

Spring Web

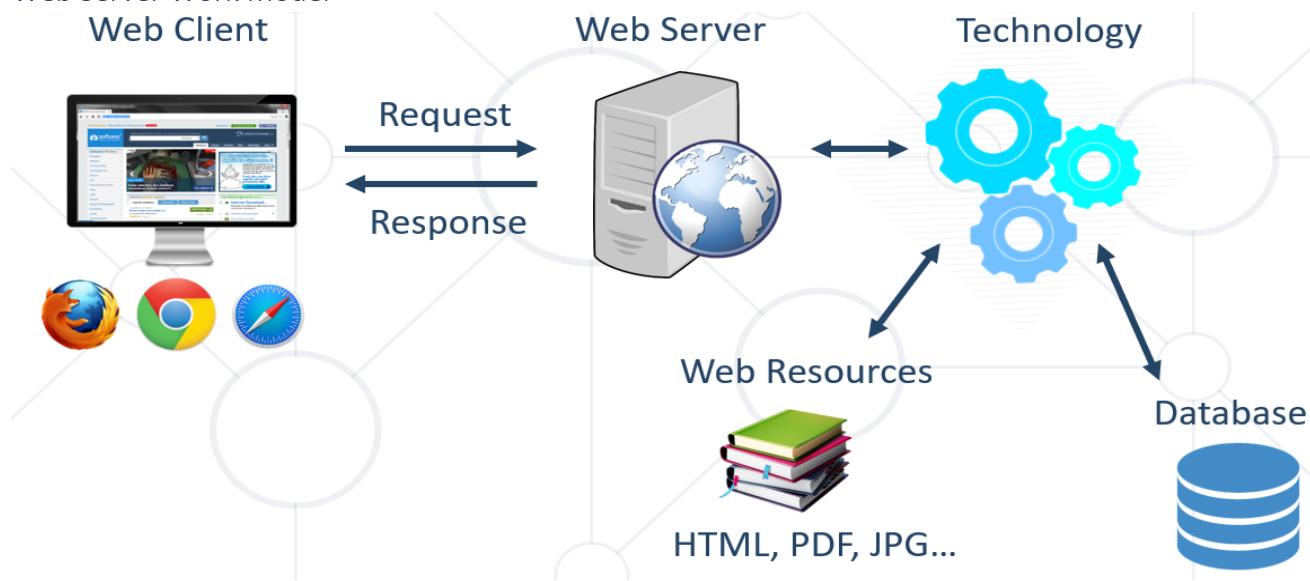
1. Internet Explained

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

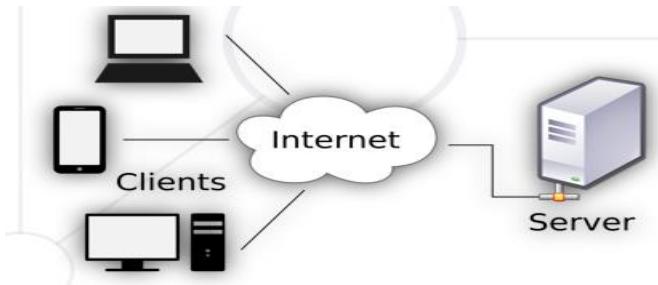


Important Definitions

- 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, SMTP, SSH

Какво е протокол? - механизъм/интерфейс (на дадения сървър или на дадената база данни), по който да структурираме **мрежовите пакети** и как да ги изпращаме и получаваме към съответното приложение.

Протоколи в Java най-общо казано се имплементират посредством Socket API (на ниско ниво), което ни позволява да реализираме протоколи използвайки TCP или UDP стек.

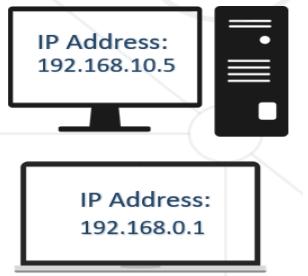
1.3. Packets (late 1960s)

- 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
 - Subnetworks

168.14.120

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

IP address classes

What Is CIDR (Classless Inter-Domain Routing)

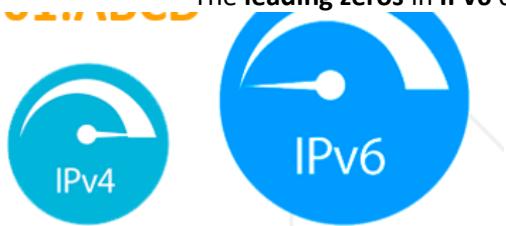
- Classless Inter-Domain Routing, is an IP addressing scheme that **improves the allocation of IP addresses**.
- **It replaces the old system based on classes A, B, and C.**
- This scheme also helped greatly **extend the life of IPv4** as well as slow the growth of routing tables

IPv4 Private Address Space and Filtering

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



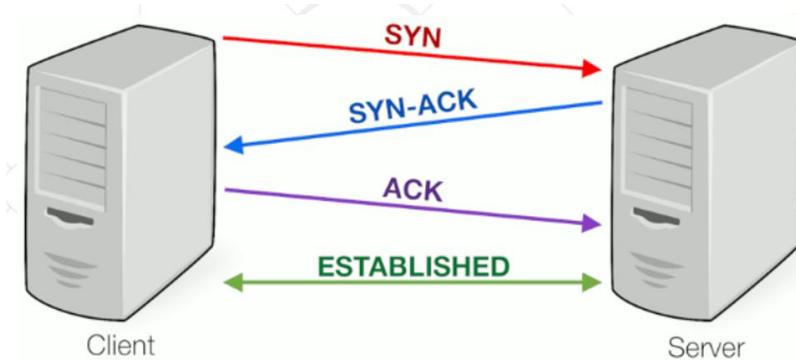
1.5. Reliability and TCP (Transmission Control Protocol)

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)

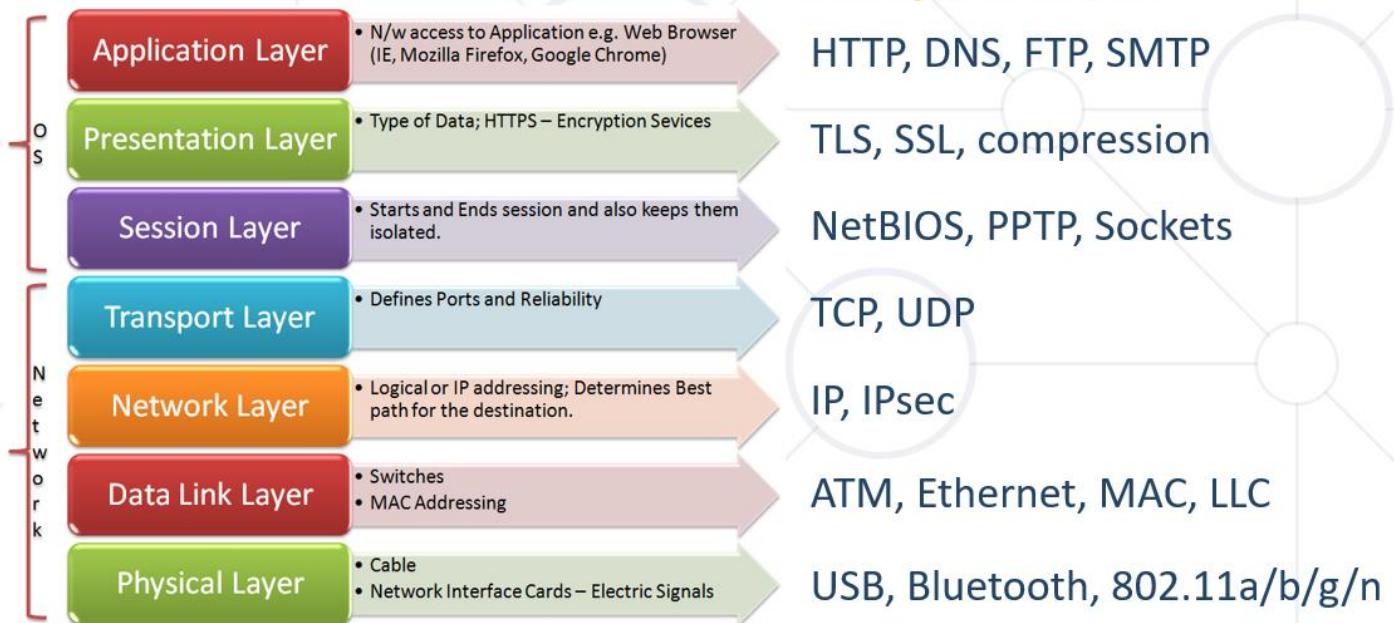
What is the OSI Model?

- **OSI** model stands for **Open System Interconnect**
- 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

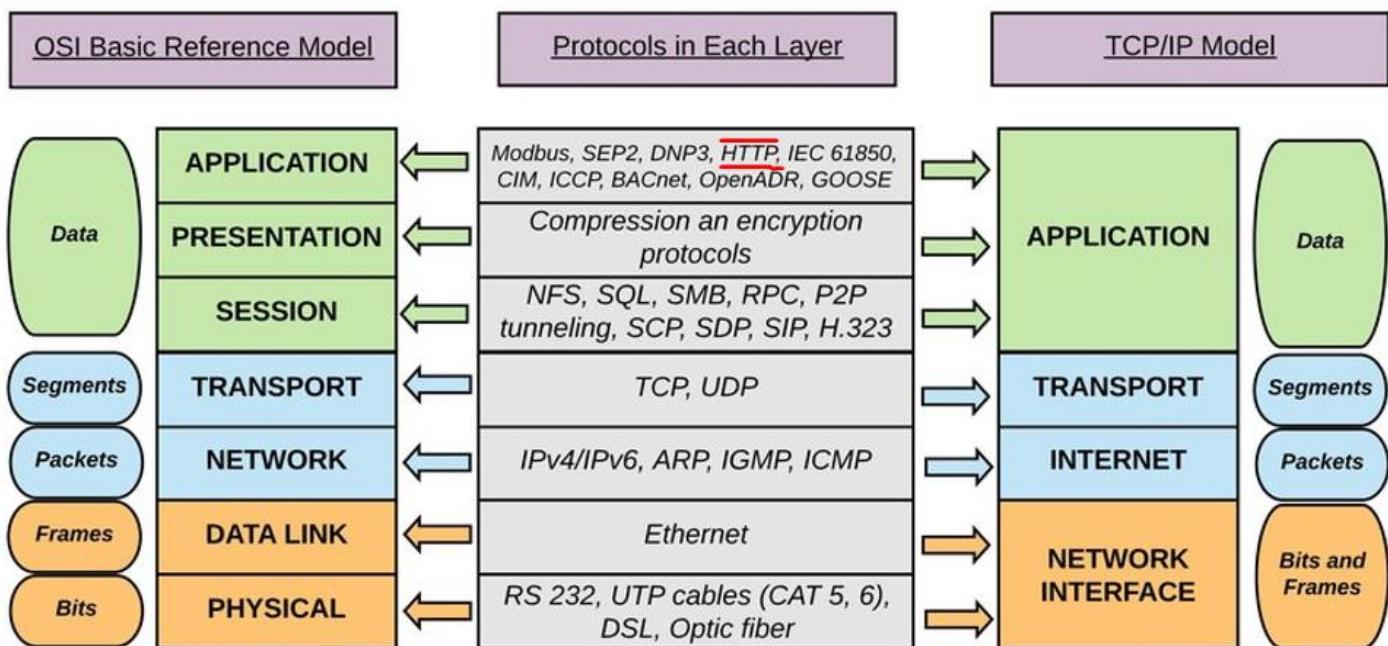
OSI Layers

- OSI Model consists of 7 layers:

I 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 (**DNS** system)
 - File Transfer Protocol (**FTP**)
 - HyperText Transfer Protocol (**HTTP**)
 - Simple Mail Transfer Protocol (**SMTP**)



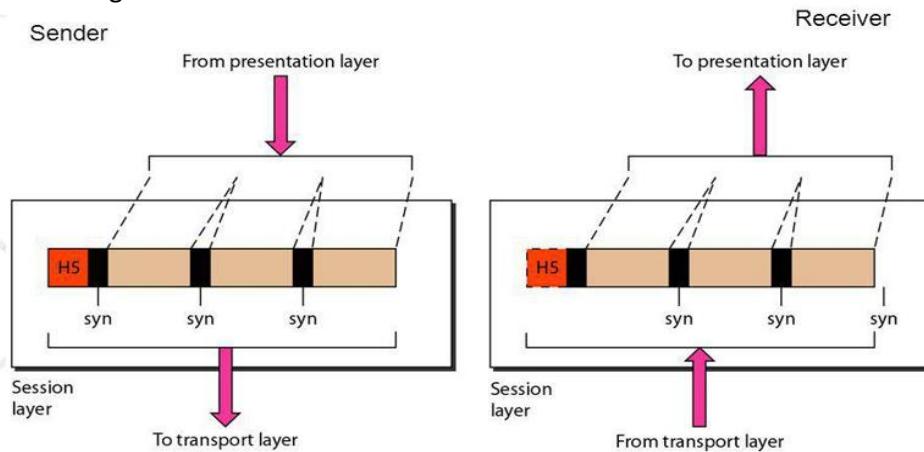
Presentation Layer – level 6

- 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



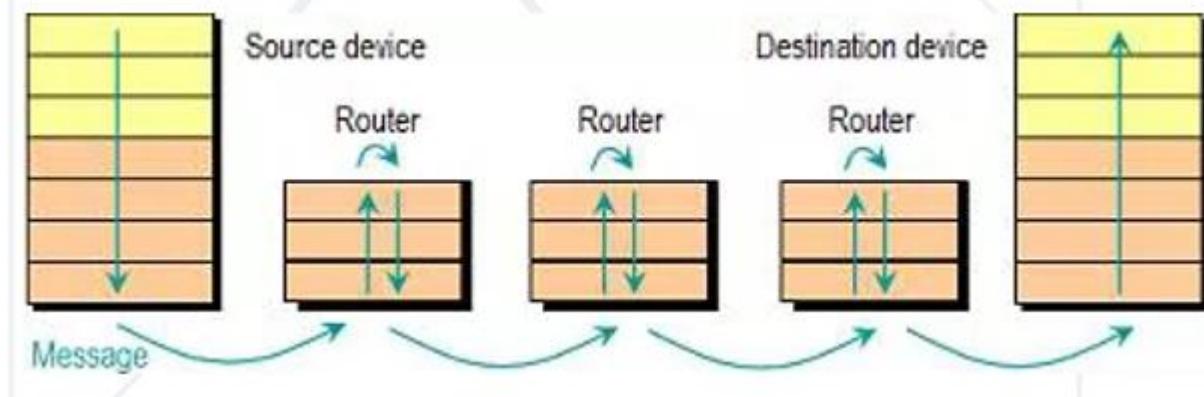
Transport Layer – level 4

- 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.

