloudinary.uploader()).thenReturn(mockUploader);
        when(mockUploader.upload(any(File.class), any(Map.class)))
                .thenReturn(Map.of("url", "success_url", "public_id", "success_id"));
        CloudinaryService service = new CloudinaryService(cloudinary);
```

```
        // Act
        CloudinaryImage result = service.upload(new MockMultipartFile("FileName", new byte[0]));

        // Assert
        assertEquals("success_url", result.getUrl());
        assertEquals("success_id", result.getPublicId());
    }

    @Test
    public void upload_must_throw_exception_when() throws IOException {
        // arrange
        Uploader mockUploader = Mockito.mock(Uploader.class);
        when(cloudinary.uploader()).thenReturn(mockUploader);
        when(mockUploader.upload(any(File.class), any(Map.class)))
                .thenThrow(new IOException());
        CloudinaryService service = new CloudinaryService(cloudinary);

        // act & assert
        assertThrows(MyFileUploadException.class,
                () -> service.upload(new MockMultipartFile("FileName", new byte[0])),
                "Can not upload file 'FileName'");
    }

    @Test
    public void deleteFromCloudinaryTestMustDeleteAnImageFromCloudinary() throws IOException {
        // Arrange
        Uploader mockUploader = Mockito.mock(Uploader.class);
        when(cloudinary.uploader()).thenReturn(mockUploader);
        when(mockUploader.destroy(anyString(), any(Map.class)))
                .thenReturn(Map.of("url", "success_url", "public_id", "success_id"));
        CloudinaryService service = new CloudinaryService(cloudinary);

        // Act
        boolean isDeleted = service.deleteFromCloudinary("1");

        // Assert
        assertEquals(true, isDeleted);
    }

    @Test
    public void deleteFromCloudinaryTestMustThrowExceptionWhenDeleteAnImageFromCloudinary()
throws IOException {
        // Arrange
        Uploader mockUploader = Mockito.mock(Uploader.class);
        when(cloudinary.uploader()).thenReturn(mockUploader);
        when(mockUploader.destroy(anyString(), any(Map.class)))
                .thenThrow(new IOException());
        CloudinaryService service = new CloudinaryService(cloudinary);

        // Act & Assert
//        boolean isDeleted = service.deleteFromCloudinary("1");
        assertThrows(MyFileDestroyFromCloudinaryException.class,
                () -> service.deleteFromCloudinary("1"),
                "Error deleting from cloudinary a file with publicId '1'");
    }
}
```

```java
//@ExtendWith(MockitoExtension.class)
@AutoConfigureTestDatabase
@SpringBootTest
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class PictureServiceTest {
    @Autowired
    private PictureService pictureService;

    private PictureEntity testPictureEntity1, testPictureEntity2;
    private ItemEntity testItemEntity;
    private String testPhotoPublicId = "abcd";

    //Mocked objects
    @Mock
    private CloudinaryService mockedCloudinaryService;
    @Mock
    private PictureRepository mockedPictureRepository;
    @Mock
    private AllItemsRepository mockedAllItemsRepository;
    @Mock
    private Cloudinary mockedCloudinary;
    @Mock
    private Uploader mockedUploader;

    @BeforeAll
    public void setup() throws IOException {
        //Arrange
        testPictureEntity1 = (new PictureEntity()
                .setId(1L)
                .setPublicId(testPhotoPublicId)
                .setUrl("bala")
                .setItemId(1L));

        testPictureEntity2 = (new PictureEntity()
                .setId(2L)
                .setPublicId("efgh")
                .setUrl("bala")
                .setItemId(2L));

        // Arrange
        mockedUploader = Mockito.mock(Uploader.class);
        when(mockedCloudinary.uploader()).thenReturn(mockedUploader);
        when(mockedUploader.upload(any(File.class), any(Map.class)))
                .thenReturn(Map.of("url", "success_url", "public_id", "success_id"));
        when(mockedUploader.destroy(anyString(), any(Map.class)))
                .thenReturn(Map.of("url", "success_url", "public_id", "success_id"));
        this.mockedCloudinaryService = new CloudinaryService(mockedCloudinary);

        this.pictureService = new PictureService(mockedPictureRepository,
mockedAllItemsRepository, mockedCloudinaryService);
    }

//    @Order(1)
    @Test
    void createPictureEntityTestSuccessfull() {
        //More to arrange
```

```
        MockMultipartFile mockedMultipartFile = new MockMultipartFile(
                "file",
                "hello.txt",
                MediaType.TEXT_PLAIN_VALUE,
                "Hello World".getBytes());

        //Act
        PictureEntity createdPictureEntity =
                this.pictureService.createPictureEntity(mockedMultipartFile, 8L);

        //Assert
        assertEquals(createdPictureEntity.getItemId(), 8L);
    }

//    @Order(2)
    @Test
    void getPictureByPublicIdTestSuccessfull() {
        //Act
        PictureEntity expected = testPictureEntity1;

//        when(mockedPictureRepository.findPictureEntityByPublicId(testPhotoPublicId))
//                .thenReturn(Optional.ofNullable(testPictureEntity1));

doReturn(Optional.of(testPictureEntity1)).when(mockedPictureRepository).findPictureEntityByPubli
cId(testPhotoPublicId);

        PictureEntity result = this.pictureService.getPictureByPublicId(testPhotoPublicId);

        //Assert
        assertEquals(expected.getPublicId(), result.getPublicId());
//        assertThrows(NoSuchElementException.class,
//                () -> this.pictureService.getPictureByPublicId(testPhotoPublicId));
    }

//    @Order(3)
    @Test
    void deleteFromPictureRepositoryTestSuccessfull() {
        this.pictureService.deleteFromPictureRepository(testPhotoPublicId);
    }

//    @Order(4)
    @Test
    void savePhotoTestSuccessfullWhenPictureIsPresent() {
//        when(mockedPictureRepository.findPictureEntitiesByItemId(1L))
//                .thenReturn(Optional.ofNullable(testPictureEntity1));

doReturn(Optional.of(testPictureEntity1)).when(mockedPictureRepository).findPictureEntitiesByIte
mId(1L);

        when(mockedPictureRepository.save(testPictureEntity1))
                .thenReturn(testPictureEntity1);

        testItemEntity = new ComputerEntity();
        testItemEntity
                .setItemId(1L)
                .setBrand("AK 47")
                .setModel("Naj-dpbria")
```

```
                .setCurrentQuantity(3)
                .setBuyingPrice(BigDecimal.valueOf(12))
                .setSellingPrice(BigDecimal.valueOf(18))
                .setCreationDateTime(LocalDateTime.now());

//        when(mockedAllItemsRepository.findById(1L))
//                .thenReturn(Optional.ofNullable(testItemEntity));
        doReturn(Optional.of(testItemEntity)).when(mockedAllItemsRepository).findById(1L);

        this.pictureService.savePhoto(testPictureEntity1);
    }
}


import bg.softuni.computerStore.service.UserService;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import static
org.springframework.security.test.web.servlet.request.SecurityMockMvcRequestPostProcessors.csrf;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@SpringBootTest
@AutoConfigureMockMvc
@AutoConfigureTestDatabase
class AdminControllerTest {
    private static final String ADMIN_CONTROLLER_PREFIX = "/pages/admins";

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private UserDetailsService appUserDetailsService;
    @Autowired
    private UserService userService;

    @BeforeEach
    public void setup() {
        this.userService.init();
        loginUser("admin");
    }

    private void loginUser(String username) {
```

```
    //The login process of user with username "admin"  doing it below
    UserDetails userDetails =
            appUserDetailsService.loadUserByUsername(username);

    Authentication authentication =
            new UsernamePasswordAuthenticationToken(
                    userDetails,
                    userDetails.getPassword(),
                    userDetails.getAuthorities()
            );

    SecurityContextHolder.
            getContext().
            setAuthentication(authentication);
}

@AfterEach
void clear() {

}

@Test
@Order(1)
@WithMockUser(username = "admin", roles = {"ADMIN"})
void addEmployeeRoleTest() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get(
                    ADMIN_CONTROLLER_PREFIX + "/set-user-role"))
            .andExpect(view().name("/user/add-or-edit-user-role"))
            .andExpect(model().attributeExists("userRolesBindingDTO"))
            .andExpect(status().isOk());
}


@Test
@Order(2)
@WithMockUser(username = "admin", roles = {"ADMIN"})
void addUserRoleConfirmTestSuccessfull() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/set-user-role")
                    .param("username", "purchase")
                    .param("roles", "EMPLOYEE_PURCHASES", "EMPLOYEE_SALES")
                    .with(csrf()))
            .andExpect(status().is2xxSuccessful());
}

@Test
@Order(3)
@WithMockUser(username = "admin", roles = {"ADMIN"})
void addUserRoleConfirmTestWhenOnlyOneRoleSelected() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/set-user-role")
                    .param("username", "purchase")
                    .param("roles", "EMPLOYEE_SALES")
                    .with(csrf()))
            .andExpect(status().is3xxRedirection())
            .andExpect(redirectedUrl("/pages/admins/set-user-role"))
            .andExpect(flash().attribute("atLeastTwoRolesShouldBeSelected", true));
}
```

```
@Test
@Order(4)
@WithMockUser(username = "admin", roles = {"ADMIN"})
void addUserRoleConfirmTestWhenAdminRoleIsPresent() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/set-user-role")
                    .param("username", "purchase")
                    .param("roles", "ADMIN", "EMPLOYEE_PURCHASES")
                    .with(csrf()))
            .andExpect(status().is3xxRedirection())
            .andExpect(redirectedUrl("/pages/admins/set-user-role"))
            .andExpect(flash().attribute("oneAdminOnlyPossible", true));
}

@Test
@Order(5)
@WithMockUser(username = "admin", roles = {"ADMIN"})
void addUserRoleConfirmTestWhenEmployeeNotSelected() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/set-user-role")
                    .param("username", "")
                    .with(csrf()))
            .andExpect(status().is3xxRedirection())
            .andExpect(redirectedUrl("/pages/admins/set-user-role"))
            .andExpect(flash().attribute("employeeNotSelected", true));
}


@Test
@Order(6)
@WithMockUser(username = "admin", roles = {"ADMIN"})
void statisticsHttpRequestsTest() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get(
                            ADMIN_CONTROLLER_PREFIX + "/statshttprequests")
                    .with(csrf()))
            .andExpect(status().isOk())
            .andExpect(view().name("/stats/stats-httprequests"))
            .andExpect(model().attributeExists("stats"));

}

@Test
@Order(7)
@WithMockUser(username = "admin", roles = {"ADMIN"})
void statisticsSalesTest() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get(
                            ADMIN_CONTROLLER_PREFIX + "/statssales")
                    .with(csrf()))
            .andExpect(status().isOk())
            .andExpect(view().name("/stats/stats-sales"))
            .andExpect(model().attributeExists("stats"));
}

@Test
@Order(8)
@WithMockUser(username = "admin", roles = {"ADMIN"})
void changeAdminUserTest() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get(
                            ADMIN_CONTROLLER_PREFIX + "/change-admin-user")
```

```
                        .with(csrf()))
                .andExpect(status().isOk())
                .andExpect(view().name("/user/change-admin-user-role"))
                .andExpect(model().attributeExists("employees"))
                .andExpect(model().attributeExists("userRolesBindingDTO"));
    }