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

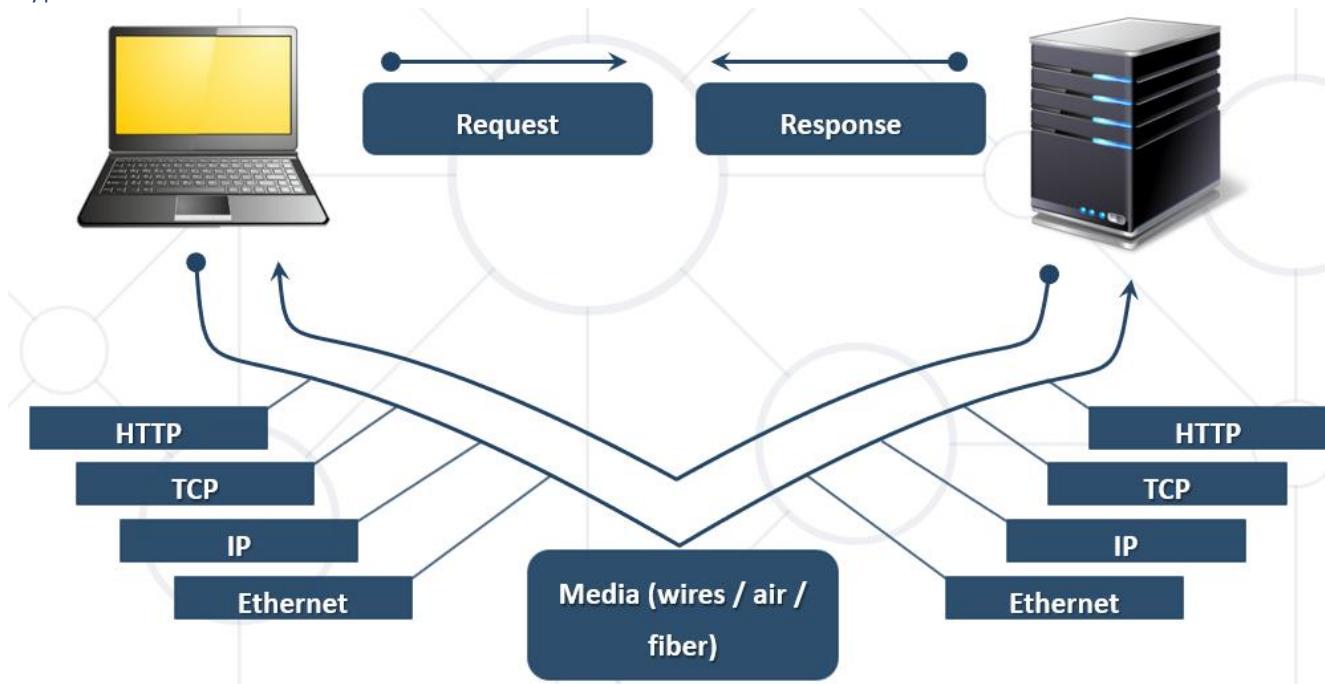
<https://blog.restcase.com/4-maturity-levels-of-rest-api-design/>

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

! HTTP request:

GET /courses/javascript HTTP/1.1

Host: www.softuni.bg

User-Agent: Mozilla/5.0

<CRLF>

The empty line denotes the end of the request header

! HTTP response:

HTTP/1.1 200 OK

Date: Mon, 5 Jul 2020 13:09:03 GMT

Server: Microsoft-HTTPAPI/2.0

Last-Modified: Mon, 12 Jul 2014 15:33:23 GMT

Content-Length: 54

<CRLF>

<html><title>Hello</title>

Welcome to our site</html>

The empty line denotes the end of the response header

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&rll=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%BA%D0%BA%D0%BD%D0%BC%D0%BC%D1%8F>

- Some invalid URLs:

[http://www.google.bg/search?q=C# .NET 4.0](http://www.google.bg/search?q=C%23+NET+4.0)

[http://www.google.bg/search?&q=бира](http://www.google.bg/search?q=%D0%B1%D0%BF%D1%80%D0%BE%D0%B1%D0%BA%D0%B0)

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          HTTP response headers
Server: Apache/2.2.14 (Linux)
Accept-Ranges: bytes
Content-Length: 84
Content-Type: text/html
<CRLF>
<html>
  <head><title>Test</title></head>
  <body>Test HTML page.</body>
</html>
```

HTTP response body

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>
  <h1>Not Found</h1>
  <p>The requested URL /img/logo.gif was not found on this server.</p>
  <hr><address>Apache/2.2.14 Server at Port
  80</address>
</body></html>
```

The HTTP response body

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** == Multi-Purpose Internet Mail Extensions
 - 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:

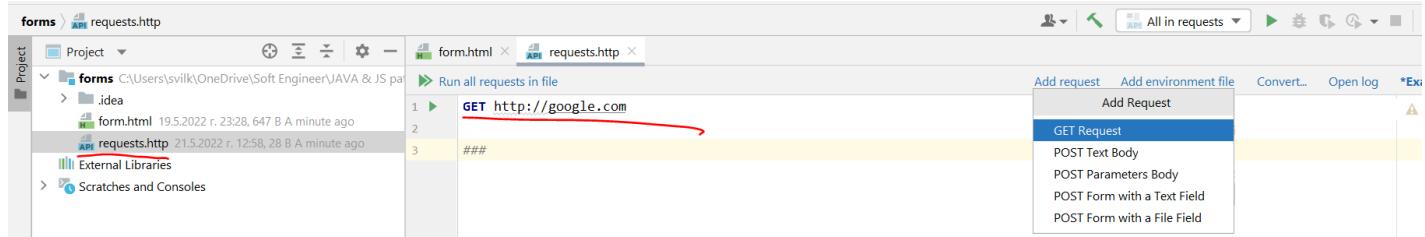
MIME Type / Subtype	Description
application/json	JSON data
image/png	PNG image
image/gif	GIF image
text/html	HTML
text/plain	Text
text/xml	XML
video/mp4	MP4 video

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

2.7. HTTP Tools

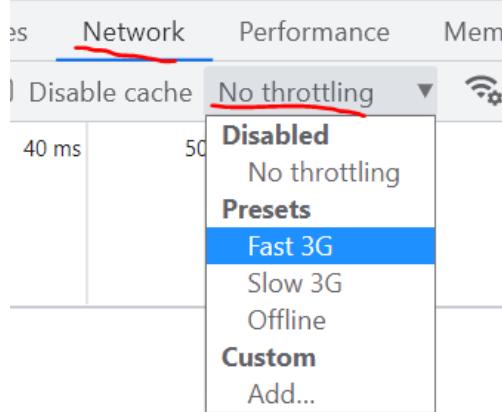
HTTP Tools for Developers:

- През IntelliJ



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

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



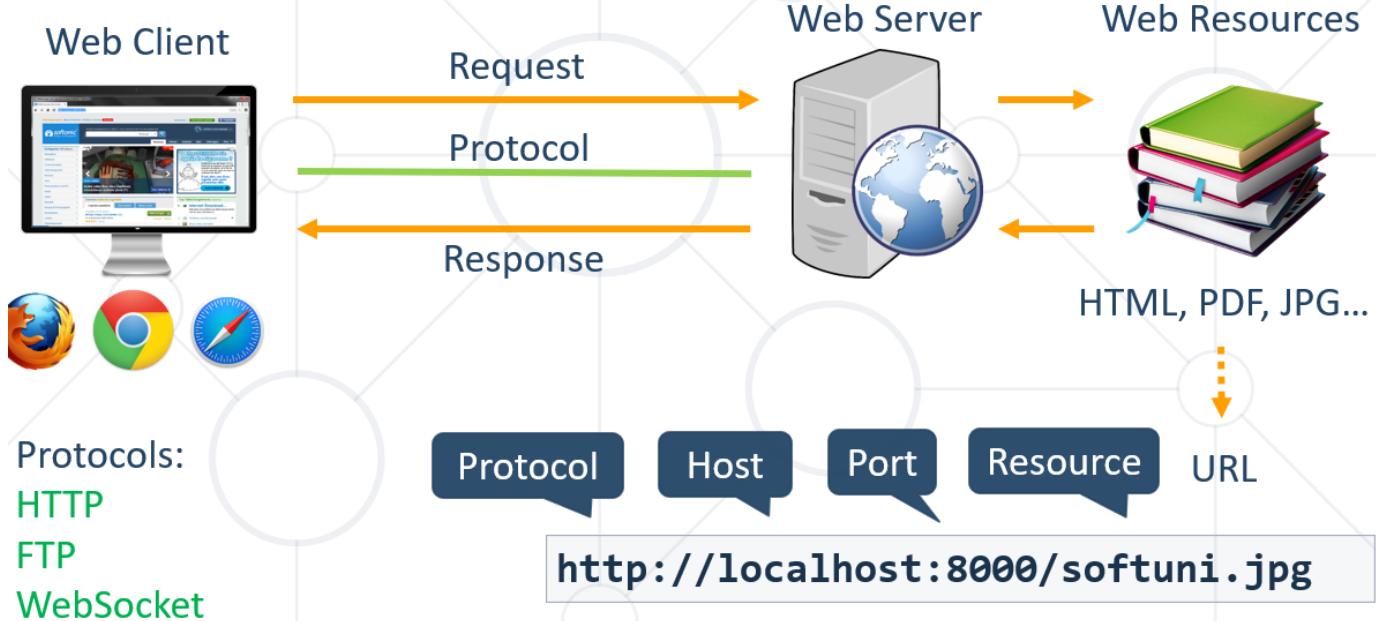
- Telnet
- През Ubuntu (Linux) и SSL връзка
- [Insomnia Rest](#)
- [Postman](#)
- [RESTClient](#)

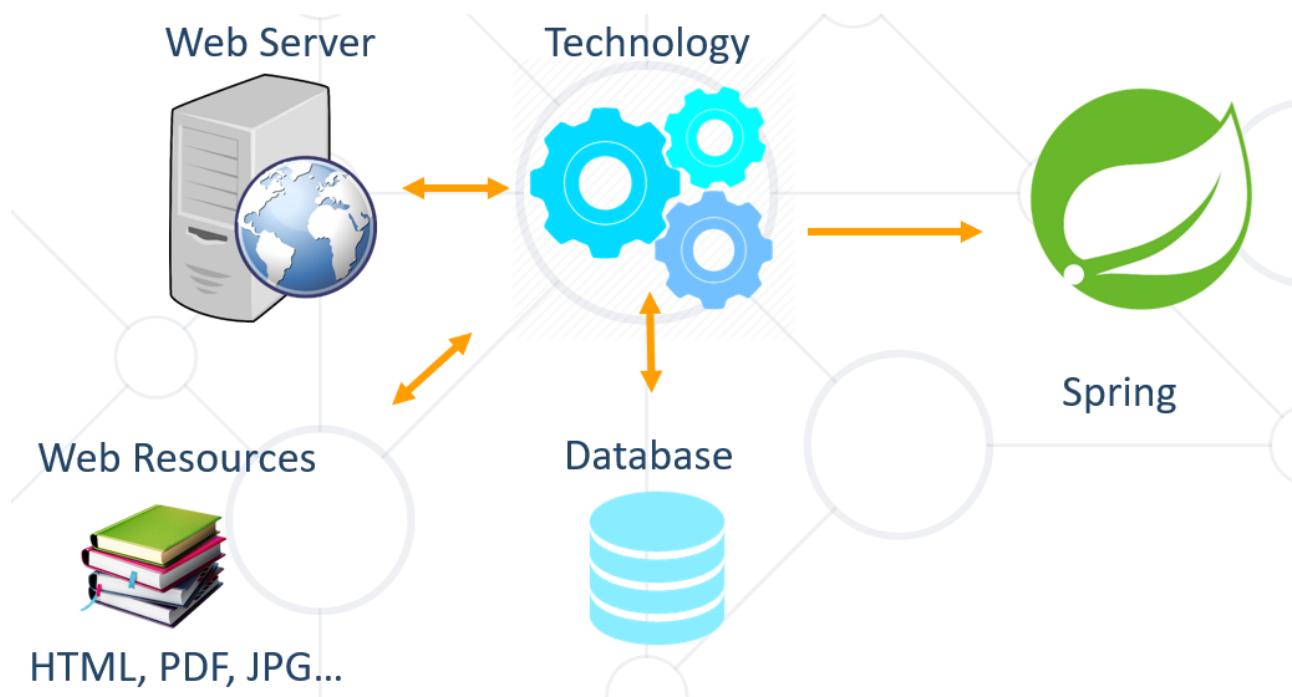
2.8. Web Server

What is a Web Server?

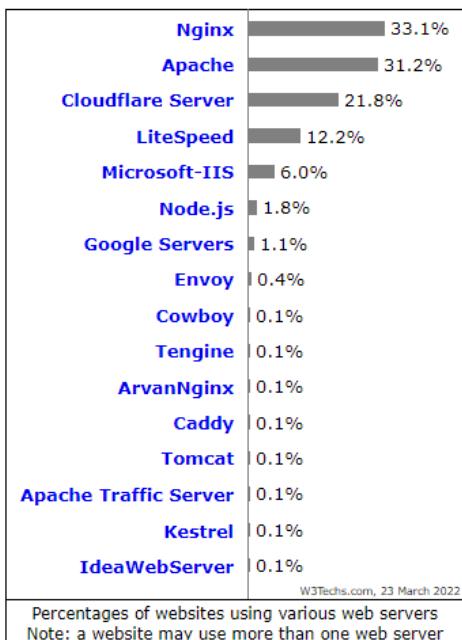
- Computer system that processes requests via **HTTP**, the basic network protocol

Web Server Work Model





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>

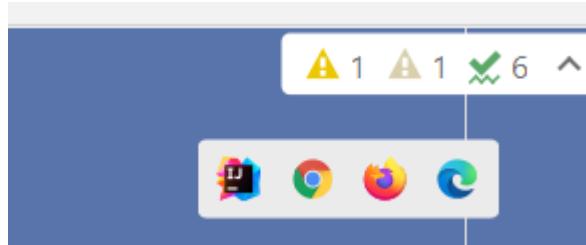
<script>
  function testMe() {
    for (let i = 0; i < 10; i++) {
      console.log("Test " + i);
    }

    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>

```

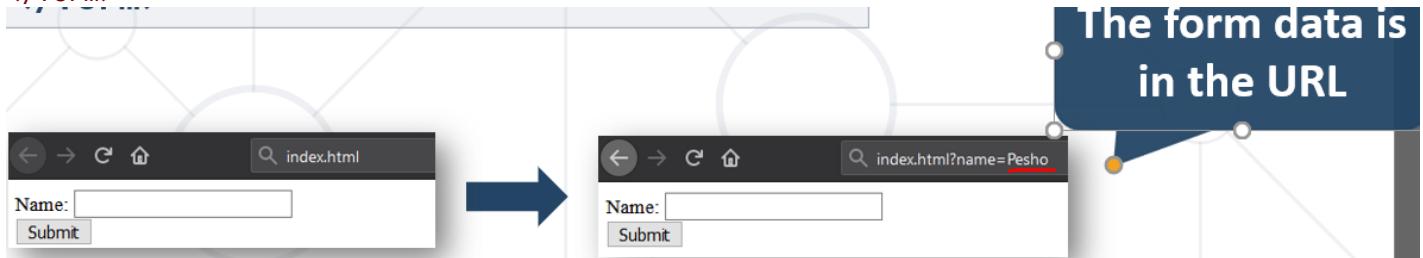


HTML Forms – Method Attribute

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

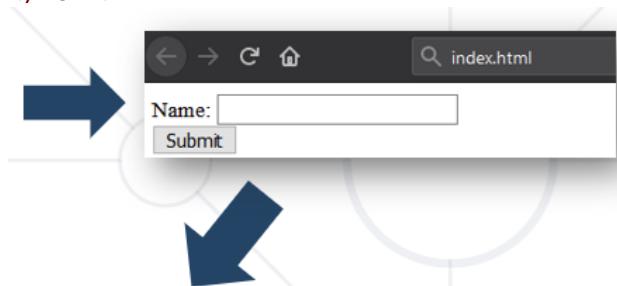
Get

```
<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>
```

A screenshot of a web browser window titled "index.cgi". It contains a form with two input fields. The first field is labeled "Name:" and has the value "Maria Smith". The second field is labeled "Age:" and has the value "19" with a small red checkmark next to it. Below the fields is a "Submit Query" button.



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?

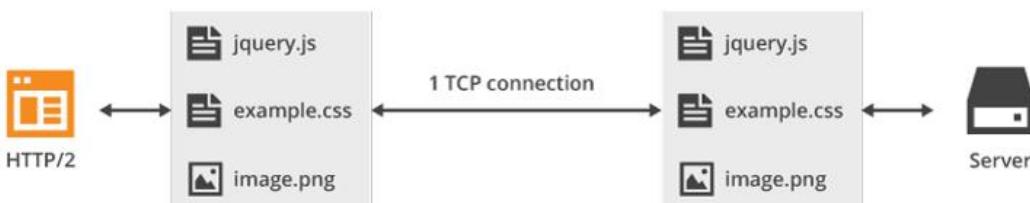
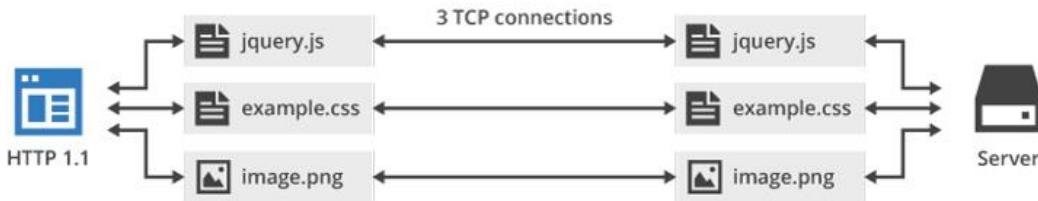
- **HTTP/2** is meant to erase the need of maintaining complex server infrastructures in order to perform well.
- **HTTP/2** communicates in binary data frames.
- **HTTP/2** introduces several **new important** elements
 - **HTTP/2 Multiplexing**
 - **HTTP/2 Header Compression**
 - **HTTP/2 Server Push**



HTTP/2 Multiplexing

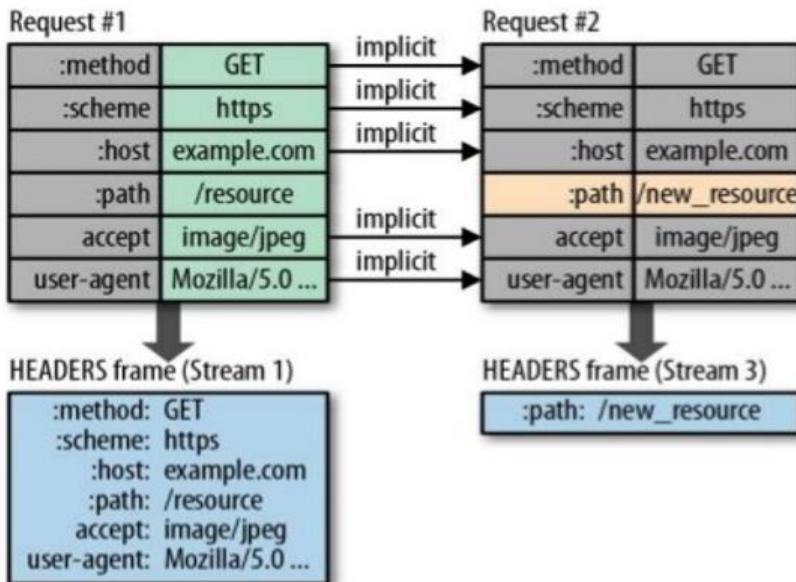
- The art of handling multiple streams over a **single** TCP connection.

Multiplexing



HTTP/2 Header Compression

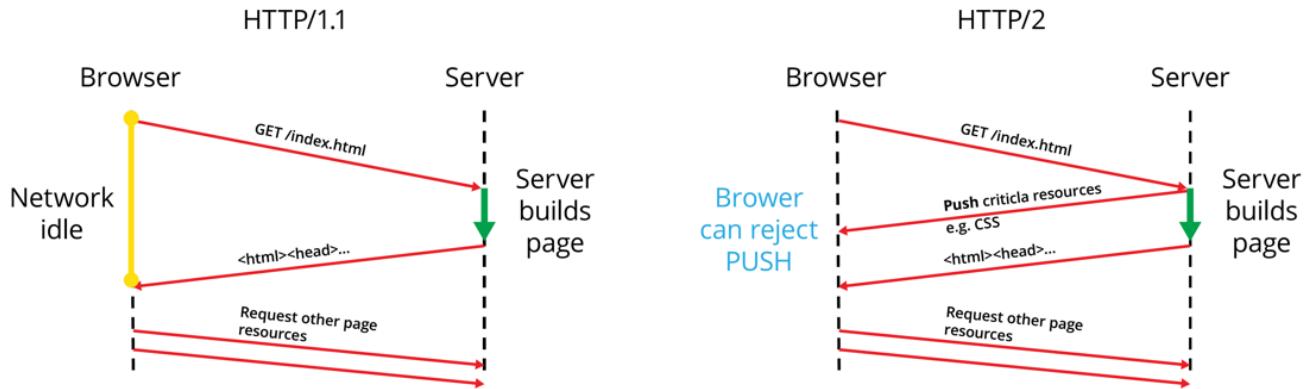
- HTTP/2** maintains a **HTTP Header Table** across requests.
- Optimizes communication drastically.**
- The process is essentially a **de-duplication**, rather than compression – не се изпращат части, които са redundant(излишни).



HTTP/2 Server Push – **TCP 3-way handshake**

HTTP/2 Server Push – TCP 3-way handshake – с един request ти връща много response-и за зареждане на нещата от страницата (в head-а на html-а каквото има да се зарежда примерно)

- **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 криптиранни 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 криптириания 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 подпись), 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/катинарчето в браузъра.

SHA256 и хеширащи алгоритми за пазене на пароли....

За разлика от хеширането, което е необратим процес, то криптирането 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. Dependency Injection

- Dependency injection is a design pattern whereby dependencies of an object are provided (“injected”) externally
- Dependency injection is a form of **IoC (Inversion of Control)** as defined by the SOLID principles
- Dependency Injection (DI) is not the only form of IoC, another example is the **template method** design pattern

Dependencies can be injected using different strategies:

- Setting a value on the field of a class directly (i.e. using reflection), also called **field injection**
- using a setter method in the class, also called **setter injection**
- By passing dependency to the constructor of the class, also called **constructor injection**
- By having the dependency inject itself to the object by having the object implement an interface with a setter method, also called **interface injection**
- Dependency Injection can be provided by an application utilities that are used to provide a mechanism for providing dependencies to object.
- However since this is a common activity for many applications a number of dependency injection frameworks provide this capability for applications
- Dependency injection provides a fundamental mechanism for **providing loose coupling** between objects.

Widely-used dependency injection frameworks for Java applications are:

- Spring framework
- Google Guice
- CDI/EJB (specifically in the context of JavaEE) - набор от спецификации надграждащи JavaSE и позволяващи реализация на нашето приложение върху Java Application Servers като например JBoss (WildFly), OracleWebLogic.
- Dagger (for Android applications)

4. Spring Core

4.1. General

- **Spring Core** framework is a DI (dependency injection) **container**

- It is responsible to create and maintain objects and their dependencies - предоставят ни се директно зависимости/библиотеки за даден вид приложение - за база данни, за web приложение, за message broker, и т.н.
- All technologies from the Spring stack are built and make use of Spring Core framework **container**

4.2. Without Spring context Demo

```

public class Application {
    public static void main(String[] args) {
        createApplication(args);
    }

    private static void createApplication(String[] args) {
        String location = args[0]; // db, file
        String message = args[1];

        import org.example.service.DatabasePersistenceManager;
        import org.example.service.FilePersistenceManager;
        import org.example.service.PersistenceManager;
        import org.example.util.DatabaseUtils;
        import org.example.util.FileUtils;

        no usages
        public class Application {
            no usages
            @ public static void main(String[] args) {
                // db, file
                String location = args[0];✓
                String message = args[1];✓

                PersistenceManager persistenceManager
                switch (location){
                    case "db": {
                        persistenceManager = new DatabasePersistenceManager();
                        ((DatabasePersistenceManager) persistenceManager).setDbUtils(new DatabaseUtils());
                        break;
                    }
                    case "file": {
                        persistenceManager = new FilePersistenceManager();
                        ((FilePersistenceManager) persistenceManager).setFileUtils(new FileUtils());
                        break;
                    }
                }

                persistenceManager.save(message);
            }
        }
    }
}

public interface PersistenceManager {
    public void save(String message);
}

```

The screenshot shows the IntelliJ IDEA interface with the code editor on the left and the 'Edit Run Configuration' dialog on the right. The code editor displays a Java application with imports and a main method. The main method calls a static helper method 'createApplication' with two arguments: 'db' and 'some_message_arg_1'. The 'Edit Run Configuration' dialog is titled 'Edit Run Configuration: 'Application''. It has fields for 'Name' (set to 'Application'), 'Run on' (set to 'Local machine'), and 'Build and run' (containing the command 'java 17 SDK of 'springcore'' followed by 'org.example.Application'). The argument 'db some_message_arg_1' is highlighted with a red circle and a red arrow pointing to it. Other fields in the dialog include 'Working directory' and 'Environment variables'.

```

}

-----
import java.util.logging.Logger;

public class DatabasePersistenceManager implements PersistenceManager {
    private static final Logger LOGGER =
Logger.getLogger(DatabasePersistenceManager.class.getName());
    private DatabaseUtils dbUtils;

    public void save(String message) {
        dbUtils.persist(message);
        LOGGER.info(String.format("Saving message: %s to database", message));
    }

    public void setDbUtils(DatabaseUtils dbUtils) {
        this.dbUtils = dbUtils;
    }
}
-----
import java.util.logging.Logger;

public class FilePersistenceManager implements PersistenceManager {
    public static final Logger LOGGER =
Logger.getLogger(FilePersistenceManager.class.getName());
    private FileUtils fileUtils;

    public void save(String message) {
        fileUtils.save(message);
        LOGGER.info(String.format("Saving message: %s to database", message));
    }

    public void setFileUtils(FileUtils fileUtils) {
        this.fileUtils = fileUtils;
    }
}

```

4.3. Spring context

- The entry point to the Spring framework is the Spring application context
- The application context (главния клас от който пускате приложението) manages the various objects used by the application through Spring (also called **beans**)
- The Spring context is bound to a particular mechanism for management of dependencies
- Built-in mechanism for management of dependencies includes:
 - XML files (using ClassPathXmlApplicationContext instance)
 - Annotations (using AnnotationConfigApplicationContext instance)

With XML configuration

Info

Spring DI framework can be supplied by the following dependencies:

```

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>

```

```

<artifactId>spring-core</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>${spring.version}</version>
</dependency>

ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("context.xml");
или
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(Application.class.getPackage().getName());
//Application е мястото откъдето се стартира приложението

```

```

ExampleService service = context.getBean(ExampleService.class);
service.someMethod();
context.close();

```

По подразбиране beans в Спринг са singleton scope.

```

context.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
       xmlns:context="http://www.springframework.org/schem
a/context"
       xsi:schemaLocation="http://www.springframework.org/schema/
beans
       http://www.springframework.org/schema/beans/spring-
beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-
context.xsd">
    <bean id="exampleservice"
          class="spring.examples.ExampleService">
    </bean>
</beans>

```

Demo:

```

public class Application {
    public static void main(String[] args) {
        //      createApplication(args);
        createApplicationUsingXmlContext(args);
    }

    private static void createApplicationUsingXmlContext(String[] args) {
        String location = args[0]; // db, file
        String message = args[1];
    }
}

```

```

        ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("context.xml");
        PersistenceManager manager = context.getBean(location, PersistenceManager.class);
        manager.save(message);
        context.close();
    }
}

-----
src.main.resources->context.xml
<!--http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation)-->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="db" class="org.example.service.DatabasePersistenceManager">
        <property name="dbUtils" ref="dbUtils"></property>
//name реферира към сетъра setDbUtils. ref реферира към друг bean dbUtils
    </bean>
    <bean id="file" class="org.example.service.FilePersistenceManager">
        <property name="fileUtils" ref="fileUtils"></property>
    </bean>
    <bean id="dbUtils" class="org.example.util.DatabaseUtils"></bean>
    <bean id="fileUtils" class="org.example.util.FileUtils"></bean>
</beans>
-----
public interface PersistenceManager {
    public void save(String message);
}

import java.util.logging.Logger;

public class DatabasePersistenceManager implements PersistenceManager {
    private static final Logger LOGGER =
Logger.getLogger(DatabasePersistenceManager.class.getName());
    private DatabaseUtils dbUtils;

    public void save(String message) {
        dbUtils.persist(message);
        LOGGER.info(String.format("Saving message: %s to database", message));
    }

    public void setDbUtils(DatabaseUtils dbUtils) {
        this.dbUtils = dbUtils;
    }
}

import java.util.logging.Logger;

public class FilePersistenceManager implements PersistenceManager {
    public static final Logger LOGGER =
Logger.getLogger(FilePersistenceManager.class.getName());
    private FileUtils fileUtils;
}

```

```

public void save(String message) {
    fileUtils.save(message);
    LOGGER.info(String.format("Saving message: %s to database", message));
}

public void setFileUtils(FileUtils fileUtils) {
    this.fileUtils = fileUtils;
}
}

```

With Annotation

Info

```
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext("root.package");
```

ИЛИ

```
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(Application.class.getPackage().getName());
//Application е мястото откъдето се стартира приложението
```

```
ExampleService service = context.getBean(ExampleService.class);
service.someMethod();
context.close();
```

- To designate(посочим) a bean class the @Component annotations can be used

```
import org.springframework.stereotype.Component;
```

```
@Component
public class ExampleService {
    public void someMethod(){
        return;
    }
}
```

- Another way to retrieve a bean instance is to annotate a method in a @Configuration class with the @Bean annotation (`createExampleService` method is the so called factory method):

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class ServiceConfiguration {

    @Bean
    public ExampleService createExampleService(){
        return new ExampleService();
    }
}
```