    @Test
    @Order(9)
    @WithMockUser(username = "admin", roles = {"ADMIN"})
    void changeAdminUserConfirmTestWhenEmployeeNotSelected() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/change-admin-
user")
                        .param("username", "")
                        .param("roles", "")
                        .with(csrf()))
                .andExpect(status().is3xxRedirection())
                .andExpect(redirectedUrl("/pages/admins/change-admin-user"))
                .andExpect(flash().attribute("employeeNotSelected", true));
    }

    @Test
    @Order(10)
    @WithMockUser(username = "admin", roles = {"ADMIN"})
    void changeAdminUserConfirmTestWhenLessThanFourRoles() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/change-admin-
user")
                        .param("username", "sales")
                        .param("roles", "EMPLOYEE_PURCHASES", "ADMIN")
                        .with(csrf()))
                .andExpect(status().is3xxRedirection())
                .andExpect(redirectedUrl("/pages/admins/change-admin-user"))
                .andExpect(flash().attribute("adminChanging", true));
    }


    @Test
    @Order(11)
    @WithMockUser(username = "admin", roles = {"ADMIN"})
    void registerNewEmployeeTest() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get(
                        ADMIN_CONTROLLER_PREFIX + "/register-new-employee"))
                .andExpect(view().name("/user/registerNewEmployee"))
                .andExpect(model().attributeExists("employeeRegistrationModel"))
                .andExpect(status().isOk());
    }

    @Test
    @Order(12)
    @WithMockUser(username = "admin", roles = {"ADMIN"})
    void registerNewEmployeeConfirmTestSuccessfull() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/register-new-
employee")
                        .param("username", "Tisho")
                        .param("email", "xadqw@dqd.com")
                        .param("firstName", "Tihomir")
```

```
                        .param("lastName", "Tihomirov")
                        .param("password", "1234")
                        .param("roles", "EMPLOYEE_PURCHASES", "CUSTOMER")
                        .with(csrf()))
                .andExpect(status().is3xxRedirection());
    }

    @Test
    @Order(13)
    @WithMockUser(username = "admin", roles = {"ADMIN"})
    void registerNewEmployeeConfirmTestCreatingEmployeeHasErrors() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/register-new-
employee")
                        .param("username", "Tisho")
                        .param("email", "")
                        .param("firstName", "")
                        .param("lastName", "Tihomirov")
                        .param("password", "1234")
                        .param("roles", "EMPLOYEE_PURCHASES", "CUSTOMER")
                        .with(csrf()))
                .andExpect(status().is3xxRedirection())
                .andExpect(redirectedUrl("/pages/admins/register-new-employee"));
    }

    @Test
    @Order(14)
    @WithMockUser(username = "admin", roles = {"ADMIN"})
    void registerNewEmployeeConfirmTestLessThanTwoRoles() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post(ADMIN_CONTROLLER_PREFIX + "/register-new-
employee")
                        .param("username", "Tisho")
                        .param("email", "Tifwf@dqwed.com")
                        .param("firstName", "Tihomir")
                        .param("lastName", "Tihomirov")
                        .param("password", "1234")
                        .param("roles", "EMPLOYEE_PURCHASES")
                        .with(csrf()))
                .andExpect(status().is3xxRedirection())
                .andExpect(flash().attribute("atLeastTwoRolesShouldBeSelected", true))
                .andExpect(redirectedUrl("/pages/admins/register-new-employee"));
    }
}
```

```
@SpringBootTest
@AutoConfigureTestDatabase
@AutoConfigureMockMvc
@WithMockUser(username = "purchase", roles = {"EMPLOYEE_PURCHASES"})
public class PictureControllerTest {
    private static final String PURCHASE_CONTROLLER_PREFIX = "/pages/purchases";
    private Long itemId = 1L;

    @Autowired
    private MockMvc mockMvc;
    @Autowired
```

```
    private UserDetailsService appUserDetailsService;
    @Autowired
    private UserService userService;

    private PictureBindingModel pictureBindingModel;
    private MockMultipartFile mockedMultipartFile;

    @MockBean
    private PictureService mockedPictureService;

    @BeforeEach
    public void setup() {
        //Arrange
        this.userService.init();
        loginUser("purchase");

        //Arrange more
        //Залагаме име на файла името на полето "picture" от PictureBindingModel -
        // за да може reflection-а да си го вземе multipart обекта правилно. Яко.
        mockedMultipartFile = new MockMultipartFile(
                "picture",
                "hello.txt",
                MediaType.TEXT_PLAIN_VALUE,
                "Hello World".getBytes());

        pictureBindingModel = new PictureBindingModel().setPicture(mockedMultipartFile);

        PictureEntity pictureEntity = new PictureEntity()
                .setPublicId("public_id")
                .setUrl("url")
                .setItemId(itemId)
                .setId(1L);

        doReturn(pictureEntity).when(mockedPictureService)
                .createPictureEntity(pictureBindingModel.getPicture(), itemId);
//        when(this.mockedPictureService.createPictureEntity(pictureBindingModel.getPicture(),
itemId))
//                .thenReturn(pictureEntity);

        doNothing().when(this.mockedPictureService).savePhoto(pictureEntity);

//        pictureController = new PictureController(mockedPictureService);
    }

    private void loginUser(String username) {
        //The login process of user with username "admin"  doing it below
        UserDetails userDetails =
                appUserDetailsService.loadUserByUsername(username);

        Authentication authentication =
                new UsernamePasswordAuthenticationToken(
                        userDetails,
                        userDetails.getPassword(),
                        userDetails.getAuthorities()
                );

        SecurityContextHolder.
```

```
                getContext().
                setAuthentication(authentication);
    }

    @Test
    void addComputerPictureTest() throws Exception {
        MockHttpServletRequestBuilder builder = MockMvcRequestBuilders
                .multipart(PURCHASE_CONTROLLER_PREFIX + "/computers/" + this.itemId +
"/addpicture")
                .file(mockedMultipartFile)
                .param("itemId", this.itemId + "")
                .with(csrf());


        mockMvc.perform(builder)
                .andExpect(status().is3xxRedirection())
                .andExpect(redirectedUrl("/items/all/computer/details/" + this.itemId));
    }
```

15.7. Lazy initialization exception

```
При тестване с in-memory база данни, първата заявка работи, а втората не работи!!!
@Query("SELECT b FROM BasketOrderEntity b JOIN FETCH b.products WHERE b.id= :id")
Optional<BasketOrderEntity> findBasketByIdEager(Long id);

Optional<BasketOrderEntity> findBasketOrderEntitiesById(Long id);
```

https://www.baeldung.com/hibernate-initialize-proxy-exception

It's important to understand **what Session**, **Lazy Initialisation, and Proxy Object** are, and how they come together in the *Hibernate* framework:

- *Session* is a persistence context that represents a conversation between an application and the database.
- *Lazy Loading* means that the object won't be loaded to the *Session* context until it is accessed in code.
- Hibernate creates a dynamic *Proxy Object* subclass that will hit the database only when we first use the object.

This error occurs when we try to fetch a lazy-loaded object from the database by using a proxy object, but the Hibernate session is already closed.

@Transactional

https://www.baeldung.com/spring-transactional-propagation-isolation

Привет!

Ексепшъна, за който говорим няма много общо с базата която използваш и се хвърля от хибернейт. Релациите by default са LAZY (мързеливи):

```
@ManyToMany
private List<ItemEntity> products;
```

Това означава, че като си поискаш баскета хибернейт няма да направи JOIN с продуктите а вместо това ще ти върне един прокси обект дето реално няма продукти в него. Ако се опиташ да ги достъпиш хибернейт трябва да направи заявка в момента на достъп. Затова е и LAZY. Само че, за да се случи това нещо трябва да имаш отворена сесия, а такава няма. Можеш да подходиш по разнични начини.

- да изпълняваш функционалността в контекста на транзакция (напр. анотация `@Transactional`)
- да ползваш JOIN FETCH
- да ползваш EAGER релация
- да използваш named graph.

Има доста ресурси в нета по този въпрос. Мисълта ми е, че грешката не е свързана с базата. Може би да започнеш от тук - https://www.baeldung.com/hibernate-initialize-proxy-exception
Запознай се също и с Transactional анотацията.

29. Proxies

Demo without SPRING
Да видим как работи SPRING реално!!!
Proxy може да са използва за извършване на някакво предварително действие преди действителното извикване на метод. Нещо аналогично на interceptor реално е.

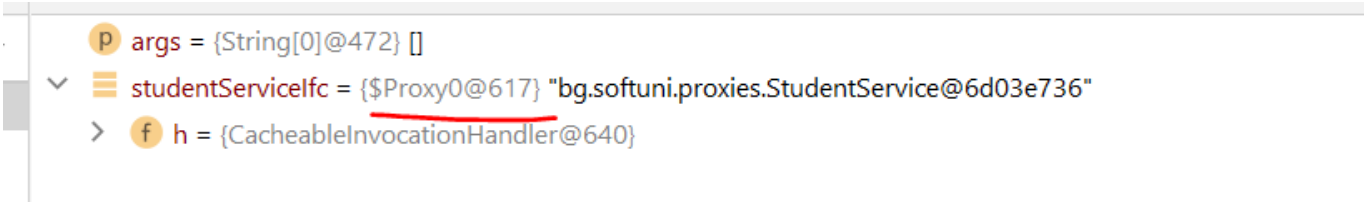
Два са видовете proxie-та:

- Динамични – работи на база на интерфейси
- CGLIB проксита – CGLIB библиотека, която прави proxy-та на база редактиране на byte кода (Spring така работи)

Когато инжектираме някъде (например в Controller) StudentService, то ние реално не го инжектираме него, а инжектираме някакво динамично proxy (което по default е Singleton bean scope). И реално ако викаме метод на StudentService, то на практика ние викаме мето на proxy-то.

Demo Proxies без Spring

```
public class Test {
    public static void main(String[] args) {
        StudentServiceIfc studentServiceIfc = getStudentService();
```




```
        System.out.println("-------");
        System.out.println(studentServiceIfc.getAllStudents());
        System.out.println("-------");
        System.out.println(studentServiceIfc.getAllStudents());
        System.out.println("-------");
    }

    private static StudentServiceIfc getStudentService() {
        return (StudentServiceIfc) Proxy.newProxyInstance(
                Test.class.getClassLoader(),
                new Class[]{StudentServiceIfc.class},  //масив от всички интерфейси, които са
имплементирани
                new CacheableInvocationHandler(new StudentService())
        );
    }
}


public interface StudentServiceIfc {
    List<StudentDTO> getAllStudents();
}


public class StudentService implements StudentServiceIfc {
    private static List<StudentDTO> allStudents = List.of(
            new StudentDTO("Pesho", 10, 20),
            new StudentDTO("Anna", 11, 21),
            new StudentDTO("Gosho", 9, 22)
    );

    @Override
    @Cacheable("students")
    public List<StudentDTO> getAllStudents() {
        System.out.println("Complex calculation of all students...");
        dummyWait();
        System.out.println("Students calculated");

        return allStudents;
    }

    private void dummyWait() {
        try {
            Thread.sleep(5000);
            // Dummy - imagine we need to perform complex operation to fetch student data...
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