Kind of injections in Spring

// Field injection

```
@Autowired(required = true)
@Qualifier("createSomeLibraryClassBean2")
private SomeLibraryClass instance;
```

```
// Constructor injection – which of the two is it?  
@Autowired  
public DatabasePersistenceManager(@Autowired DatabaseUtils dbUtils) {  
    this.dbUtils = dbUtils;  
}
```

```
// Setter injection  
@Autowired  
public void setDbUtils(DatabaseUtils dbUtils) {  
    this.dbUtils = dbUtils;  
}
```

Demo Annotations with @Component

```
public class Application {  
    public static void main(String[] args) {  
        // createApplication(args);  
        // createApplicationUsingXmlContext(args);  
        createApplicationUsingAnnotationContext(args);  
    }  
  
    private static void createApplicationUsingAnnotationContext(String[] args) {  
        String location = args[0]; // db, file  
        String message = args[1];  
  
        AnnotationConfigApplicationContext context = new  
        AnnotationConfigApplicationContext("org.example");  
        PersistenceManager manager = context.getBean(location, PersistenceManager.class);  
        manager.save(message);  
        context.close();  
    }  
}  
  
----  
public interface PersistenceManager {  
    public void save(String message);  
}  
  
----  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import java.util.logging.Logger;  
  
@Component("db") //името на бийна е db  
public class DatabasePersistenceManager implements PersistenceManager {  
    private static final Logger LOGGER =  
    Logger.getLogger(DatabasePersistenceManager.class.getName());  
  
    private DatabaseUtils dbUtils;  
  
    public DatabasePersistenceManager(@Autowired DatabaseUtils dbUtils){  
        this.dbUtils = dbUtils;  
    }  
  
    public void save(String message) {
```

```

        dbUtils.persist(message);
        LOGGER.info(String.format("Saving message: %s to database", message));
    }
}

-----
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Component("file")
public class FilePersistenceManager implements PersistenceManager {
    public static final Logger LOGGER =
Logger.getLogger(FilePersistenceManager.class.getName());

    @Autowired
    private FileUtils fileUtils;

    public void save(String message) {
        fileUtils.save(message);
        LOGGER.info(String.format("Saving message: %s to database", message));
    }

    public void setFileUtils(FileUtils fileUtils) {
        this.fileUtils = fileUtils;
    }
}

-----
import org.springframework.stereotype.Component;

@Component("dbUtils")
public class DatabaseUtils {

    public void persist(String message) {

    }
}

-----
import org.springframework.stereotype.Component;

@Component("fileUtils")
public class FileUtils {

    public void save(String message) {

    }
}

```

Demo Annotations with @Configuration and @Bean

Използва се когато имаме външна библиотека например, и искаме да я направим да работи в Spring context. В този случай не анотираме класа WebServicePersistenceManager.

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class BeanConfig {

    @Bean
    public WebServicePersistenceManager createWebServicePersistenceManager() {
        return new WebServicePersistenceManager();
    }

}

public class WebServicePersistenceManager implements PersistenceManager{
    private static final Logger LOGGER =
    Logger.getLogger(WebServicePersistenceManager.class.getName());

    @Override
    public void save(String message) {
        LOGGER.info(String.format("Sending message: %s to web service", message));
    }
}

```

4.4. Spring beans

<https://docs.spring.io/spring-framework/reference/core/beans.html>

Info

As a summary, we can create Spring beans in any of the following ways:

- through bean elements in **XML configuration**
- through **@Component-annotated classes**
- through **@Bean-annotated factory methods**

Spring XML and annotation-based configuration can be mixed, but in general it is not a good practice.

A bean can be injected as a dependency of another bean using any of the following strategies:

- constructor injection: passing the bean as an argument
- setter/field injection: using a proper setter or reference as the injection target
- factory method: passing the bean as an argument to a factory method that creates the instance

Dependency Injection can be performed by means of the following annotations, i.e. if the Spring project is old and has integration/libraries for JavaEE or Java SE (it has it by default), then we may use **@Inject** or **@Resource** in a Spring project :

- **@Autowired** (provided by Spring framework) - on a constructor or on a setter (or on a field)
- **@Inject** (provided by CDI from JavaEE) - we make it on the field here usually
- **@Resource** (provided by Java SE)

The bean to inject might be determined by type, by name or by qualifier.

Bean definition

Each Spring bean is represented by a **BeanDefinition** instance that provides metadata about the bean such as:

- bean configuration (such as scope)
- bean class and creation mechanism
- bean dependencies
- bean lifecycle

В редки случаи може да се наложи да правим наши си BeanDefinition класове, които да използваме

```
BeanDefinition definition = context.getBeanFactory().getBeanDefinition(location);
System.out.println(definition);
// Generic bean: class [org.example.service.DatabasePersistenceManager];
scope=singleton; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0;
// autowireCandidate=true; primary=false; factoryBeanName=null;
factoryMethodName=null; initMethodNames=null; destroyMethodNames=null;
// defined in file [/media/svilen/404677E04677D55E/SVILEN/bgjug/Spring Core
framework/springcore/target/classes/org/example/service/DatabasePersistenceManager.class
]
```

Bean scopes

A bean scope defines how long a Spring bean exists in a particular context:

- **Singleton (default)** - (Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
- **Prototype** - Scopes a single bean definition to any number of object instances.
- **Request** - за уеб приложения, Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
- **Session** - Scopes a single bean definition to the lifecycle of an HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
- **Application** - Scopes a single bean definition to the lifecycle of a **ServletContext**. Only valid in the context of a web-aware Spring ApplicationContext.
- **WebSocket** - Scopes a single bean definition to the lifecycle of a WebSocket. Only valid in the context of a web-aware Spring ApplicationContext.

When using annotation-based configuration, a scope is specified with the @Scope annotation.

```
import org.springframework.beans.factory.config.ConfigurableBeanFactory;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component("db")
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class DatabasePersistenceManager implements PersistenceManager {
```

Ако е Prototype, прави две отделни инстанции на бийна, които имат различни id-та в паметта

```
System.out.println(System.identityHashCode(manager)); //връща id-то на обекта от Java
HEAP паметта
System.out.println(System.identityHashCode(manager2));
....
```

823723302
1714078840

4.5. Customizing the bean lifecycle

InitializingBean and DisposableBean

The bean lifecycle can be customized (веднага след като бъде готов бийна и преди да бъде унищожен) by:

- Implementing an interface such as InitializingBean (afterPropertiesSet() method) or DisposableBean (destroy() method)

```
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;
```

```
@Component("db")
public class DatabasePersistenceManager implements PersistenceManager, InitializingBean,
DisposableBean {

@Override
public void afterPropertiesSet() throws Exception {} //изпълнява се след като са inject-
нати полетата в дадения бийн

@Override
public void destroy() throws Exception {} //преди да бъде унищожен бийна от контекста –
извиква се само за Singleton бийнове
```

@PostConstruct and @PreDestroy

- Annotating methods with @PostConstruct and @PreDestroy annotations (preferable as it avoids coupling with Spring interfaces) - тези 2 анатации идват от JavaEE

```
<dependency>
  <groupId>javax.annotation</groupId>
  <artifactId>javax.annotation-api</artifactId>
  <version>1.3.2</version>
</dependency>
```

```
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
```

```
@PostConstruct
public void postConstruct() {}
```

When we annotate a method in Spring Bean with @PostConstruct annotation, **it gets executed after the spring bean is initialized**. We can have only one method annotated with @PostConstruct annotation.

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

```
@PreDestroy
public void preDestroy() {}
```

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

Bean post processing

- One or more bean post processor (instances of the **BeanPostProcessor** interface) can be defined to provide additional **pre** and **post** initialization logic
- The methods that need to be implemented are:

- o `postProcessBeforeInitialization` - тук може да изпълним логика още преди да е инициализиран бийна!!!
- o `postProcessAfterInitialization` - след като бийна е вече инициализиран

Това е още един начин да слагаме логика преди инициализация на bean и преди унищожаване на bean!

```
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Component
public class LoginProcessor implements BeanPostProcessor {
    private static final Logger LOGGER =
Logger.getLogger(LoginProcessor.class.getName());

    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
        LOGGER.info("Before bean %s is initialized ...".formatted(beanName));
        return BeanPostProcessor.super.postProcessBeforeInitialization(bean, beanName);
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {
        LOGGER.info("After bean %s is initialized ...".formatted(beanName));
        return BeanPostProcessor.super.postProcessAfterInitialization(bean, beanName);
    }
}
```

```
Nov 20, 2023 4:27:37 PM org.example.processors.LoginProcessor postProcessBeforeInitialization
INFO: Before bean dbUtils is initialized ...
Nov 20, 2023 4:27:37 PM org.example.processors.LoginProcessor postProcessAfterInitialization
INFO: After bean dbUtils is initialized ...
```

@Lazy

Бийна ще се инициализира при първото извикване на метод от него

```
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import java.util.logging.Logger;

@Component("db")
@Lazy
public class DatabasePersistenceManager implements PersistenceManager
```

@DependsOn

Когато искаме определен ред на инициализация на бийновете. Кой след кой да се инициализират

```

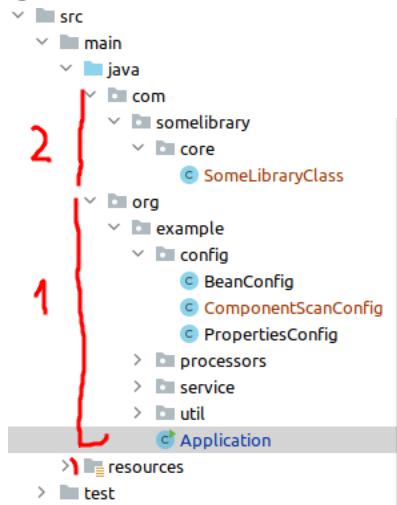
import org.springframework.context.annotation.DependsOn;
import org.springframework.stereotype.Component;

@Component("dbUtils")
@DependsOn({"fileUtils", })
public class DatabaseUtils {

    @Autowired
    @Autowired(required = true) //Declares whether the annotated dependency is required.
    Default is true. Ако му зададем false, няма да разберем, че не ни работи приложението
    private SomeLibraryClass instance;
}

```

@Qualifiers



.....

```

package org.example.config;

import com.somelibrary.core.SomeLibraryClass;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan("com.somelibrary.core")
public class ComponentScanConfig {

```

// Когато някои от класовете в дадената библиотека не са готови бийнове, то можем да ги направим като такива ето така:

```

    @Bean
    public SomeLibraryClass createSomeLibraryClassBean() {
        return new SomeLibraryClass();
    }

```

```

    @Bean
    public SomeLibraryClass createSomeLibraryClassBean2() {
        return new SomeLibraryClass();
    }
}

```

```

}

.....
import org.springframework.beans.factory.annotation.Qualifier;

@Autowired
@Qualifier("createSomeLibraryClassBean2") //посредством името на метода на бийна ==
името на бийна
private SomeLibraryClass instance;

```

Или можем и така

```

.....
@Autowired
private SomeLibraryClass instance;

package org.example.config;

import com.somelibrary.core.SomeLibraryClass;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;

@Configuration
@ComponentScan("com.somelibrary.core")
public class ComponentScanConfig {

    // Когато някои от класовете в дадената библиотека не са готови бийнове, то можем да
    // ги направим като такива ето така:
    @Bean
    @Primary
    public SomeLibraryClass createSomeLibraryClassBean() {
        return new SomeLibraryClass();
    }

    @Bean
    public SomeLibraryClass createSomeLibraryClassBean2() {
        return new SomeLibraryClass();
    }
}

```

4.6. Component scan

- The packages used to scan for Spring components are supplied when the Spring context is created
- In addition, root packages to be scanned for Spring components can be provided by the `@ComponentScan` annotation on a `@Configuration`-annotated class that is already scanned during context initialization - когато искаме да зададеме допълнителни пакети с бийнове (ако имаме няколко места в нашия проект)
- Сканира за класове анотирани като бийнове и от друга подпапка на нашия проект

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

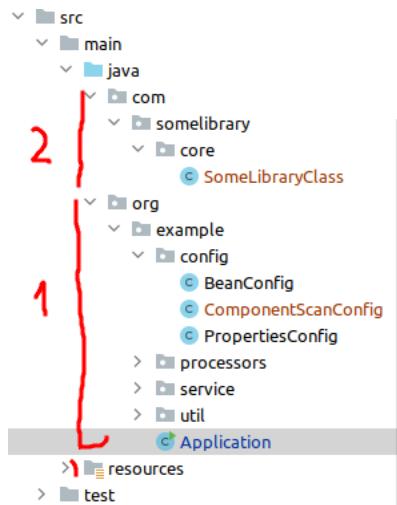
```

```

@Configuration
@ComponentScan("com.somelibrary.core")
public class ComponentScanConfig {

    // Когато някои от класовете в дадената библиотека не са готови бийнове, то можем да ги направим като такива ето така:
    @Bean
    public SomeLibraryClass createSomeLibraryClassBean() {
        return new SomeLibraryClass();
    }
}

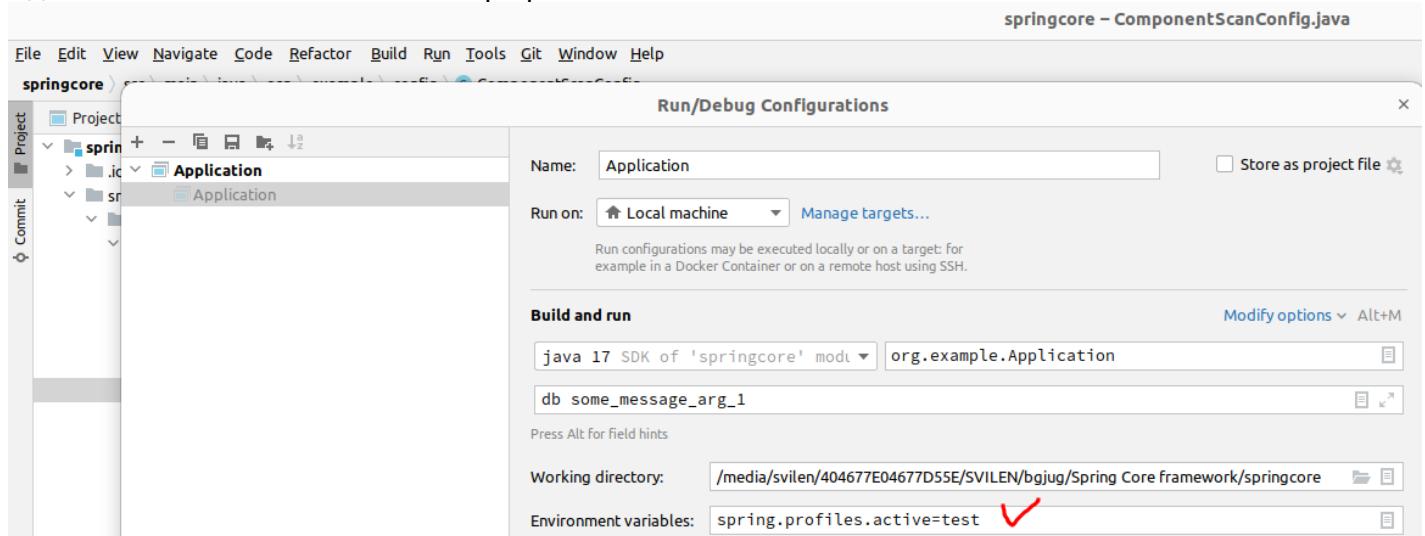
```



4.7. Using profiles/профили

Можем да окажем кои бийнове да се използват/зареждат за даден профил.