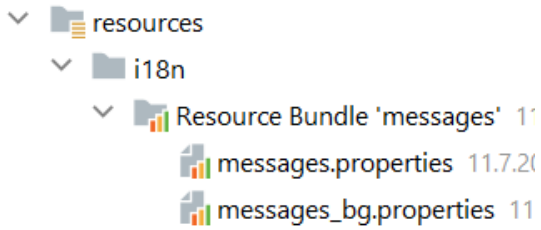
```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Cacheable {
  String value();
}

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class CacheableInvocationHandler implements InvocationHandler {
    private Map<String, Object> cachedValues = new ConcurrentHashMap<>();
    private final Object realObject;

    public CacheableInvocationHandler(Object realObject) {
        this.realObject = realObject;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        Cacheable cacheableAnnotation = realObject.getClass().
                getMethod(method.getName(), method.getParameterTypes()).
                getAnnotation(Cacheable.class); //дали има анотация Cacheable

        if (cacheableAnnotation != null) { //ако има Cacheable анотация
            //
            String cacheId = cacheableAnnotation.value();
            if (cachedValues.containsKey(cacheId)) {
                return cachedValues.get(cacheId);  //връщаме от кешираните стойности
            } else { //слагаме в кеша тук
                var value = method.invoke(realObject, args);
                cachedValues.put(cacheId, value);
                return value; //връщаме бавно първия път
            }
        } else {
            return method.invoke(realObject, args);  //връщаме резултата, който се връща от
самия метод
        }
    }
}
```

30. I18N – Change International language

```
import org.springframework.web.servlet.LocaleResolver;
```

```java
@Configuration
public class I18NConfig {
  @Bean
  public LocaleResolver localeResolver() {
    CookieLocaleResolver clr = new CookieLocaleResolver();
    clr.setCookieName("lang");
    return clr;
  }

  @Bean
  public LocaleChangeInterceptor localeChangeInterceptor() {
    LocaleChangeInterceptor lci = new LocaleChangeInterceptor();
    lci.setParamName("lang");
    return lci;
  }

  @Bean
  public MessageSource messageSource() {
    ResourceBundleMessageSource resourceBundleMessageSource = new ResourceBundleMessageSource();
    resourceBundleMessageSource.setBasename("i18n/messages");
    resourceBundleMessageSource.setDefaultEncoding("UTF-8");
    return resourceBundleMessageSource;
  }
}
```

```java
@Configuration
public class WebConfig implements WebMvcConfigurer {

  private LocaleChangeInterceptor localeChangeInterceptor;

  public WebConfig(LocaleChangeInterceptor localeChangeInterceptor) {
    this.localeChangeInterceptor = localeChangeInterceptor;
  }

  @Override
  public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(localeChangeInterceptor);
  }
}
```



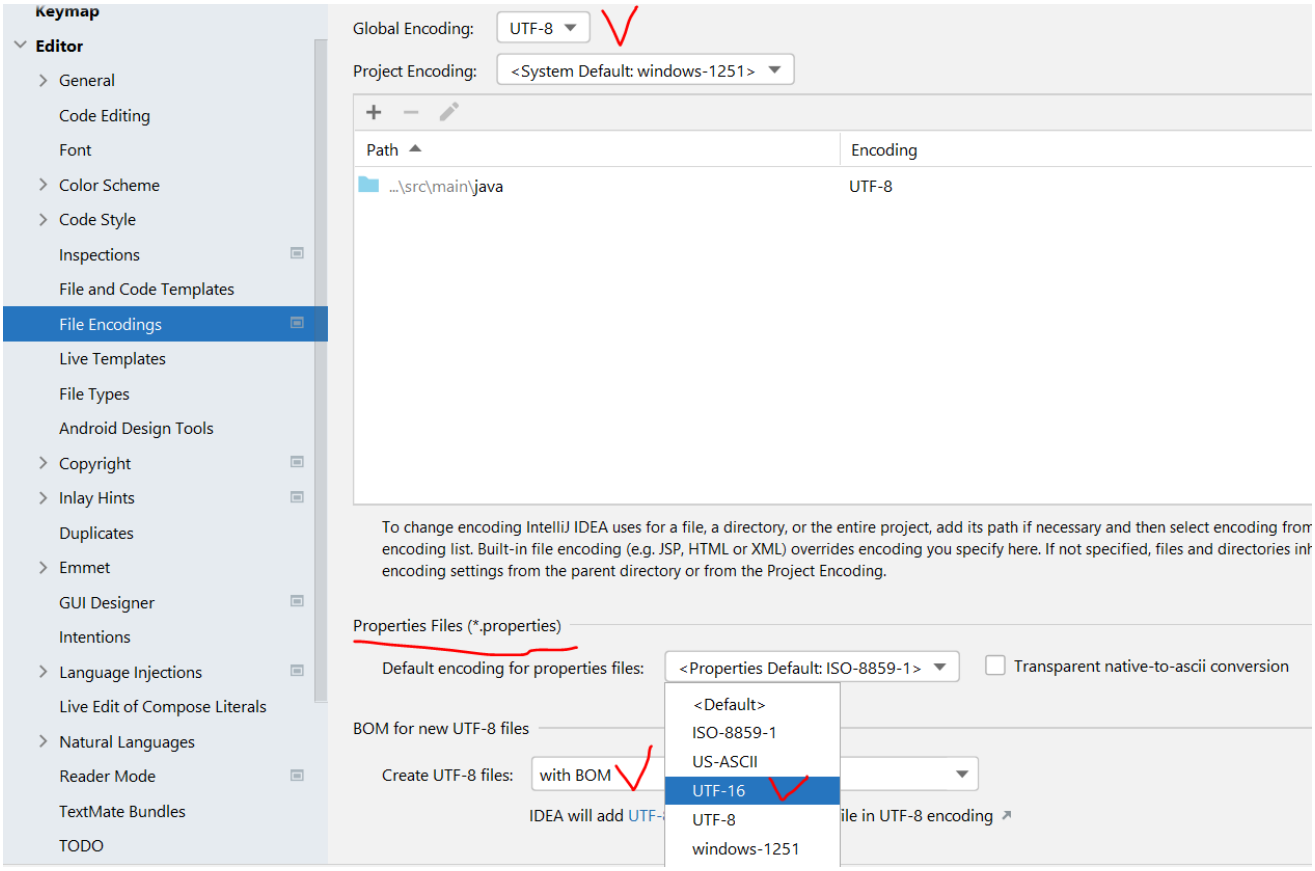

```properties
navbar_language = Language
navbar_login = Login
navbar_register = Register
navbar_all_brands= All brands
navbar_all_offers= All offers
navbar_search= Search
```

```
navbar_language = Ð²Ð·Ð¸Ðº
navbar_login = Ð□Ñ□Ð¾Ð´
navbar_register = Ð ÐµÐ³Ð¸Ñ□Ñ□Ð°Ñ□Ð¸Ñ□
navbar_all_brands= Ð□Ñ□Ð¸Ñ□ÐºÐ¸ Ð¾Ð°Ñ□ÐºÐ¸
navbar_all_offers= Ð□Ñ□Ð¸Ñ□ÐºÐ¸ Ð¾Ð±Ñ□Ð²Ð¸
navbar_search= Ð¢Ñ□Ñ□Ñ□ÐµÐ½Ðµ
```

```
<li class="nav-item">
    <form th:method="get">
        <label class="text-white" th:text="#{navbar_language}" for="lang"></label>
        <select
            onchange="this.form.submit()"
            id="lang"
            name="lang">
           <option value="en_US" th:selected="${#locale.toString()} == 'en_US'">English</option>
           <option value="bg_BG" th:selected="${#locale.toString()} == 'bg_BG'">Български</option>
        </select>
    </form>
</li>
```

```
<li class="nav-item">
    <a class="nav-link" th:text="#{navbar_search}" href="/offers/search">Search</a>
</li>
<li class="nav-item">
    <a class="nav-link" th:text="#{navbar_all_offers}" href="/offers/all">All Offers</a>
</li>
```

Настройка IntelliJ:



31. Cloudinary

https://cloudinary.com/

Виж пълното демо от лекцията Workshop 1.
За проекта – да ги качим ръчно в Cloudinary.

Можем да пазим файловете в Cloudinary по определен начин. Единият подход и с името на файла.
https://support.cloudinary.com/hc/en-us/articles/207548479-How-to-choose-between-different-resource-naming-methods

Cloudinary supports several methods for naming your uploaded resources.

Via API:

- Fully random 20 characters long public ID, which keeps your image URLs obscure for privacy reasons (That is the default behavior)
- The original filename followed by a unique 6 random characters suffix (activated by specifying `use_filename:true` & `unique_filename:true`)
- The original filename as-is, which will override any existing image that shares the same name (activated by specifying `use_filename:true` & `unique_filename:false`)

- Setting a specific public ID, this will override any existing image that shares the same name (`public_id:"my_resource"`)

Другият подход е с
Manage Structured Metadata: Structured metadata enables you to predefine a number of typed fields, with or without validation rules, as optional or required fields for every asset in your account. For example, if your application supports user-generated content, you can define structured metadata fields that will be captured and passed as part of your end-user uploads to keep track of things like the user who uploaded the asset, product categories, countries, or other important data, and can ensure that these fields are used consistently for all assets in your account.

Реално, ние като си ги пазим в база данни, то от базата данни ще вземаме линковете, и няма проблеми!!!

application.properties

```
#Cloudinary
cloudinary.cloud-name=dislhsmj5
cloudinary.api-key=532389729736197
cloudinary.api-secret=${CLOUDINARY_SECRET}
#Multipart file
spring.servlet.multipart.max-file-size=5MB
spring.servlet.multipart.max-request-size=5MB
spring.servlet.multipart.enabled=true
```

application.yml

```
spring:
  h2:
    console:
      enabled: true
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB
  mvc:
    hiddenmethod:
      filter:
        enabled: true

  jpa:
#    defer-datasource-initialization: true
    database-platform: org.hibernate.dialect.H2Dialect
    hibernate:
      ddl-auto: create

#Cloudinary properties
cloudinary:
  cloud_name: "dtfd8gw16"
```

```
  api_key: "218356847735493"
  api_secret: ${CLOUDINARY_SECRET}"
```

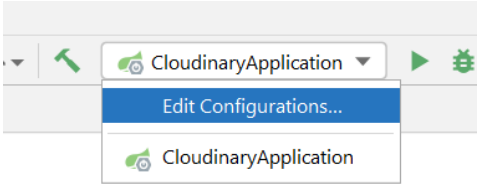
pom.xml

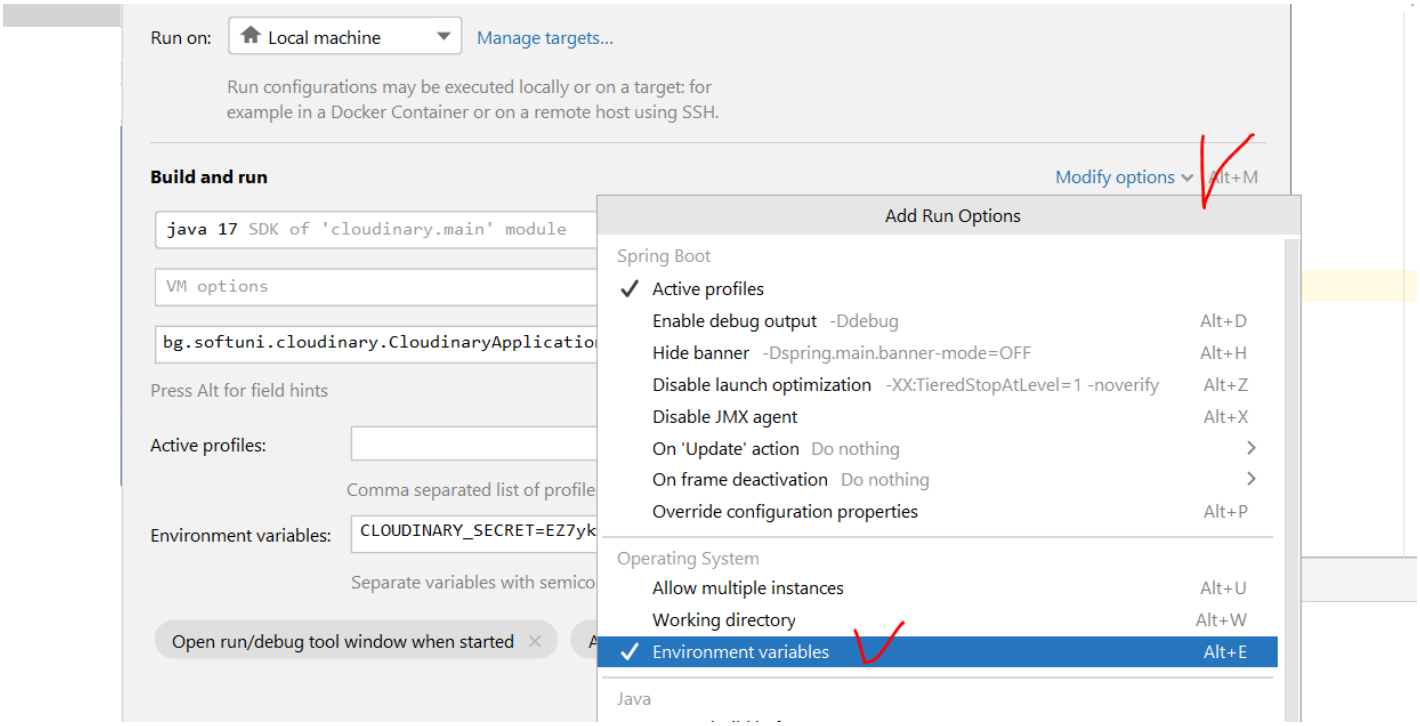
```
<dependency>
    <groupId>com.cloudinary</groupId>
    <artifactId>cloudinary</artifactId>
    <version>1.0.2</version>
</dependency>
```

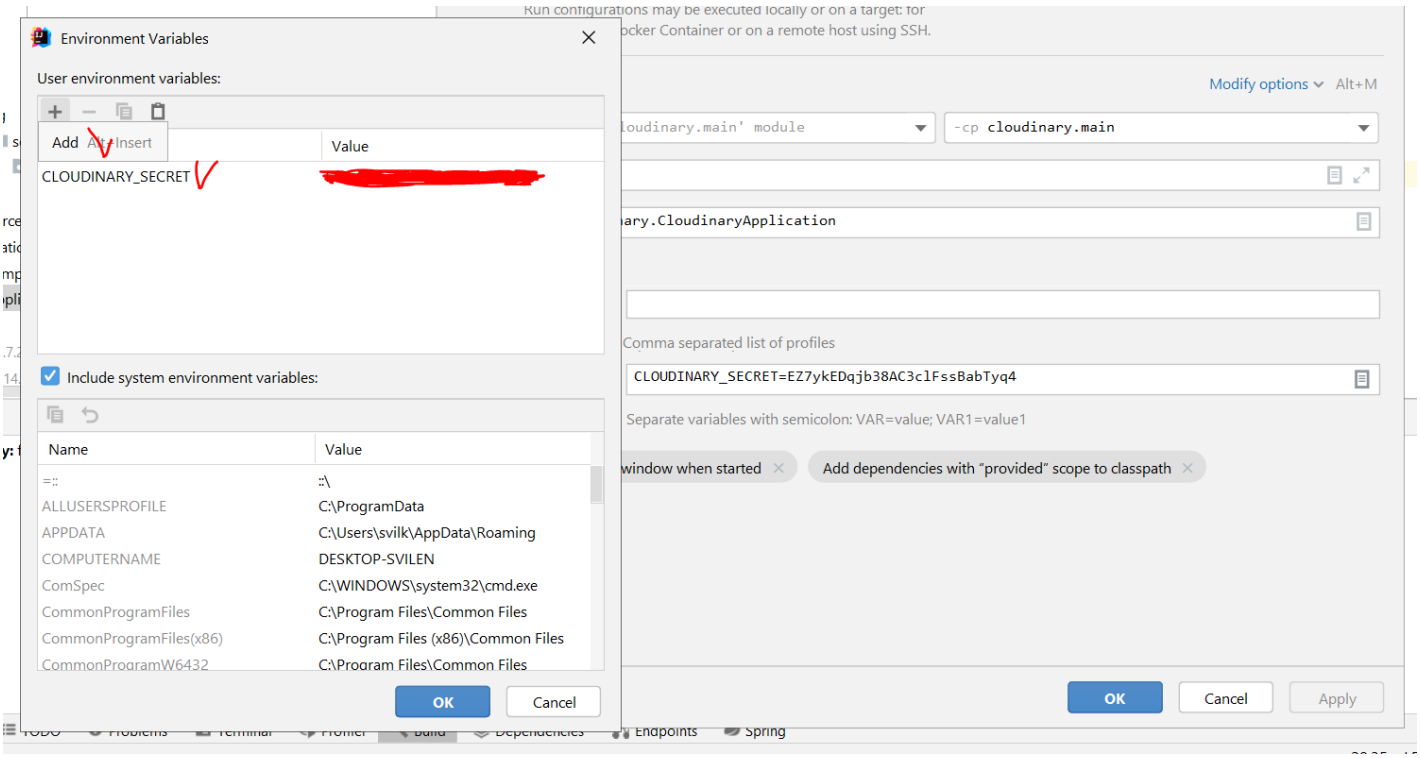
build.gradle

```
implementation 'com.cloudinary:cloudinary:1.0.14'
```

Добавяне на environment variable – ръчно в IntelliJ







Добавяме и @ConfigurationProperties

```java
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Configuration
@ConfigurationProperties(prefix = "cloudinary") //running with annotating processor
public class CloudinaryConfig {
    private String cloudName;
    private String apiKey;
    private String apiSecret;

    /**
     * Sets the Cloud name associated with the cloudinary account.
     * @param cloudName
     * @return this
     */
    public String getCloudName() {
        return cloudName;
    }

    public CloudinaryConfig setCloudName(String cloudName) {
        this.cloudName = cloudName;
        return this;
    }

    public String getApiKey() {
        return apiKey;
    }
```

```
    public CloudinaryConfig setApiKey(String apiKey) {
        this.apiKey = apiKey;
        return this;
    }

    public String getApiSecret() {
        return apiSecret;
    }

    public CloudinaryConfig setApiSecret(String apiSecret) {
        this.apiSecret = apiSecret;
        return this;
    }
}
```

```
@Configuration
public class AppConfig {
    private final CloudinaryConfig config;

    public AppConfig(CloudinaryConfig config) {
        this.config = config;
    }

    @Bean
    public Cloudinary cloudinary() {
        return new Cloudinary(
                Map.of("cloud_name", config.getCloudName(),
                        "api_key", config.getApiKey(),
                        "api_secret", config.getApiSecret())
        );
    }
}
```

```
import com.cloudinary.Cloudinary;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.File;
import java.io.IOException;
import java.util.Map;

@Service
public class CloudinaryService {
    private final Cloudinary cloudinary;
    private static final String TEMP_file = "temp-file";
    private static final String URL = "url";
    private static final String PUBLIC_ID = "public_id";

    public CloudinaryService(Cloudinary cloudinary) {
        this.cloudinary = cloudinary;
    }

    public CloudinaryImage upload(MultipartFile multipartFile) throws IOException {
        File tempFile = File.createTempFile(TEMP_file, multipartFile.getOriginalFilename());
        multipartFile.transferTo(tempFile);

        try {
```

```
            @SuppressWarnings("unchecked")
            Map<String, String> uploadResult = cloudinary
                    .uploader()
                    .upload(tempFile, Map.of());

            String url = uploadResult.getOrDefault(URL,
"https://thumbs.dreamstime.com/b/illustration-internet-connection-problem-concept-error-page-
not-found-isolated-white-background-funny-blue-dinosaur-230224212.jpg");
            String publicId = uploadResult.getOrDefault(PUBLIC_ID, "");

            var result = new CloudinaryImage()
                    .setPublicId(publicId)
                    .setUrl(url);

            return result;
        } finally {
            tempFile.delete();
        }

    }

    public boolean delete(String publicId) {
        try {
            this.cloudinary.uploader().destroy(publicId, Map.of());
        } catch (IOException e) {
            return false;
        }

        return true;
    }
}
```

Има и вариант само с 1 клас AppCloudConfig **– от курсовата работа на Дойча:**

```
import com.cloudinary.Cloudinary;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashMap;
import java.util.Map;

@Configuration
public class AppCloudConfig {
    @Value("${cloudinary.cloud-name}")
    private String cloudName;
    @Value("${cloudinary.api-key}")
    private String apiKey;
    @Value("${cloudinary.api-secret}")
    private String apiSecret;

    @Bean
    public Cloudinary createCloudinaryConfig() {
        Map<String, Object> config = new HashMap<>();
        config.put("cloud_name", cloudName);
```

```
        config.put("api_key", apiKey);
        config.put("api_secret", apiSecret);
        return new Cloudinary(config);
    }
}

@Service
public class CloudinaryServiceImpl implements CloudinaryService {
    private static final String TEMP_FILE = "temp-file";
    private static final String URL = "url";

    private final Cloudinary cloudinary;

    public CloudinaryServiceImpl(Cloudinary cloudinary) {
        this.cloudinary = cloudinary;
    }

    @Override
    public String uploadImage(MultipartFile multipartFile) throws IOException {
        File file = File.createTempFile(TEMP_FILE, multipartFile.getOriginalFilename());
        multipartFile.transferTo(file);

        Map uploadResult = cloudinary.uploader().upload(file, Collections.emptyMap());
        return uploadResult.get(URL).toString();
    }
}
```

annotationProcessor "org.springframework.boot:spring-boot-configuration-processor"
има и за maven проекти същото dependency