Един от начините за оказване на профили е от JVM environment variables:



Indicates that a component is eligible for registration when one or more specified profiles are active. A profile is a named logical grouping that may be activated programmatically via `ConfigurableEnvironment setActiveProfiles` or declaratively by setting the `spring.profiles.active` property as

a JVM system property, as an environment variable, or as a Servlet context parameter in web.xml for web applications. Profiles may also be activated declaratively in integration tests via the @ActiveProfiles annotation.

```
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;

@Component("db")
//@Profile("local || test") //spring expression language support
@Profile({"local", "test"})
public class DatabasePersistenceManager implements PersistenceManager
```

4.8. Event handling

- The application context also provides **publish and subscribe mechanism** through events
- Beans that implement the ApplicationListener interface may consume triggered events

```
36 * @see org.springframework.context.event.ApplicationEventMulticaster
37 * @see org.springframework.context.event.EventListener
38 */
39 @FunctionalInterface
40 public interface ApplicationListener<E extends ApplicationEvent> extends EventListener {
41
42    /**
43     * Handle an application event.
44     * @param event the event to respond to
45     */
46    void onApplicationEvent(E event);
47
48 }
```



- Друг механизъм за взаимодействие между beans, който не е свързан с Dependency Injection, а по-скоро е като publish and subscribe mechanism
- Events are either:
 - Built-in (such as **ContextStartedEvent** or **ContextStoppedEvent**)
 - custom

Пример:

```
package com.somelibrary.core;

import org.springframework.context.ApplicationEvent;

public class MessageSavedEvent extends ApplicationEvent {
    public MessageSavedEvent(Object source) {
        super(source);
    }
}
.....
package org.example.service;

import com.somelibrary.core.MessageSavedEvent;

@Component("db")
public class DatabasePersistenceManager implements PersistenceManager {
```

```

private static final Logger LOGGER =
Logger.getLogger(DatabasePersistenceManager.class.getName()) ;

@Autowired
private DatabaseUtils dbUtils;

@Autowired
private ApplicationEventPublisher publisher;

public void save(String message) {
    publisher.publishEvent(new MessageSavedEvent(""));
    dbUtils.persist(message);
    LOGGER.info(String.format("Saving message: %s to database", message));
}

.....
package org.example.service;

import com.somelibrary.core.MessageSavedEvent;
import org.springframework.context.ApplicationListener;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Component("file")
public class FilePersistenceManager implements PersistenceManager,
ApplicationListener<MessageSavedEvent> {
    public static final Logger LOGGER =
Logger.getLogger(FilePersistenceManager.class.getName());

    @Override
    public void onApplicationEvent(MessageSavedEvent event) {
        LOGGER.info("Message event received in file persistence manager bean");
    }
}
.....
package org.example.util;

import com.somelibrary.core.MessageSavedEvent;
import org.springframework.context.ApplicationListener;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Component("fileUtils")
public class FileUtils implements ApplicationListener<MessageSavedEvent> {
    public static final Logger LOGGER = Logger.getLogger(FileUtils.class.getName());

    public void save(String message) {}

    @Override
    public void onApplicationEvent(MessageSavedEvent event) {
        LOGGER.info("Message event received in file utils bean....");
    }
}

```

4.9. Application properties

@PropertySource

- Different sources of application properties can be included for use by the application
- They can be specified in a XML-based configuration using a **<property-placeholder>** element
- In annotation-based configuration, property sources are specified with the **@PropertySource** annotations

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Configuration
@PropertySource("classpath:application.properties")
public class PropertiesWithJavaConfig {
    @Bean
    public ExampleService createExampleService(){
        return new ExampleService();
    }
}
```

@Value

- To inject a value from a property source, the **@Value** annotation can be used
- ```
@org.springframework.beans.factory.annotation.Value("${jdbc.url}")
private String jdbcUrl;
```

By convention, **application.properties** file is the default properties file used by Spring framework and does not need to be registered as a property source when the file is at the correct class path.

### Demo

```
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Configuration
@PropertySource("classpath:application.properties")
@PropertySource("classpath:other.properties")
public class PropertiesConfig {

src.main.resources->application.properties
example.key=some_value
jdbc.url=some_jdbc_url

src.main.resources->other.properties
other.key=other_some_value
```

### Environment

- Application properties from property sources can also be retrieved by injecting an **Environment** instance

```
@Autowired
private org.springframework.core.env.Environment environment;
```

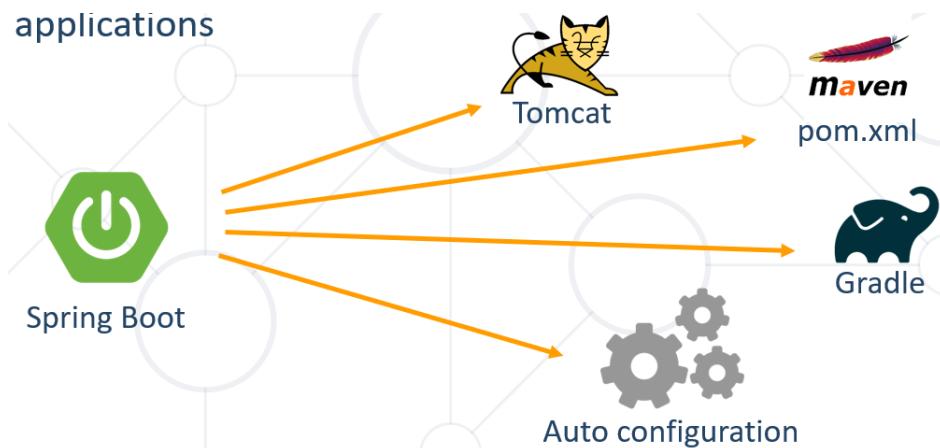
Може да чете от повече от един конфигурационен файл

```
environment.getProperty("jdbc.url");
environment.getProperty("other.key");
```

## 5. Spring Boot Introduction

### 5.1. What is Spring Boot?

Spring Boot Overview



- Spring boot takes a step further in simplifying deployment and configuration of applications that use Spring framework
- Goals of Spring Boot
  - A radically **faster and widely accessible getting-started** experience
  - **Opinionated, but get out of the way quickly** - идва с готова конфигурация, но лесно можем да махнем/променим нещо от нея (например GSON библиотеката идва конфигурирана от Spring като я вкарваме като dependency, но ние можем да си я настроим ръчно като по този начин не се съобразяваме с това което Спринг са настроили)
  - Provide a range of **common non-functional features** - monitoring, Spring Actuator, way of deploying
  - **No code generation** - в други фреймърци при първоначално стартиране се генерира доста излишен код
- In particular, it provides capabilities such as:
  - Automatic discovery of application.properties file and other places we have config files/packages
  - Automatic configuration of the Spring application (such as running an embedded Tomcat for deployment of @RestController classes - server to listen to http endpoints)
- Главният клас, от който се стартира приложението е препоръчително да бъде на root package-a!!! **{@SpringBootApplication}** анотацията сканира всичко с **@ComponentScan** от текущата главна директория и всички поддиректории}

```
@SpringBootApplication
public class BootdemoApplication {

 public static void main(String[] args) {
 SpringApplication.run(BootdemoApplication.class, args);
 }

}

bg
+- jug
 +- store
 -- JugStore.java
 |
 +- product
 -- Product.java
 -- ProductController.java
 -- ProductService.java
 -- ProductRepository.java
 |
 +- order
 -- Order.java
 -- OrderController.java
 -- OrderService.java
 -- OrderRepository.java
```

Main class  
Recommended: root package above

Default package  
Recommended: do not use



## Entry point

A typical Spring Boot application entrypoint looks like the following:

```
@org.springframework.boot.autoconfigure.SpringBootApplication
public class Application {
```

```
 public static void main(String[] args) {
 org.springframework.boot.SpringApplication.run(Application.class, args);
 }
}
```

## Creating Spring Boot Project

- Just go to <https://start.spring.io/> or open IntelliJ
- Create it via Spring initializer



Project  Maven Project  Gradle Project Language  Java  Kotlin  Groovy

Spring Boot  3.0.0 (SNAPSHOT)  3.0.0 (M3)  2.7.1 (SNAPSHOT)  2.7.0  2.6.9 (SNAPSHOT)  2.6.8

Project Metadata

Group: bg.softuni  
Artifact: intro  
Name: intro  
Description: Demo project for Spring Boot  
Package name: bg.softuni.intro  
Packaging:  Jar  War  
Java:  18  17  11  8

Dependencies

**Spring Web** WEB   
Build web, including RESTful applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Boot DevTools** DEVELOPER TOOLS   
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

**Spring Data JPA** SQL   
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

**ADD DEPENDENCIES...** CTRL + B

**GENERATE** CTRL + d **EXPLORE** CTRL + SPACE **SHARE...**

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

War = webarchive

## Spring Dev Tools

- ✗ Spring Web
- ✗ Thymeleaf
- ✗ Spring Boot DevTools ✗
- ✗ Spring Data JPA
- ✗ MySQL Driver
- ✗ Validation

- 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")
}
```

## build.gradle

```
developmentOnly 'org.springframework.boot:spring-boot-devtools'
```

При промяна на класпътя (build.classes.\* за Gradle), има два клас лоъдъра, благодарение на които не се налага да рестартираме ръчно приложението (само при разработка). Реално се презареждат само класовете на нашето приложение, а не всички класове!

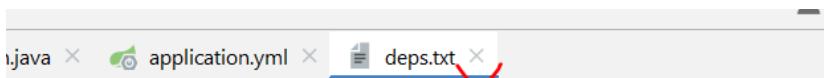
В application.yaml файла например пък можем да изключим кеширането на дадени thymeleaf templates.

Example of Gradle

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

В терминала пишем:

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



```
dependencies

 classpath - Configuration for Spring Boot archive artifacts. (n)
 dependencies

 classpath - Compile classpath for source set 'main'.
 .springframework.boot:spring-boot-starter-data-jpa -> 2.7.0
 - org.springframework.boot:spring-boot-starter-aop:2.7.0
 +--- org.springframework.boot:spring-boot-starter:2.7.0
 | +--- org.springframework.boot:spring-boot:2.7.0
 | | +--- org.springframework:spring-core:5.3.20
 | | | \--- org.springframework:spring-jcl:5.3.20
 | | \--- org.springframework:spring-context:5.3.20
```

**build.gradle file**

```
plugins {
 id 'org.springframework.boot' version '2.7.0'
 id 'io.spring.dependency-management' version '1.0.11.RELEASE' //грижи се за версияте
 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' //нама версии тута 😊
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>
 </dependency>
 </dependencies>
</pre>

```

```

<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>

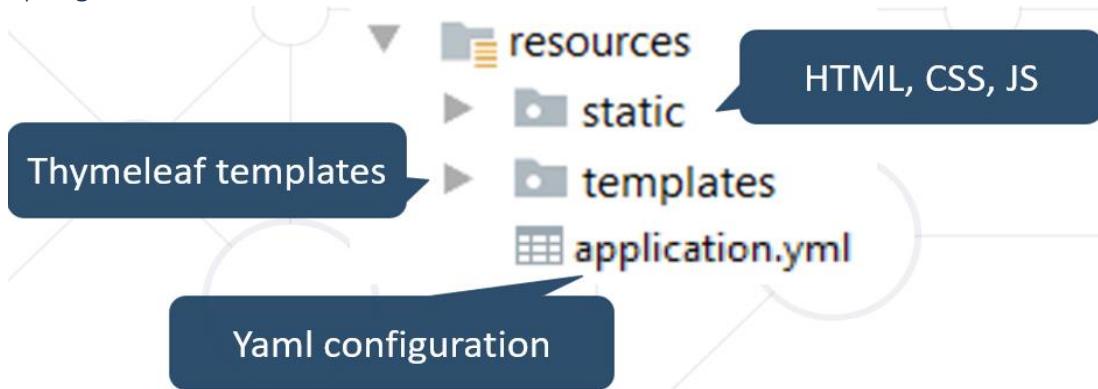
</dependencies>

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

</project>

```

## Spring Resources

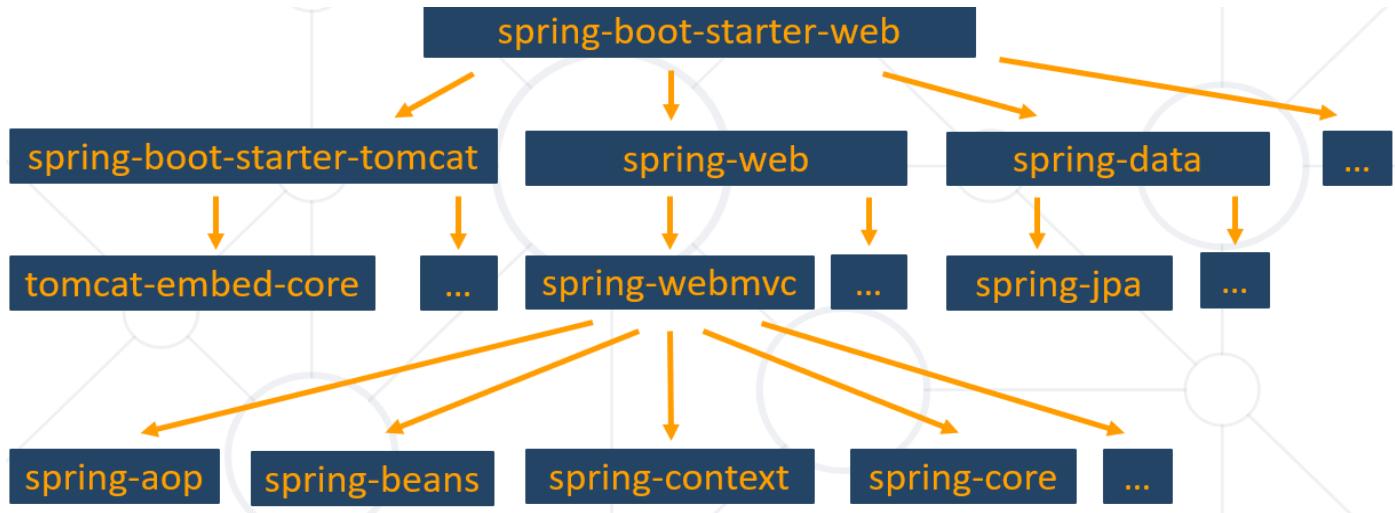


## Spring Boot Main Components

- 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 is provided by any of the below starter dependencies:
  - spring-boot-starter-web
  - spring-boot-starter-test
  - spring-boot-starter-jdbc
  - spring-boot-starter-data-jpa
  - spring-boot-starter-email
  - a number of others