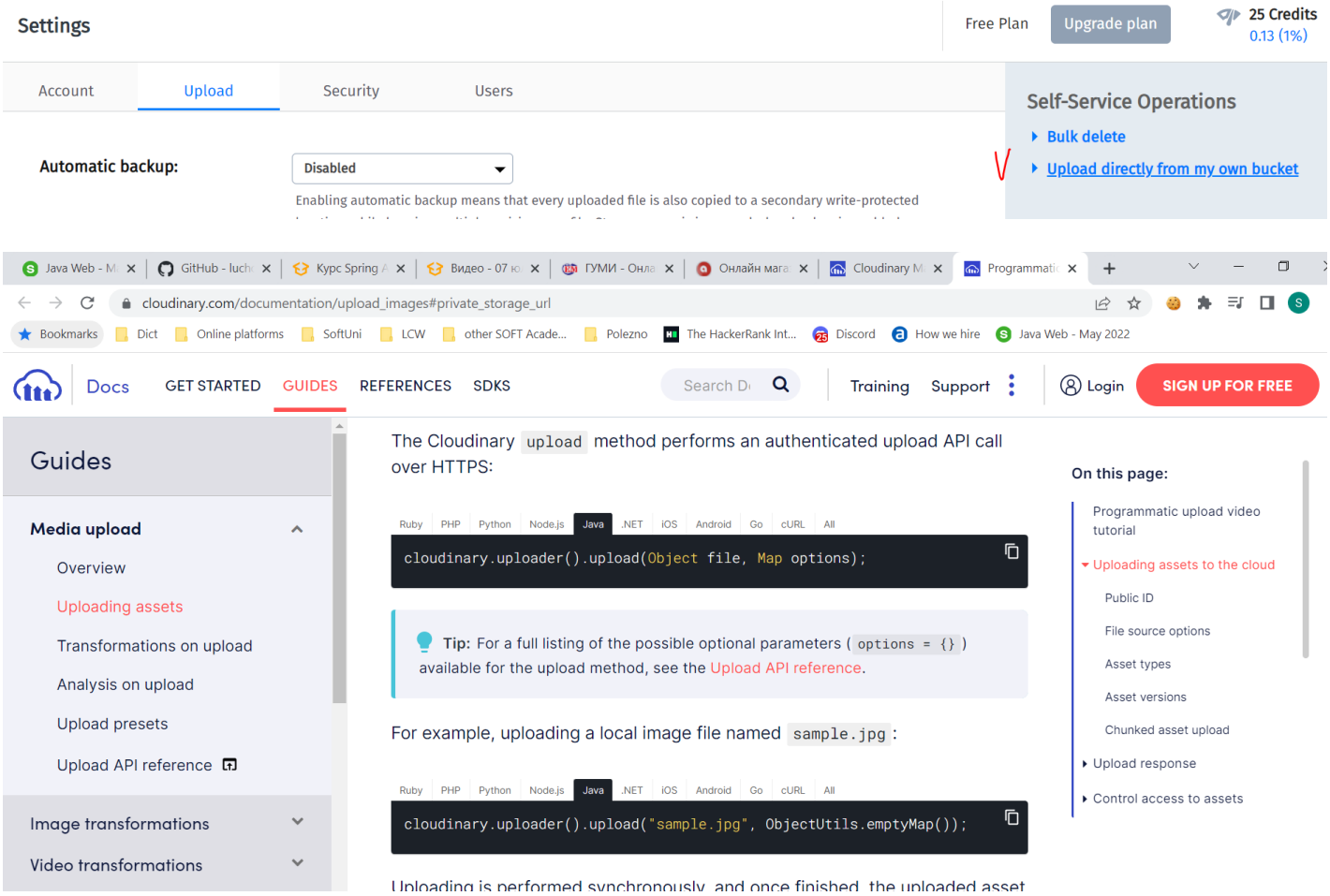
```
cloudinary:
  cloud_name: "dtfd8gw16"
  api
  api
```

Cannot resolve configuration property 'cloudinary.cloud_name'

Define configuration key 'cloudinary.cloud_name' Alt+Shift+Enter More actions... Alt+Enter

No documentation found.

Commands in JAVA how to make the upload:

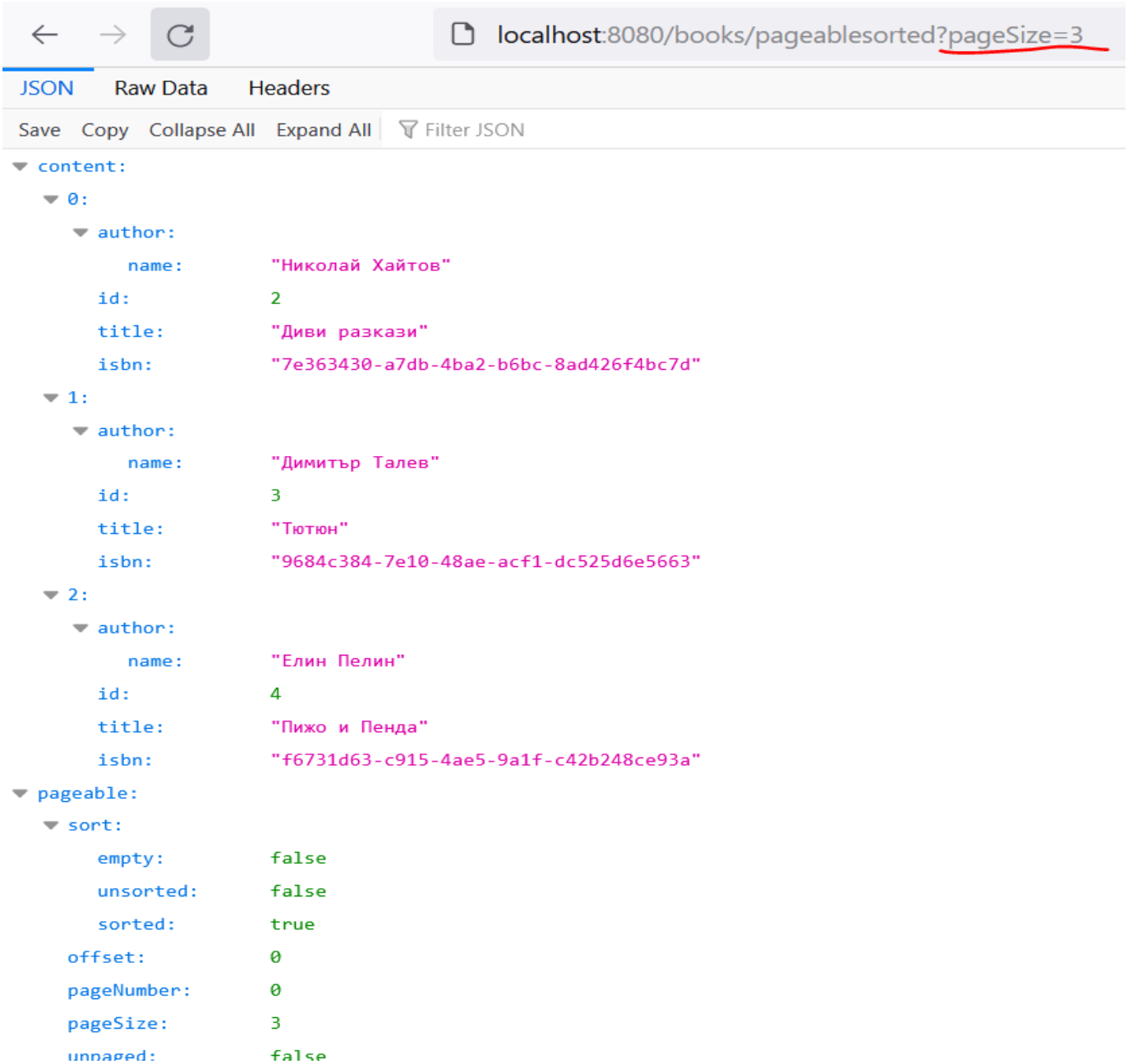


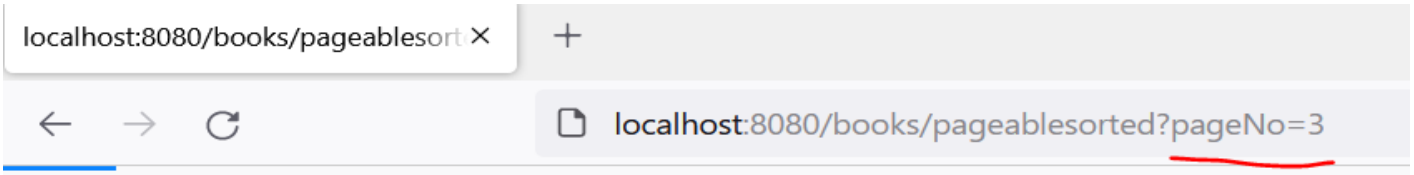

```
@SuppressWarnings({ "rawtypes", "unchecked" })
public class Cloudinary {
```

32. Pageabale and Sorted

```
@Service
public class BookServiceImpl {
    private BookRepository bookRepository;
    private ModelMapper modelMapper;
    private AuthorRepository authorRepository;

    public BookServiceImpl(BookRepository bookRepository, ModelMapper modelMapper,
AuthorRepository authorRepository) {
        this.bookRepository = bookRepository;
        this.modelMapper = modelMapper;
        this.authorRepository = authorRepository;
    }


    private BookDTO asBook(BookEntity book) {
```

```
        BookDTO bookDTO = modelMapper.map(book, BookDTO.class);
        AuthorDTO authorDTO = modelMapper.map(book.getAuthor(), AuthorDTO.class);

        bookDTO.setAuthor(authorDTO);

        return bookDTO;
    }


    public Page<BookDTO> getBooksPerPage(Integer pageNo, Integer pageSize, String sortBy) {
        Pageable pageable = PageRequest.of(pageNo, pageSize, Sort.by(sortBy));

        return bookRepository
                .findAll(pageable)
                .map(e -> asBook(e));
    }
}



@RestController
@RequestMapping("/books")
public class BooksController {
    private BookServiceImpl bookService;

    public BooksController(BookServiceImpl bookService) {
        this.bookService = bookService;
    }


    @DeleteMapping("/{bookId}")
    public ResponseEntity<BookDTO> deleteBookById(@PathVariable("bookId") Long bookId) {
        bookService
                .deleteBook(bookId);

        return ResponseEntity.noContent().build();
    }


    @GetMapping("/pageablesorted")
    public ResponseEntity<Page<BookDTO>> getBooksPerPage(
            @RequestParam(name = "pageNo", defaultValue = "0") Integer pageNo,
            @RequestParam(name = "pageSize", defaultValue = "3") Integer pageSize,
            @RequestParam(name = "sortBy", defaultValue = "id") String sortBy) {

        return ResponseEntity.ok(
                bookService.getBooksPerPage(pageNo, pageSize, sortBy)
        );
    }
}
```