## Spring Boot CLI (Command Line Interface)

We can run the application from the IntelliJ Idea Run/Debug/Stop services. However, we can use the CLI:

- **Command Line Interface** - Spring Boot software to run and test Spring Boot applications  
Още един начин да се стартира проект.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\teodo\Desktop\spring-1.5.2.RELEASE\bin>spring init -d=web, data-jpa
Using service at https://start.spring.io
Project extracted to 'C:\Users\teodo\Desktop\spring-1.5.2.RELEASE\bin\data-jpa'
C:\Users\teodo\Desktop\spring-1.5.2.RELEASE\bin>
```

We can run it with **maven CLI command**:

**mvn spring-boot:run**

## Spring Boot Actuator

- Expose different types of information about the **running application** (see more in chapter Deployment, the section for Actuator )

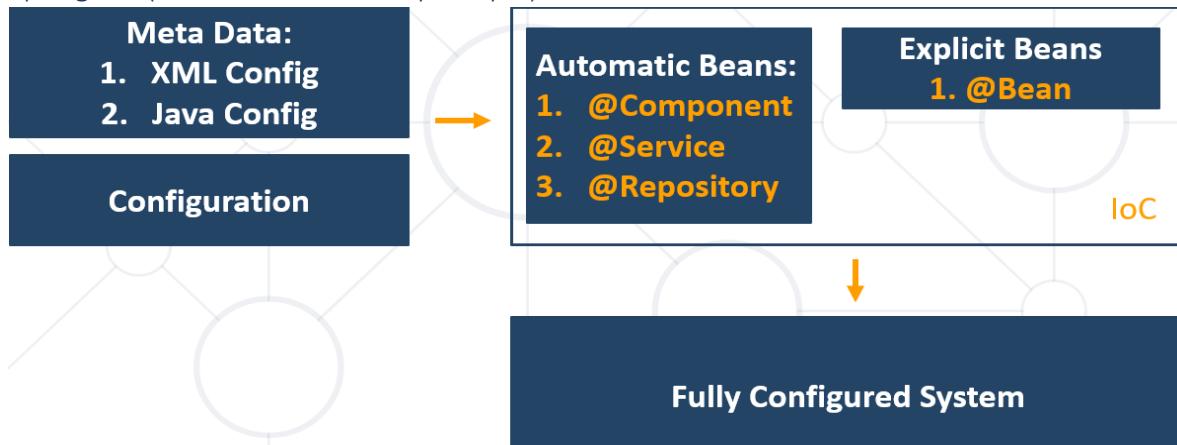
```
build.gradle
dependencies {
 compileOnly("org.springframework.boot:spring-boot-starter-actuator")
}
```

```

{
 "status": "UP",
 "diskSpace": {
 "status": "UP",
 "total": 160571584512,
 "free": 310485760
 },
 "db": [
 {
 "status": "UP",
 "database": "MySQL",
 "hello": 1
 }
]
}

```

Spring IoC (Inversion of Control principle)



Inversion of Control - the Dependency Injection option

- 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

Bean Declaration

- Object that is **instantiated**, **assembled**, and otherwise managed by a **Spring IoC** container  
T.e. не е необходимо да използваме new оператора.

```

public class Dog implements Animal {
 private String name;
 public Dog() {}
}

```

```
//GETTERS AND SETTERS
}
```

**А тук вече използваме new:**

```
@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 в рамките на дадена http сесия
- **Application** – за web-servlet приложения
- **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

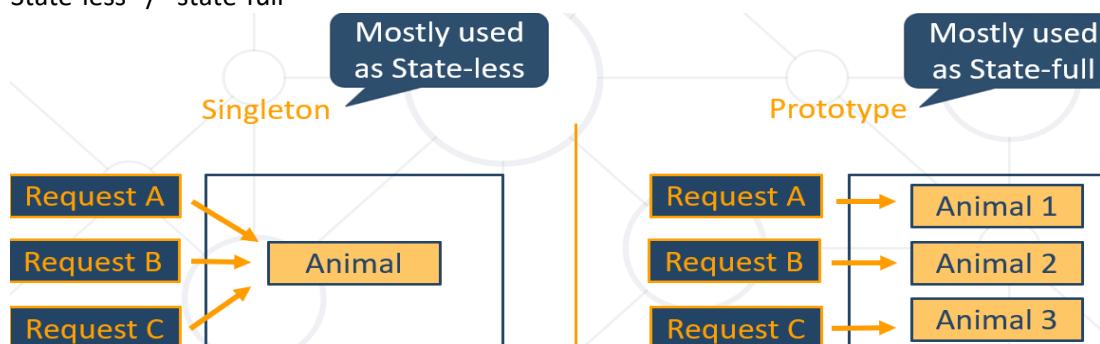
```

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

```

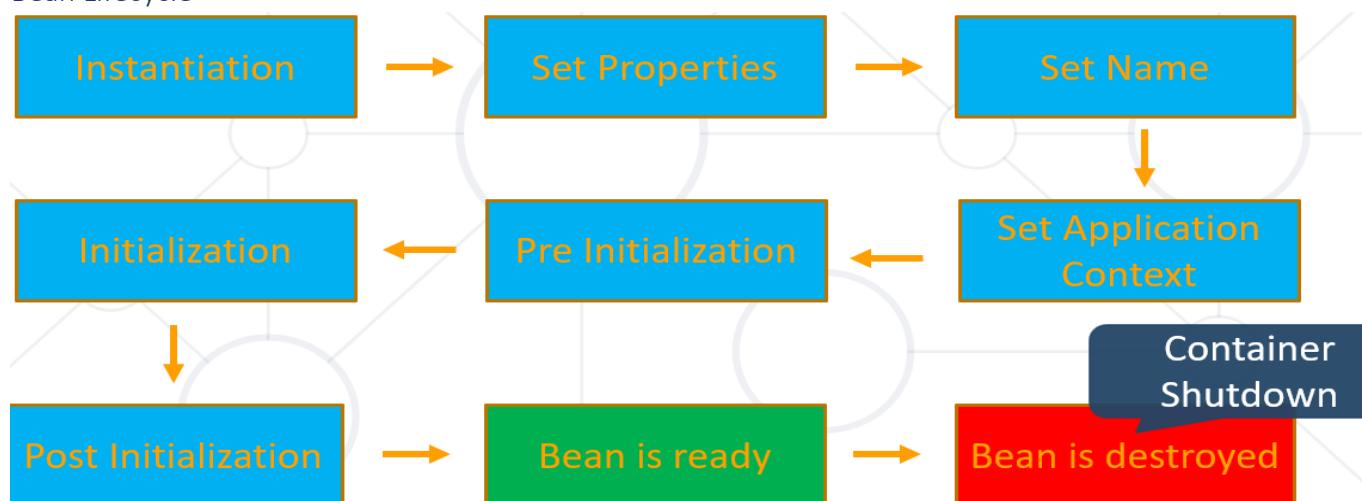
### Singleton vs Prototype scope

- The default one is **Singleton**. It is easy to change to **Prototype**
- State-less / state-full



### 5.2. Customizing the bean lifecycle

#### Bean Lifecycle



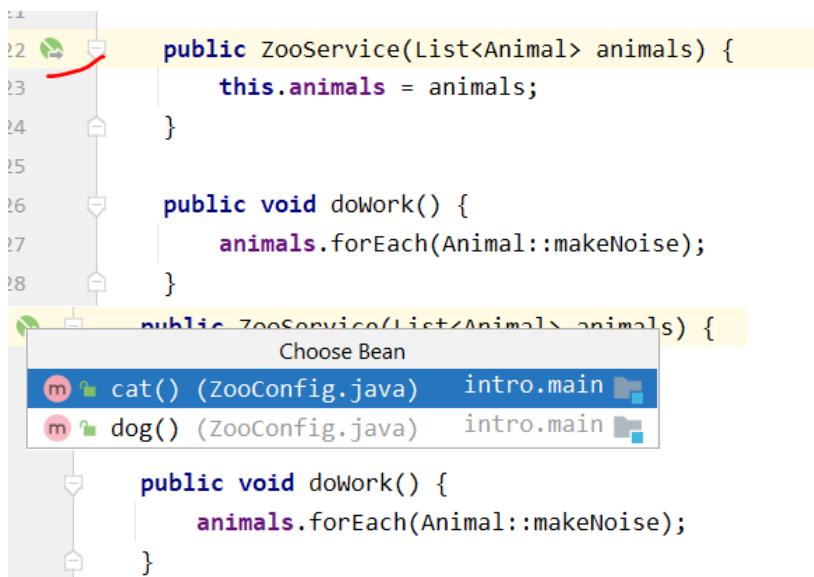
Обхождане на всеки от bean-овете

IntelliJ средата ни дава препратка към двата bean-a

```
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; //OPEN-CLOSE principle here
 public ZooService(List<Animal> animals) {
 this.animals = animals;
 }

 public void doWork() {
 animals.forEach(Animal::makeNoise);
 }
}
```

@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;
}
```

Using **@Primary**

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

**@Configuration**

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

**@Primary**

```
@Bean
public Animal dog() {
```

```

 return new Dog();
 }
}

@Service
public class ZooService {

 private final Animal animal;

 @Autowired
 public ZooService(Animal animal) {
 this.animal = animal;
 }

 public void doWork() {
 animal.makeNoise();
 }
}

```

### @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;
}

```

### Using **@Qualifier**

```

public class Dog implements Animal{
 private final boolean superDog;

 public Dog(){
 this(false);
 }

 public Dog(boolean superDog){
 this.superDog = superDog;
 }
}

```

```

@Override
public void makeNoise() {
 if (superDog) {
 System.out.println("Super Bark super bark");
 } else {
 System.out.println("Bark bark");
 }
}

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

 @Bean("normalDog")
 public Animal dog() {
 return new Dog();
 }

 @Bean("mySuperDog")
 public Animal superDog() {
 //todo: add superpower to this dog
 return new Dog(true);
 }
}

```

```

@Service
public class ZooService {
 private final Animal animal;

 @Autowired
 public ZooService(@Qualifier("mySuperDog") Animal animal) {
 this.animal = animal;
 }

 public void doWork() {
 animal.makeNoise();
 }
}

```

ИЛИ така

```

@Service
public class ZooService {
 private final Animal animal1;
 private final Animal animal2;

 @Autowired
 public ZooService(@Qualifier("mySuperDog") Animal animal1,
 @Qualifier("normalDog") Animal animal2) {
 this.animal1 = animal1;
 this.animal2 = animal2;
 }
}

```

```

 public void doWork() {
 animal1.makeNoise();
 animal2.makeNoise();
 }
}

```

## 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; Java EE) 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();
 }
}

```

### 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 – идва от javax/Jakarta EE) вместо **DisposableBean** (които са части Spring).

### 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()));
 }
}

```

### 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 this Dog bean is: " + name);
 }

 @Override
 public void destroy() throws Exception {
 System.out.println("Dog is about to die... Bye!");
 }
}

```

### 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();
 }
}

```

### 5.3. Application Properties

#### 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&createDatabaseIfNotExist=true&serverTimezone=UTC&useUnicode=true&characterEncoding=UTF-8"
 username: root
 password:
 servlet:
 multipart:
 max-file-size: 1MB
 max-request-size: 5MB
 mvc:
 hiddenmethod:
 filter:
 enabled: true
 sql: //execute SQL / run SQL from the resources folder - always
 init:
 mode: always never //executes the file with sql scripts!
 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

logging:
level:
 org.springframework: DEBUG #debugging SPRING while Logging more info
 org.hibernate.SQL: DEBUG
 org.hibernate.type.descriptor.sql.BasicBinder: 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

@Value annotation

**application.yaml**

```
demo:
env:
 firstName: Pesho
 lastName: Petrov
```

```

import jakarta.annotation.PostConstruct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ConfigDemoValueAnnotation {
 private final Logger LOGGER =
LoggerFactory.getLogger(ConfigDemoValueAnnotation.class);

 private final String firstName;
 private final String lastName;

 public ConfigDemoValueAnnotation(@Value("${demo.env.firstName}") String firstName,
 @Value("${demo.env.lastName}") String lastName) {
 this.firstName = firstName;
 this.lastName = lastName;
 }

 @PostConstruct
 void init() {
 LOGGER.info("First name: {}, Last name: {}", firstName, lastName);
 }
}

```

@ConfigurationProperties annotation

### **application.yaml**

```

demo:
 env:
 firstName: Pesho
 lastName: Petrov

```

---

```

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Configuration
@ConfigurationProperties("demo.env")
public class ConfigProps {
 private String firstName;
 private String lastName;

 public String getFirstName() {
 return firstName;
 }

 public String getLastName() {
 return lastName;
 }

 public void setFirstName(String firstName) {
 this.firstName = firstName;
 }

 public void setLastName(String lastName) {

```

```

 this.lastName = lastName;
 }
}

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.EnableConfigurationProperties;

@SpringBootApplication
@EnableConfigurationProperties
public class DemoApplication {

 public static void main(String[] args) {
 SpringApplication.run(DemoApplication.class, args);
 }

}

import jakarta.annotation.PostConstruct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Component
public class ConfigPropertiesDemo {
 private final Logger LOGGER = LoggerFactory.getLogger(ConfigPropertiesDemo.class);

 private final ConfigProps configProps;

 //constructor injection - in the new versions the @Autowired annotation is not required
 public ConfigPropertiesDemo(ConfigProps configProps) {
 this.configProps = configProps;
 }

 @PostConstruct
 void init() {
 LOGGER.info("First name: {}, Last name: {}", configProps.getFirstName(),
 configProps.getLastName());
 }
}

```

## 5.4. CommandLineRunner

When a task should be performed during the application startup.

След инициализация на нашето приложение, нашия CommandLineRunner се включва да изпълним някаква логика и да логнем някакви неща в конзолата.