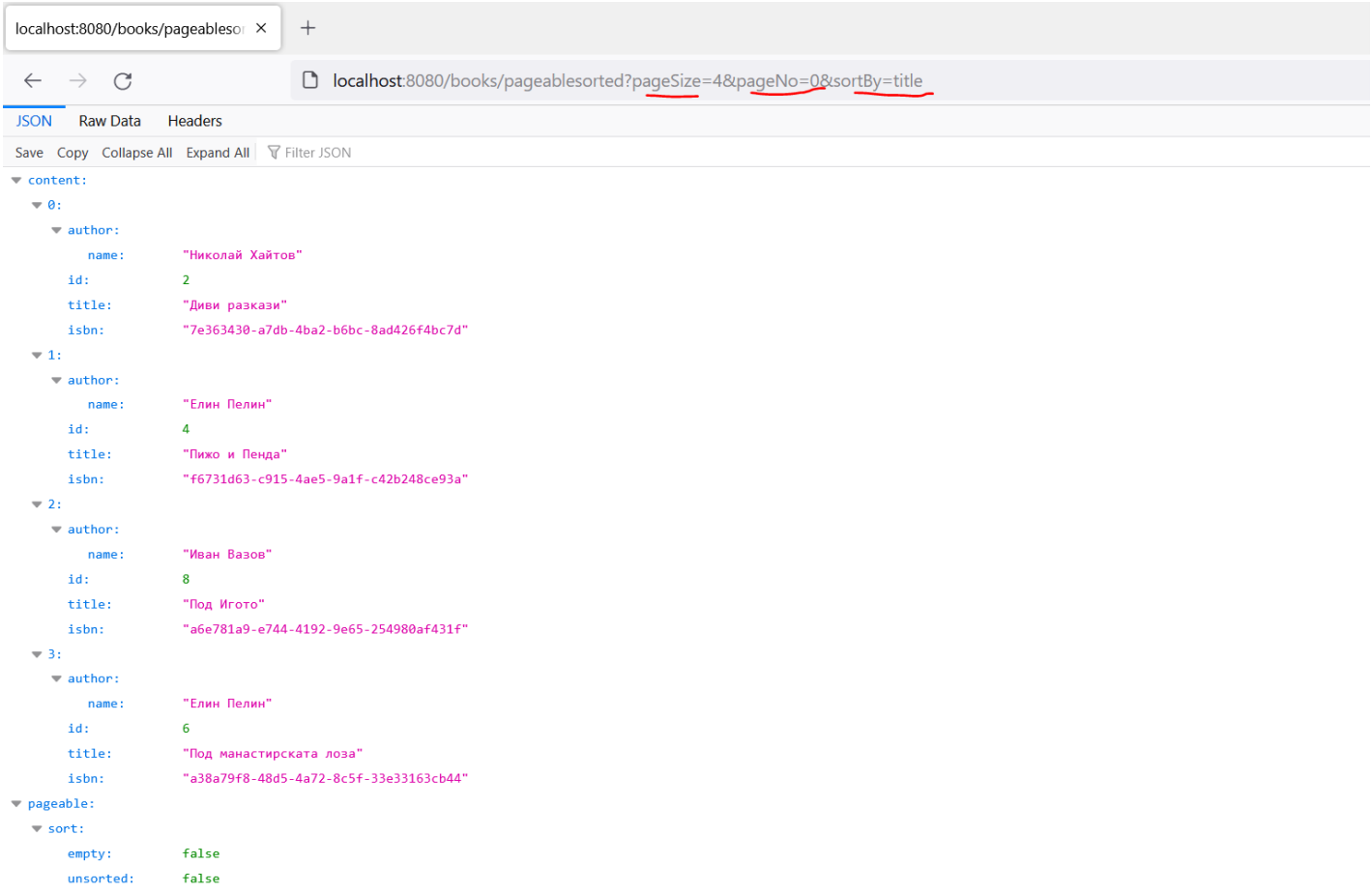

localhost:8080/books/pageablesorted?pageSize=3
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
content:
0:
author:
name: "Николай Хайтов"
id: 2
title: "Диви разкази"
isbn: "7e363430-a7db-4ba2-b6bc-8ad426f4bc7d"
1:
author:
name: "Димитър Талев"
id: 3
title: "Тютюн"
isbn: "9684c384-7e10-48ae-acf1-dc525d6e5663"
2:
author:
name: "Елин Пелин"
id: 4
title: "Пижо и Пенда"
isbn: "f6731d63-c915-4ae5-9a1f-c42b248ce93a"
pageable:
sort:
empty: false
unsorted: false
sorted: true
offset: 0
pageNumber: 0
pageSize: 3
unpaged: false



JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON


```
content:
  0:
    author:
      name:          "Йордан Йовков"
    id:              11
    title:           "Чифликът край границата"
    isbn:            "52e8ba56-baf1-4ead-a17c-33866571024d"
pageable:
  sort:
    empty:           false
    unsorted:        false
    sorted:          true
  offset:            9
  pageNumber:        3
  pageSize:          3
  unpaged:           false
  paged:             true
last:                true
totalPages:          4
totalElements:       10
size:                3
number:              3
sort:
  empty:             false
```



40. Other

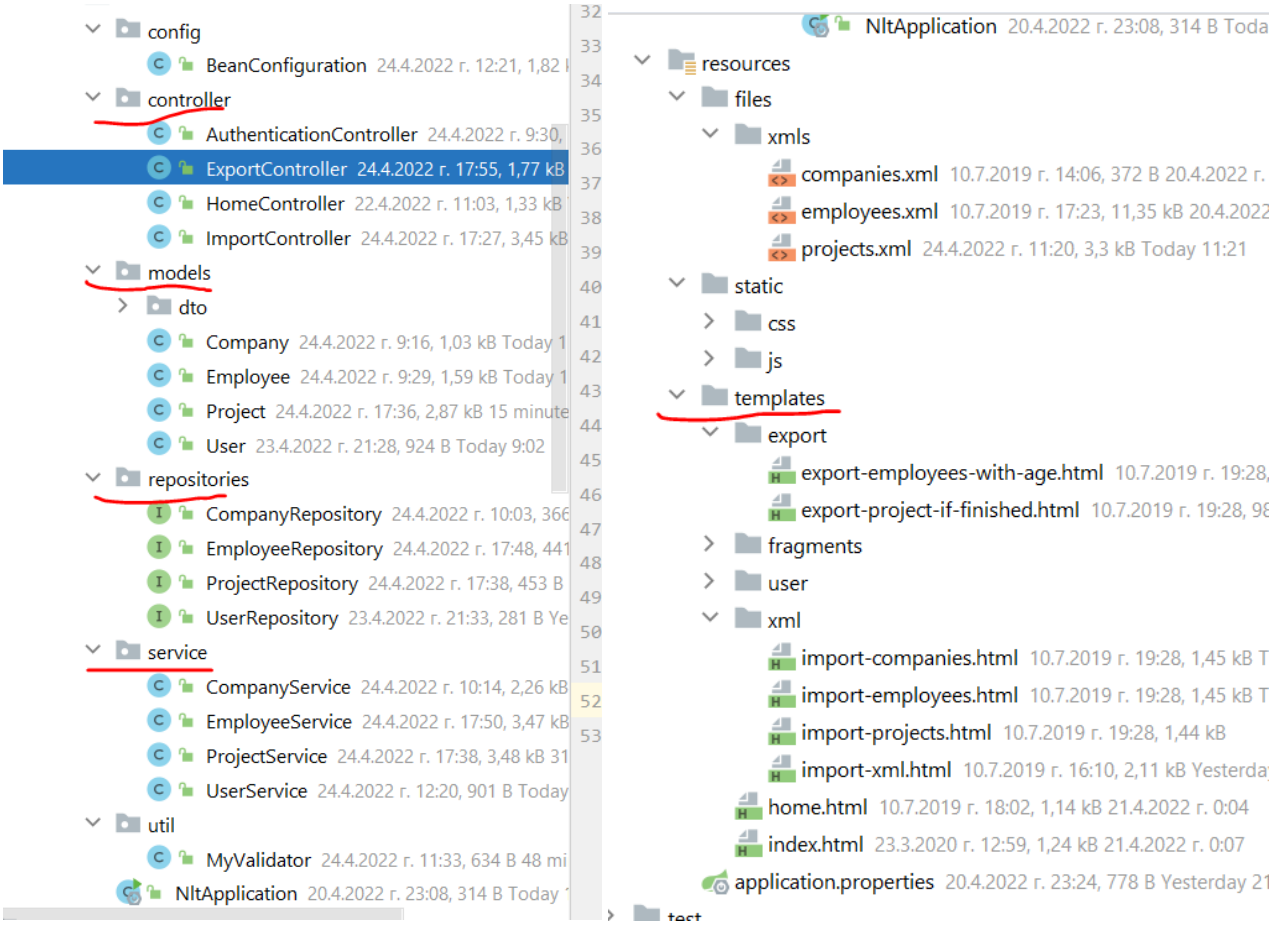
```
@PostMapping(value = "/users/register", consumes = MediaType.APPLICATION_FORM_URLENCODED_VALUE)
public String doRegister(@RequestBody MultiValueMap<String, String> paramMap){
    System.out.println(paramMap.get("user"));

    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";
    }
```

Logout с Rest

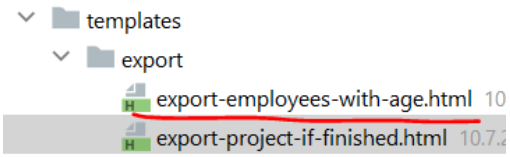
Трябва да си вземем и crsf токена към post заявката към users/logout url-а

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>
```

Regex in 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>
```

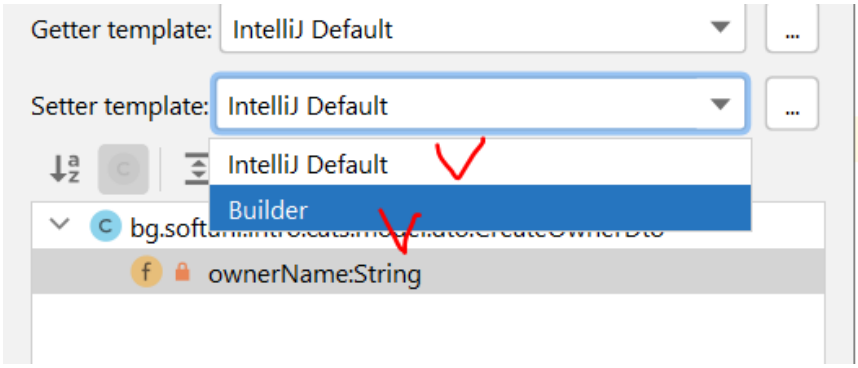
Uploading a file

```
@PostMapping("/") //покажи всички
public String storeContact(Contact contact, @RequestParam("picture")MultipartFile picture) {
    this.contacts.add(contact);

    return "redirect:/"; //върни ме на началната страница //редиректваме към този URL, не към
html 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>
```

Build pattern

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 с искания от нас Bean

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.

```
@Id
@GeneratedValue(generator = "uuid-string")
@GenericGenerator(name = "uuid-string",
        strategy = "org.hibernate.id.UUIDGenerator")
public String getId() { return id; }

public void setId(String id) { this.id = id; }
```

```
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;
```

```
Пре repository-то  използваме String когато имаме UUID
@Repository
public interface UserRepository extends JpaRepository<UserEntity, String> {
}
```

```
Произволен UUID номер
aBook.setAuthor(author).setTitle(bookTitle).setIsbn(UUID.randomUUID().toString());
```

org.slf4j.Logger

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;

import java.util.Optional;


@Service
public class UserService {
    private UserRepository userRepository;
    private CurrentUser currentUser;
    private Logger LOGGER = LoggerFactory.getLogger(UserService.class);


if (userOpt.isEmpty()) {
    LOGGER.info("User with name [{}] not found.", loginDto.getUsername()); //в къдравите скоби се изписва
        LOGGER.debug("User with name {} not found.", loginDto.getUsername()); //в къдравите скоби се изписва

    return false;
}
```

Partial security – password encryption

```
@Configuration
public class ApplicationConfig {

    @Bean
    public PasswordEncoder passwordEncoder(){
```

```
        return new Pbkdf2PasswordEncoder();
    }
}
```

```
public void registerAndLogin(UserRegisterDto userRegisterDto) {
    UserEntity newUser = new UserEntity()
            .setActive(true)
            .setEmail(userRegisterDto.getEmail())
            .setFirstName(userRegisterDto.getFirstName())
            .setLastName(userRegisterDto.getLastName())
            .setPassword(passwordEncoder.encode(userRegisterDto.getPassword()));

    newUser = userRepository.save(newUser);

    login(newUser);
}
```

```
var rawPassword = loginDto.getPassword();
var encodedPassword = userOpt.get().getPassword(); //което се е записало в базата данни

boolean success = passwordEncoder.matches(rawPassword, encodedPassword);
```

html new project generating in Java

Записваме файлът като .html
! + Таб
html:5 + Enter

POST-redirect-GET principle

Create an object,
then re-load **the same** URL page,
and return the created object.

```
@Controller
@RequestMapping("/users")
public class UserRegistrationController {
    private UserService userService;

    public UserRegistrationController(UserService userService) {
        this.userService = userService;
    }

    @ModelAttribute("userModel") //изпълнява се в рамките на текущия контролер само!!!
    public void initUserModel(Model model){
        model.addAttribute("userModel", new UserRegisterDto());
    }

    @GetMapping("/register")
    public String register() {
        // когато зареждаме за първи път страницата, то автомиатично ще влезе към модела атрибут
userModel == празен new UserRegisterDto()
        return "auth-register.html";
```

```
    }

    @PostMapping("/register")
    public String register(@Valid UserRegisterDto userRegisterDto,
                           BindingResult bindingResult,
                           RedirectAttributes redirectAttributes) {

        if (bindingResult.hasErrors()) {
            redirectAttributes.addFlashAttribute("userModel", userRegisterDto);
            redirectAttributes.addFlashAttribute("org.springframework.validation.BindingResult.
userModel", bindingResult);
            return "redirect:/users/register";
        }

        userService.registerAndLogin(userRegisterDto);
        return "redirect:/"; //редиректваме към този URL, не към html страница
    }
}
```

Java dynamic proxies

Благодарение на тях, целият Spring и анотациите работят на основата на тези proxies.

sessionStorage от DevTools на браузъра се различава от Cookie HTTP session – да.

localStorage and sessionStorage are perfect for persisting non-sensitive data needed within client scripts between pages (for example: preferences, scores in games). The data stored in localStorage and sessionStorage can easily be read or changed from within the client/browser so should not be relied upon for storage of sensitive or security-related data within applications.

As cookies are used for authentication purposes and persistence of user data, **all** cookies valid for a page are sent from the browser to the server for **every** request to the same domain - this includes the original page request, any subsequent Ajax requests, all images, stylesheets, scripts, and fonts. For this reason, cookies should not be used to store large amounts of information. The browser may also impose limits on the size of information that can be stored in cookies. Typically cookies are used to store identifying tokens for authentication, session, and advertising tracking. The tokens are typically not human readable information in and of themselves, but encrypted identifiers linked to your application or database.

localStorage, sessionStorage, and cookies are all subject to "same-origin" rules which means browsers should prevent access to the data except the domain that set the information to start with.

Client-side validation are localStorage и sessionStorage

1. **LocalStorage**

   Pros:

   1. Web storage can be viewed simplistically as an improvement on cookies, providing much greater storage capacity. If you look at the Mozilla source code we can see

that **5120KB** (**5MB** which equals **2.5 Million chars** on Chrome) is the default storage size for an entire domain. This gives you considerably more space to work with than a typical 4KB cookie.
2. The data is not sent back to the server for every HTTP request (HTML, images, JavaScript, CSS, etc) - reducing the amount of traffic between client and server.
3. The data stored in localStorage persists until explicitly deleted. Changes made are saved and available for all current and future visits to the site.

Cons:

4. It works on same-origin policy. So, data stored will only be available on the same origin.

1. **sessionStorage**

Pros:

1. It is similar to localStorage.
2. The data is not persistent i.e. data is only available per window (or tab in browsers like Chrome and Firefox). Data is only available during the page session. Changes made are saved and available for the current page, as well as future visits to the site on the same tab/window. Once the tab/window is closed, the data is deleted.

Cons:

3. The data is available only inside the window/tab in which it was set.
4. Like localStorage, it works on same-origin policy. So, data stored will only be available on the same origin.

Server-side validation is Cookies, including Cookies HTTP sessionStorage

3. **Cookies**

Pros:

1. Compared to others, there's nothing AFAIK.

Cons:

2. The 4K limit is for the entire cookie, including name, value, expiry date etc. To support most browsers, keep the name under 4000 bytes, and the overall cookie size under 4093 bytes.
3. The data is sent back to the server for every HTTP request (HTML, images, JavaScript, CSS, etc) - increasing the amount of traffic between client and server.

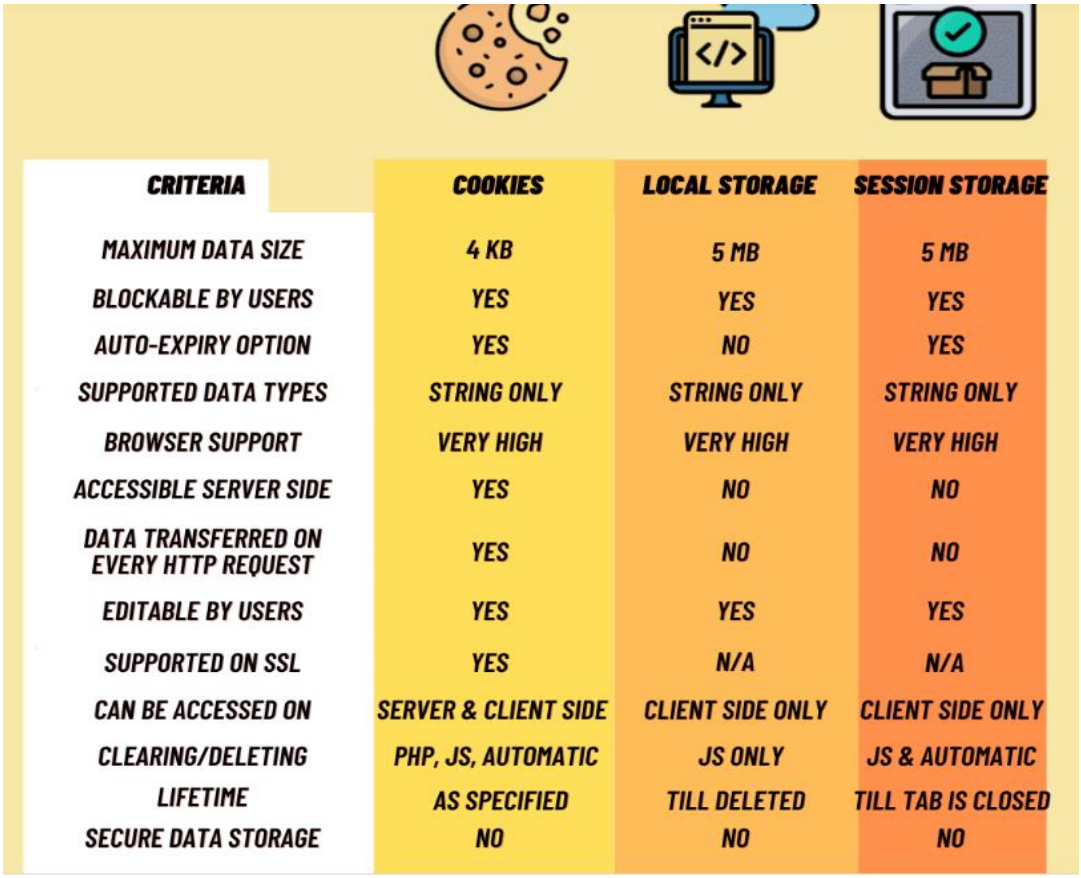
Typically, the following are allowed:

1. **300** cookies in total
2. **4096 bytes** per cookie
3. **20 cookies** per domain
4. **81920 bytes** per domain(Given 20 cookies of max size 4096 = 81920 bytes.)

LocalStorage	5MB/10MB storage It's not session based, need to be deleted via JS or manually Client side reading only Less older browsers support
SessionStorage	5MB storage It's session based and working per window or tab Client side reading only Less older browsers support
Cookie	4KB storage Expiry depends on the setting and working per window or tab Server and client side reading More older browsers support

Cookies vs. Local Storage vs. Session Storage

	Cookies	Local Storage	Session Storage
Capacity	4kb	10mb	5mb
Browsers	HTML4 / HTML 5	HTML 5	HTML 5
Accessible from	Any window	Any window	Same tab
Expires	Manually set	Never	On tab close
Storage Location	Browser and server	Browser only	Browser only
Sent with requests	Yes	No	No



CRITERIA	COOKIES	LOCAL STORAGE	SESSION STORAGE
MAXIMUM DATA SIZE	4 KB	5 MB	5 MB
BLOCKABLE BY USERS	YES	YES	YES
AUTO-EXPIRY OPTION	YES	NO	YES
SUPPORTED DATA TYPES	STRING ONLY	STRING ONLY	STRING ONLY
BROWSER SUPPORT	VERY HIGH	VERY HIGH	VERY HIGH
ACCESSIBLE SERVER SIDE	YES	NO	NO
DATA TRANSFERRED ON EVERY HTTP REQUEST	YES	NO	NO
EDITABLE BY USERS	YES	YES	YES
SUPPORTED ON SSL	YES	N/A	N/A
CAN BE ACCESSED ON	SERVER & CLIENT SIDE	CLIENT SIDE ONLY	CLIENT SIDE ONLY
CLEARING/DELETING	PHP, JS, AUTOMATIC	JS ONLY	JS & AUTOMATIC
LIFETIME	AS SPECIFIED	TILL DELETED	TILL TAB IS CLOSED
SECURE DATA STORAGE	NO	NO	NO

HTTP Session: A session is a global variable stored on the server. Each session is assigned a unique id which is used to retrieve stored values.

Authorization vs. authentication

Authentication – чрез механизъм се определя кой си
Authorization – какво моя user има право да прави в системата

други

Websocket-и работят с постоянни данни – подходящо за чатове!!!
Tomcat е на apache

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

Browser -> ….. -> controller (DTO) -> Model -> Thymeleaf -> browser
@Controller + response body = @RestController

```
th:field
th:value – в някои особени случаи се използва само
```

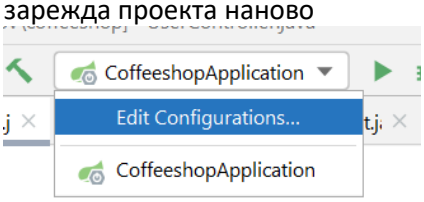
Валидацията при клиента работи още преди да сме дали submit бутона на формата и е само за удобство на клиента. Реално винаги трябва да имаме валидации при back-end server-а след като сме изпратили заявката.

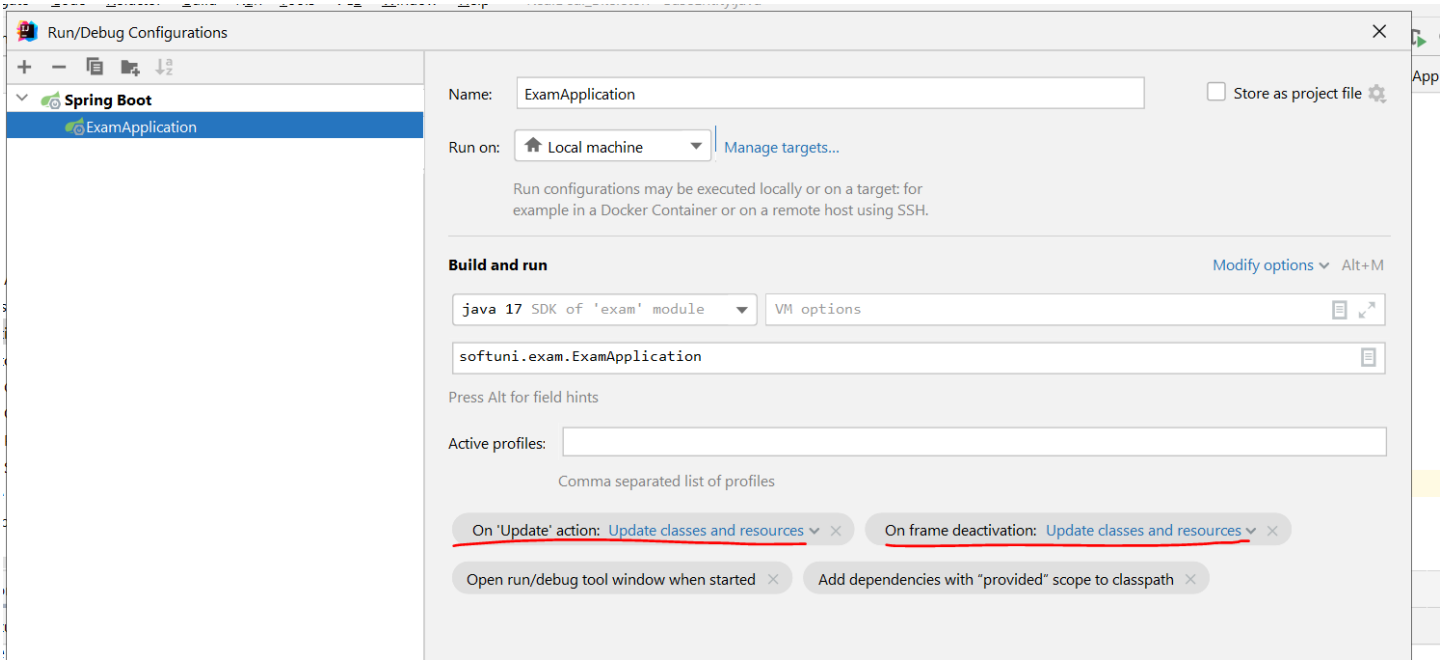
JS е нещото, което може да вмъкне json данни (дошли от http rest протокола) в html кода, без значение дали използваме thymeleaf или не.

Thymeleaf бил server-side технология, един вид server-side rendering

Live templating

On Update action / On frame deactivation.
За да не чакаме по 7-10 секунди след всяка промяна в кода (някакви дреболии по fron-енд кода особено) да зарежда проекта наново





Настройки Thymeleaf

When copying manually templates in resoiurces folder:
Option A) – restart IntelliJ

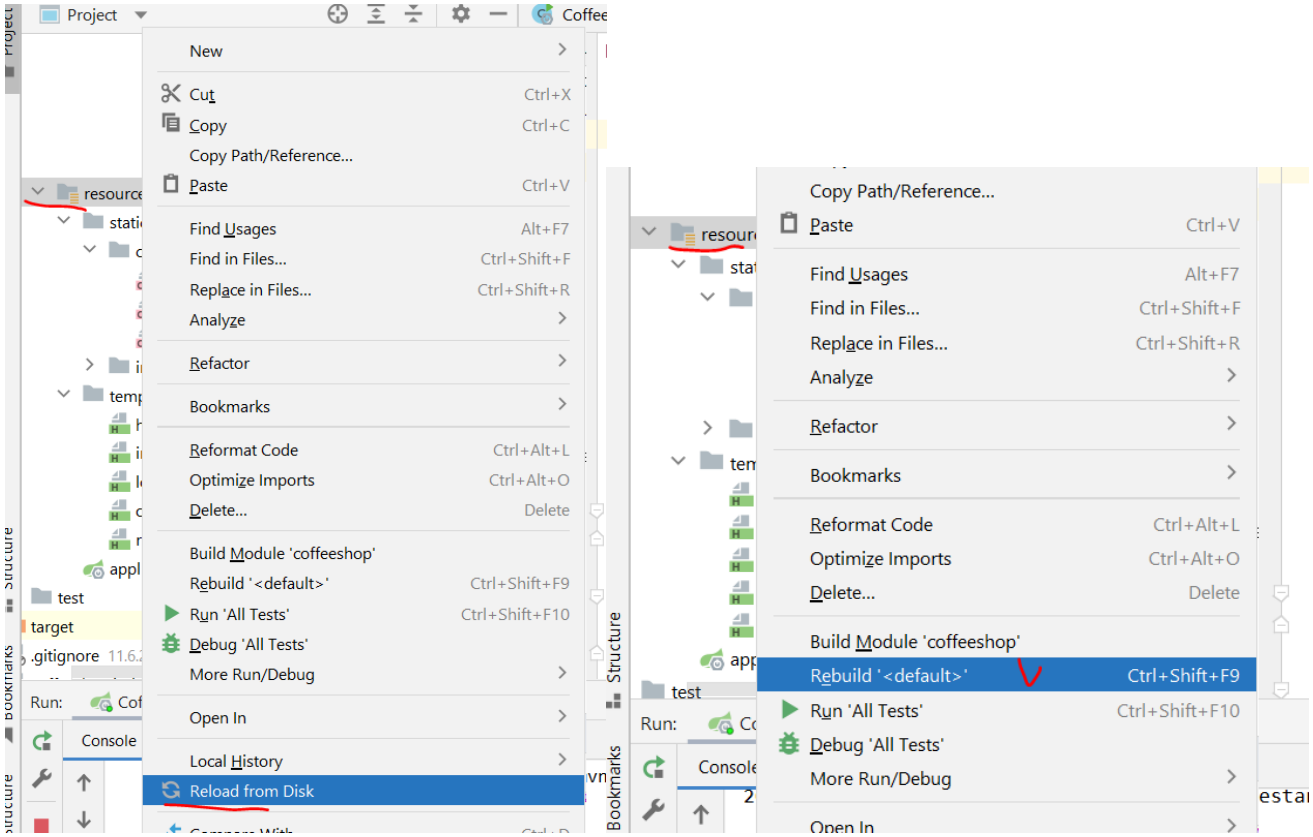
Option B) – run these 2 commands

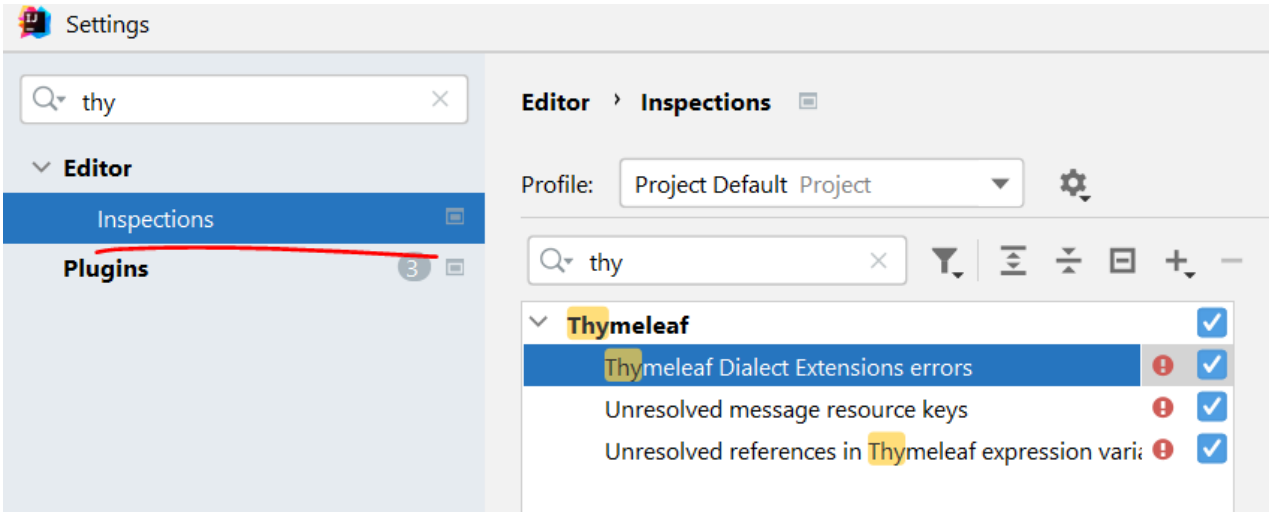
виж версията дали е такава, при мен това беше проблема
в pom.xml при Maven

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.3</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

в build.gradle при Gradle

```
plugins {
    id 'org.springframework.boot' version '2.6.7'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
}
```





JWT (JSON Web Token)

What is JSON Web Token?

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

When should you use JSON Web Tokens?

https://jwt.io/introduction

Client-Side vs. Server-Side Rendering

https://www.openmymind.net/2012/5/30/Client-Side-vs-Server-Side-Rendering/

How It Works

With **client-side rendering**, your initial request loads the page layout, CSS and JavaScript. It's all common except that some or all of the content isn't included. Instead, the JavaScript makes another request, gets a response (likely in JSON), and generates the appropriate HTML (likely using a templating library).

With server-side rendering, your initial request loads the page, layout, CSS, JavaScript and content.

For subsequent updates to the page, the client-side rendering approach repeats the steps it used to get the initial content. Namely, JavaScript is used to get some JSON data and templating is used to create the HTML.

Initial Load

Comparing the initial flow of the two approaches, it should be obvious that **client-side rendering is going to be slower**. **It requires more JavaScript to be downloaded, which is more JavaScript to parse**. **It requires a 2nd HTTP request to load the content, and then requires more JavaScript to generate the template**. Even if the initial JavaScript gets cached, it still needs to get parsed, and the 2nd request isn't going to happen until the document is loaded.

При **client-side rendering** веднъж има заявка да се get-не базовата html страница, и втори или повече пъти има още rest заявки, които JS script-а генерира с някаква логика.

Timing атаки

Обикновено ако хакер се пробва да се логва много пъти, той следи за колко време се връща отговор от сървъра. Ако е за 10ms значи няма такъв user. Ако е за 100ms, значи съществува такъв user.
Spring нарочно забавя random отговора от server-а, и хакера да не може да разбере.