Обикновено го закачаме за някой бийн този CommandLineRunner. А не че самото приложение се стартира чрез него.

```

@Component
public class DBInit implements CommandLineRunner {
 private final Logger LOGGER = LoggerFactory.getLogger(DBInit.class);

```

```

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

}

```

**Можем да го закачим и за класа, който инициализира цялото ни приложение:**

```

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.EnableConfigurationProperties;

@SpringBootApplication
@EnableConfigurationProperties
public class DemoApplication implements CommandLineRunner {

 public static void main(String[] args) {
 SpringApplication.run(DemoApplication.class, args);
 }

 @Override
 public void run(String... args) throws Exception {
 // some logic here
 }
}

```

## 5.5. ApplicationRunner вместо CommandLineRunner interface

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class AppRunner implements ApplicationRunner {
 private final SeedService seedService;
 private final BookRepository bookRepository;
 private final AuthorRepository authorRepository;

 @Autowired
 public AppRunner(SeedService seedService, BookRepository bookRepository, AuthorRepository
authorRepository) {
 this.seedService = seedService;
 this.bookRepository = bookRepository;
 this.authorRepository = authorRepository;
 }

 @Override
 public void run(ApplicationArguments args) throws Exception {
 System.out.println("Starting");
 // this.seedService.seedAuthors();
 // this.seedService.seedCategories();
 // this.seedService.seedAll();

 // this._01_booksAfter2000();
 }
}

```

```
// this._02_allAuthorsWithBookBefore1990();
// this._03_allAuthorsOrderedByBookCount();
this._04_GetAllBooksByAuthorname();
}
```

## 5.6. @SpringBootApplication annotation

Дава ни @ComponentScan

**@SpringBootApplication** анотацията сканира всичко с **@ComponentScan** от текущата главна директория и всички поддиректории и вкарани/импортирани външни депенденсита/библиотеки за текущия ни проект!

Дава ни @EnableAutoConfiguration

Настройва целия ни application. Има много на брой технологии/библиотеки. Една от тях е GSON.

Само ако GSON е наличен в клас пътя

```
0 @AutoConfiguration
1 @ConditionalOnClass(Gson.class)
2 @EnableConfigurationProperties(Gso
3 public class GsonAutoConfiguration
```

Наличен е когато го добавим в application.properties (application.yaml) файла:  
implementation 'com.google.code.gson:gson'

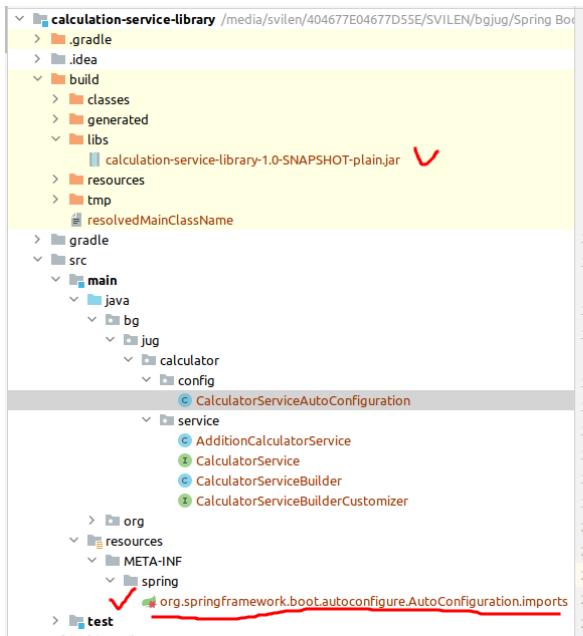
Започват да се експозуват бийнове от дадената библиотека автоматично в application context-a. Т.е. когато го inject-нем бийна от библиотеката директно в някой наш бийн.

**Освен ако аз не съм си вкаран ръчно/изрично такъв бийн в application context-а чрез никакъв customizing builder!!! Тогава Спринг ще отстъпи назад! Opinionated, but get out of the way quickly!**

## 5.7. Opinionated, but get out of the way quickly - Demo @AutoConfiguration - пример за отдръпване от Спринг:

Пакетиране до библиотека calculation-service-library

Тръгване от обикновен проект



```

1 package bg.jug.calculator.config;
2
3 import bg.jug.calculator.service.CalculatorService;
4 import bg.jug.calculator.service.CalculatorServiceBuilder;
5 import bg.jug.calculator.service.CalculatorServiceBuilderCustomizer;
6 import org.springframework.boot.autoconfigure.AutoConfiguration;
7 import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
8 import org.springframework.context.annotation.Bean;
9
10 import java.util.List;
11
12 @AutoConfiguration
13 public class CalculatorServiceAutoConfiguration {
14 @Bean
15 @ConditionalOnMissingBean //този бийн ще бъде включен в spring контекста на при.
16 public CalculatorServiceBuilder calculatorServiceBuilder(List<CalculatorService>
17 // The default builder
18 CalculatorServiceBuilder builder = new CalculatorServiceBuilder()
19 .setStartMessage("Starting calculation with %d and %d")
20 .setFinishMessage("The result of the calculation is %d");
21
22 customizerList.forEach(c -> c.customize(builder));
23
24 return builder;

```

## build.gradle

```

plugins {
 id 'java'
 id 'org.springframework.boot' version '3.1.3'
 id 'io.spring.dependency-management' version '1.1.3' //грижи се за съвместимост на
 версиите
}

group 'bg.jug.calculator'
version '1.0-SNAPSHOT'

repositories {
 mavenCentral()
}

dependencies {
 implementation 'org.springframework.boot:spring-boot-actuator-autoconfigure'

 testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'
 testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'
}

jar {
 enabled(true)
}

bootJar {
 enabled = false
}

```

```
org.springframework.boot.autoconfigure.AutoConfiguration.imports ×
1 bg.jug.calculator.config.CalculatorServiceAutoConfiguration
```

Тази gradle команда в комбинация с конфигурационния файл **build.gradle генерира .jar** файл:  
./gradlew clean build

```

package bg.jug.calculator.config;

import bg.jug.calculator.service.CalculatorService;
import bg.jug.calculator.service.CalculatorServiceBuilder;
import bg.jug.calculator.service.CalculatorServiceBuilderCustomizer;
import org.springframework.boot.autoconfigure.AutoConfiguration;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;

import java.util.List;

@AutoConfiguration
public class CalculatorServiceAutoConfiguration {
 @Bean
 @ConditionalOnMissingBean //този бийн ще бъде включен в spring контекста на приложението, което
 //ползва тази библиотека calculation-service-library - само ако в нашето приложение не сме
 //сложили в спринг контекста CalculatorServiceBuilder
 public CalculatorServiceBuilder
calculatorServiceBuilder(List<CalculatorServiceBuilderCustomizer> customizerList) {
 // The default builder
 CalculatorServiceBuilder builder = new CalculatorServiceBuilder()
 .setStartMessage("Starting calculation with %d and %d")
 .setFinishMessage("The result of the calculation is %d");

 customizerList.forEach(c -> c.customize(builder));

 return builder;
 }

 @Bean
 @ConditionalOnMissingBean //този бийн ще бъде включен в spring контекста на приложението,
 //което ще ползва тази библиотека calculation-service-library - само ако в нашето приложение не сме
 //сложили в спринг контекста CalculatorService
 public CalculatorService calculatorService(CalculatorServiceBuilder builder) {
 return builder.build();
 }
}

public interface CalculatorService {
 int apply(int op1, int op2);
}

public class AdditionCalculatorService implements CalculatorService {

 private final String startMessage;
 private final String finishMessage;
```

```

public AdditionCalculatorService(String startMessage, String finishMessage) {

 this.startMessage = startMessage;
 this.finishMessage = finishMessage;
}

@Override
public int apply(int op1, int op2) {
 System.out.printf(startMessage, op1, op2);
 System.out.println();

 int result = op1 + op2;

 System.out.printf(finishMessage, result);
 System.out.println();

 return result;
}
}

public interface CalculatorServiceBuilderCustomizer {
 void customize(CalculatorServiceBuilder builder);
}

```

## Използване на библиотеката calculation-service-library

**build.gradle**

```

dependencies {
 implementation files("/media/svilen/404677E04677D55E/SVILEN/bgjug/Spring Boot/calculation-
service-library/build/libs/calculation-service-library-1.0-SNAPSHOT-plain.jar")
}

```

-----

The screenshot shows the 'External Libraries' section of the Android Studio project navigation bar. A yellow highlight covers the 'calculation-service-library-1.0-SNAPSHOT-plain.jar' entry. Inside, it shows the package structure: 'bg' > 'jug' > 'calculator' > 'config' (containing 'CalculatorServiceAutoConfiguration'), 'service' (containing 'AdditionCalculatorService', 'CalculatorService', 'CalculatorServiceBuilder', and 'CalculatorServiceBuilderCustomizer'), and 'META-INF'. Below this, there's a 'spring' folder containing 'org.springframework.boot.autoconfigure.AutoConfiguration.impr...' and a 'MANIFEST.MF' file. To the right of the library tree, the build.gradle file is displayed with the cursor on the line 'implementation files("/media/svilen/404677E04677D55E/SVILEN/bgjug/Spring Boot/calculation-service-library/build/libs/calculation-service-library-1.0-SNAPSHOT-plain.jar")'.

**Това което е конфигурирано в библиотеката *calculation-service-library* се изпълнява - инджектване на бийна от *calculation-service-library* библиотеката става автоматично.**

```

import bg.jug.calculator.service.CalculatorService; //from calculation-service-library
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

```

```

@Component
public class CalculatorDemo implements CommandLineRunner {
 private CalculatorService calculatorService;

 // Constructor injection
 public CalculatorDemo(CalculatorService calculatorService) {
 this.calculatorService = calculatorService;
 }

 @Override
 public void run(String... args) throws Exception {
 System.out.println(calculatorService.apply(3, 4));
 }
}

```

Starting calculation with 3 and 4

The result of the calculation is 7

7

---

**Бийна се инжектва тук не от `calculation-service-library` библиотеката, а от `CalculatorConfig.calculatorServiceBuilderCustomizer` бийн от текущото приложение. Т.е. отдръпване от Спринг.**

```

package com.example.demo.config;

import bg.jug.calculator.service.CalculatorServiceBuilderCustomizer; //from calculation-service-
library
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class CalculatorConfig {

 @Bean
 public CalculatorServiceBuilderCustomizer calculatorServiceBuilderCustomizer() {
 return b -> {
 b.setStartMessage("Customized start message: %d and %d");
 b.setFinishMessage("Customized return message: %d");
 };
 }
}

```

Смяна на съобщението

Customized start message: 3 and 4

Customized return message: 7

7

---

```
package com.example.demo.config;
```

```

import bg.jug.calculator.service.CalculatorService; //from calculation-service-library
import bg.jug.calculator.service.CalculatorServiceBuilderCustomizer; //from calculation-service-
library

```

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class CalculatorConfig {

 @Bean
 public CalculatorServiceBuilderCustomizer calculatorServiceBuilderCustomizer() {
 return b -> {
 b.setStartMessage("Customized start message: %d and %d");
 b.setFinishMessage("Customized return message: %d");
 };
 }

 @Bean
 public CalculatorService customCalculatorService() {
 return (op1, op2) -> op1 * op2;

 return new CalculatorService() {
 @Override
 public int apply(int op1, int op2) {
 System.out.printf("Changed from addition to multiply. Operands are: %d and %d", op1,
 op2);
 System.out.println();

 return op1 * op2;
 }
 };
 }
}

```

Changed from addition to mutiply. Operands are: 3 and 4  
12

## 5.8. Profiles

### Info

- Segregation - can segregate your configuration and make it available on certain environments
  - @Profile annotation - can be used on:
    - @Component (@Service, @Controller, @RestController),
    - @Configuration
    - and @ConfigurationProperties
      - Warning - combinations of too much profiles makes life harder

```

@Component
public class GreetingDemo implements CommandLineRunner {

 private List<GreetingService> greeters;

 public GreetingDemo(List<GreetingService> greeters) {
 this.greeters = greeters;
 }

 @Override
 public void run(String... args) throws Exception {
 System.out.println("-----");

```

```

 greeters.forEach(gs -> {
 String greet = gs.greet();
 System.out.println(greet);
 });

 System.out.println("-----");
 }
}

import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Service;

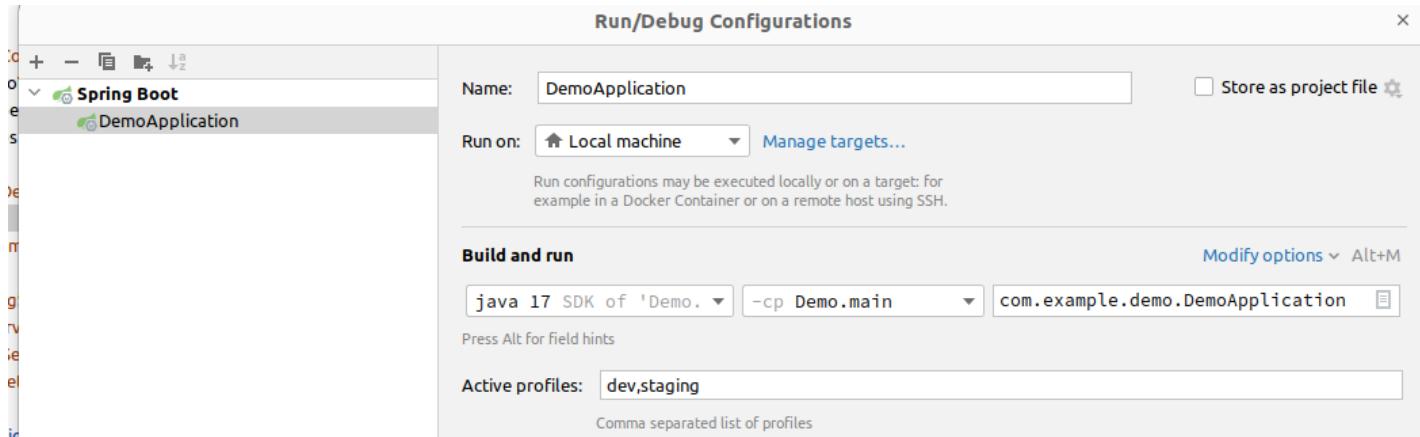
@Service
@Profile("dev")
public class DevGreetingService implements GreetingService {
 @Override
 public String greet() {
 return "Hello from DEV environment";
 }
}

@Service
@Profile("prod")
public class ProdGreetService implements GreetingService{
 @Override
 public String greet() {
 return "Hello from PROD";
 }
}

@Service
@Profile("staging")
public class StagingGreetService implements GreetingService{
 @Override
 public String greet() {
 return "Hello from STAGING";
 }
}

```

Локално:

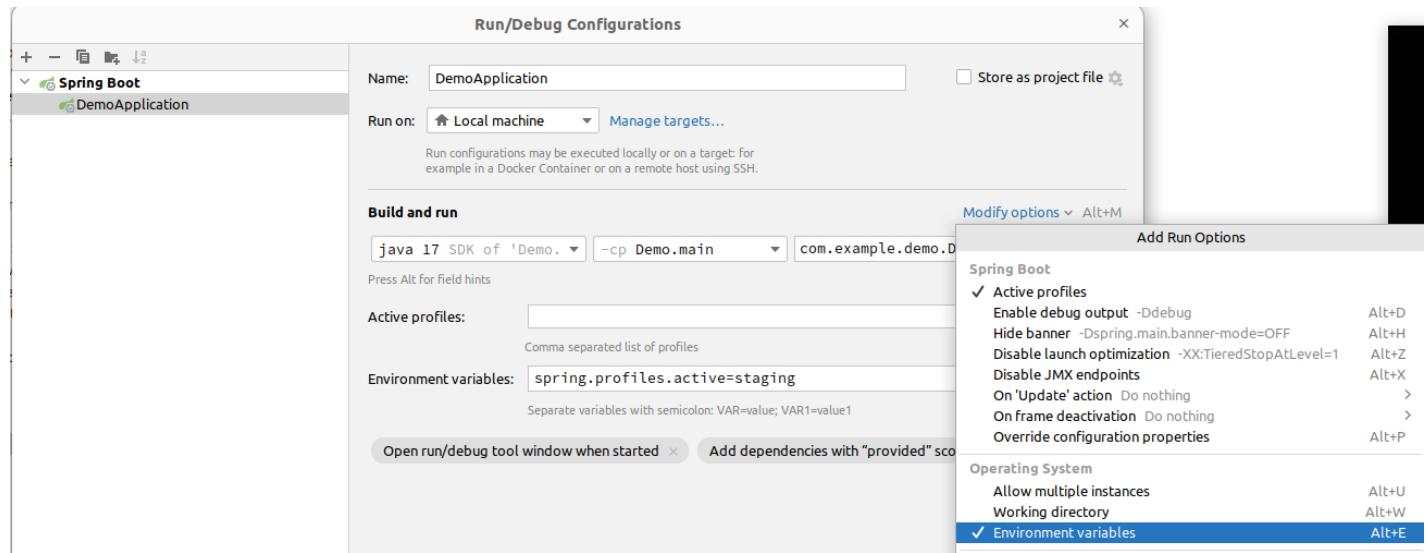


В application.yaml файла:

```
spring:
 profiles:
 active: dev, prod
```

На облак като е:

След като се деплоине на някакъв cloud, то можем оттам да го контролираме

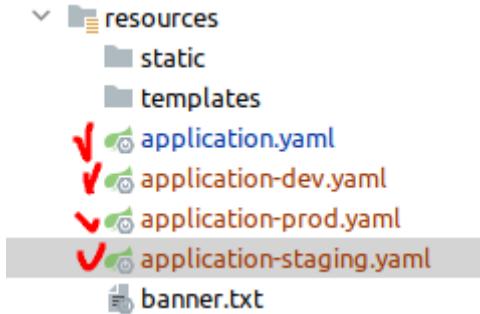


С custom environment variables на облака и в application.yaml файла:

```
spring:
 profiles:
 active: ${MY_VAR:dev} #default is dev
```



С отделни application.properties файлове



### application-staging.yaml

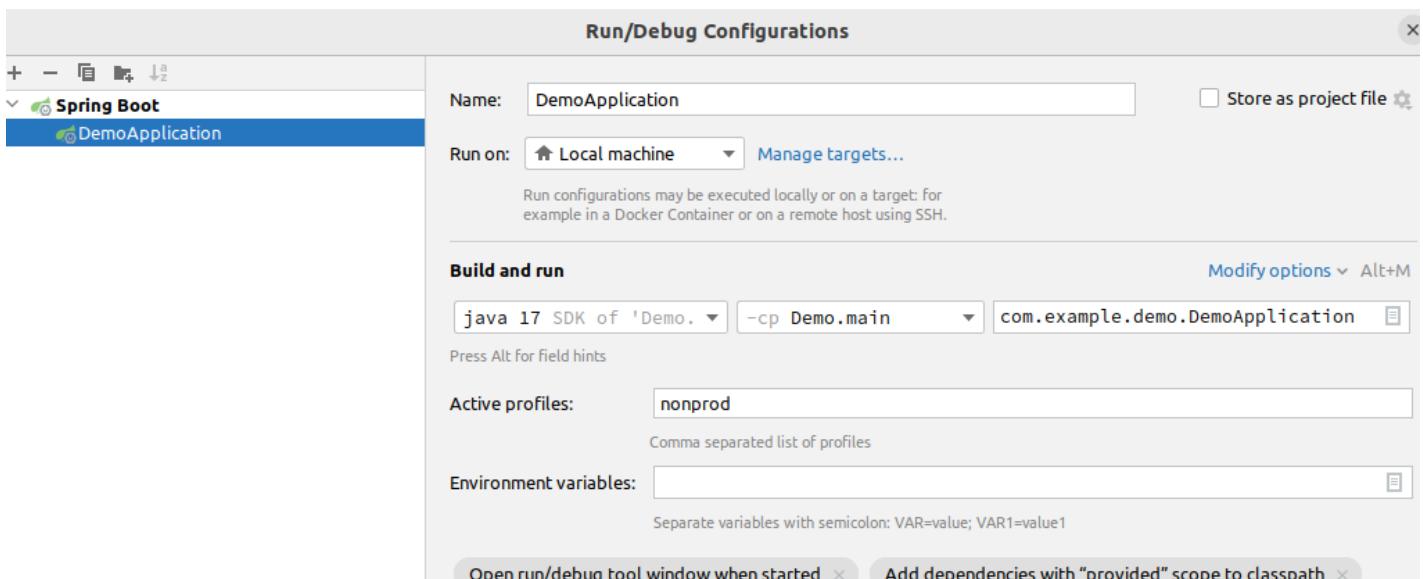
```
demo:
 env:
 firstName: PeshoSTAGING
 lastName: PetrovSTAGING
```



**ДА НЕ РАЗЧИТАМЕ ТУК(с отделни application.yaml файлове) за профилите кое е с приоритет ако има 2 или 3 активни !!!**

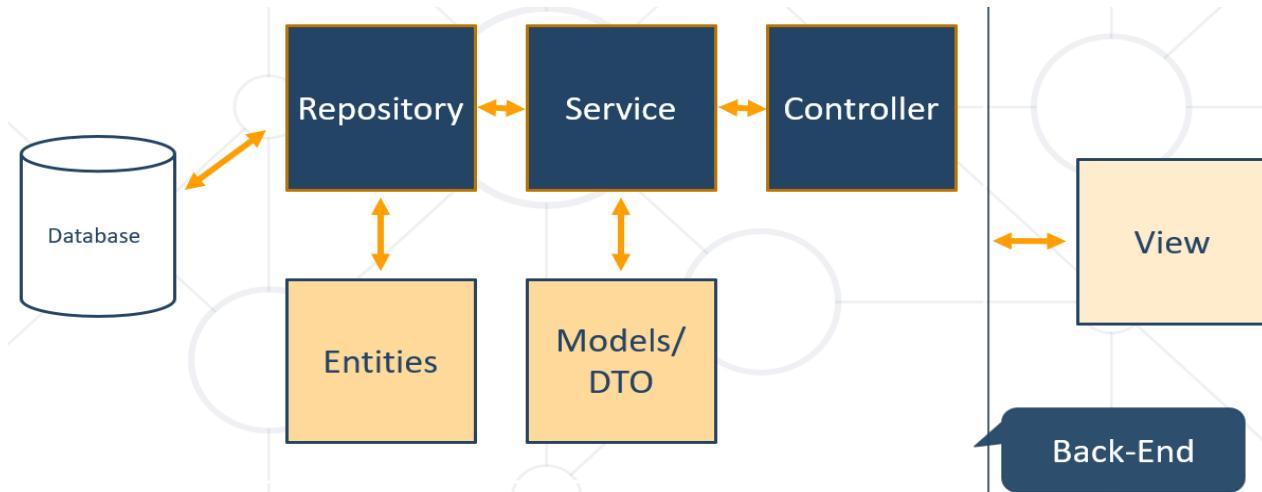
С профилна група

```
spring:
 profiles:
 group:
 nonprod[0]: "dev"
 nonprod[1]: "staging"
```



## 5.9. 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> { }
```

Indicates that an annotated class is a "Repository", originally defined by **Domain-Driven Design** (Evans, 2003) as "a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects".

Teams implementing **traditional Java EE patterns such as "Data Access Object"** may also apply this stereotype to DAO classes, though care should be taken to understand the distinction between Data Access Object and DDD-style repositories before doing so. This annotation is a general-purpose stereotype and individual teams may narrow their semantics and use as appropriate.

A class thus annotated is eligible for Spring `DataAccessException` translation when used in conjunction with a `PersistenceExceptionTranslationPostProcessor`. The annotated class is also clarified as to its role in the overall application architecture for the purpose of tooling, aspects, etc.

As of Spring 2.5, this annotation also serves as a specialization of `@Component`, allowing for implementation classes to be autodetected through classpath scanning.

## Services

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

`@Service` - This annotation serves as a specialization of `@Component`, allowing for implementation classes to be autodetected through classpath scanning.

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

 @Autowired //не е задължително, даже е излишно
 public CatServiceImpl(CatRepository catRepository) {
 this.catRepository = catRepository;
 }

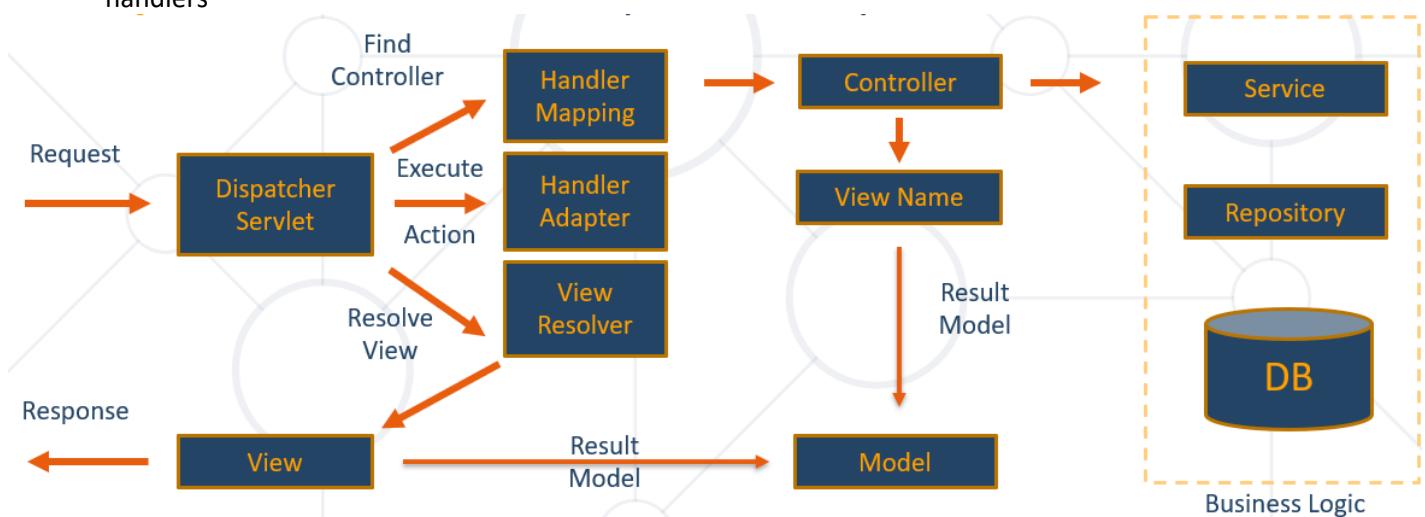
 @Override
 public void buyCat(CatModel catModel) { //TODO Implement the method }
}
```

## 6. MVC Spring Introduction

### 6.1. What is Spring MVC?

What is Spring MVC?

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

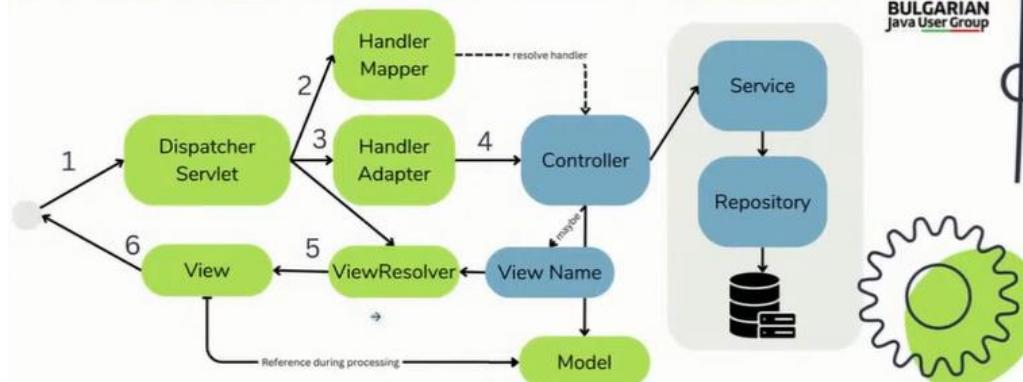


Най-често handler е контролер.

В нашите приложения приемаме, че handler = controller

По ред на номерата е потока на нещата как се случват.

## SPRING MVC ARCHITECTURE



Ние като дивелъпъри творим в **синята част**.

**Зелената част** е вградена от самия Spring Boot framework/Servlet API –

**DispatcherServlet.doDispatch(HttpServletRequest request, HttpServletResponse response)**

.....

**mappedHandler = getHandler(processedRequest);**

Project structure:

- ExRateEntity
- repository
- service
- impl
- web
- ForexApplication
- resources
- static.css
- templates
- index.html
- application.yaml

Code in HomeController.java:

```
@Controller
public class HomeController {

 @GetMapping("/")
 public String home(Model model) {
 model.addAttribute(attributeName: "today", LocalDate.now());
 return "index";
 }
}
```

Debugger view:

- "http-nio-8080-exec-5"@11,274 in group "main": RUNNING
- doDispatch:1068, DispatcherServlet (org.springframework.web.servlet)
- doService:979, DispatcherServlet (org.springframework.web.servlet)
- processRequest:1014, FrameworkServlet (org.springframework.web.servlet)
- doGet:903, FrameworkServlet (org.springframework.web.servlet)
- service:564, HttpServlet (jakarta.servlet.http)
- service:885, FrameworkServlet (org.springframework.web.servlet)
- service:858, HttpServlet (jakarta.servlet.http)
- internalDoFilter:205, ApplicationFilterChain (org.apache.catalina.core)
- doFilter:149, ApplicationFilterChain (org.apache.catalina.core)
- doFilter:51, WsFilter (org.apache.tomcat.websocket.server)
- internalDoFilter:174, ApplicationFilterChain (org.apache.catalina.core)
- doFilter:149, ApplicationFilterChain (org.apache.catalina.core)
- doFilterInternal:100, RequestContextFilter (org.springframework.web.filter)
- doFilter:116, OncePerRequestFilter (org.springframework.web.filter)
- internalOnFilter:174, ApplicationFilterChain (org.apache.catalina.core)

Watch list (highlighted in red):

- mappedHandler = (HandlerExecutionChain@16611) "HandlerExecutionChain with [bg.jug.forex.web.HomeController#home(Model)]"
- handler = (HandlerMethod@16614) "bg.jug.forex.web.HomeController#home(Model)"
- interceptorList = (ArrayList@16615) size = 3
- interceptorIndex = -1
- multipartRequestParsed = false
- asyncManager = (WebAsyncManager@16580)
- mv = null
- dispatchException = null

**HandlerAdapter.handle(HttpServletRequest request, HttpServletResponse response, Object handler)**

```

mv = {ModelAndView@16635} " ModelAndView [view="index"]
 ↪ view = "index"
 ↪ model = {ModelMap@16640} size = 1
 > ⏎ "today" -> {LocalDate@16645} "2024-01-11"

```

## DispatcherServlet.render(ModelAndView mv, HttpServletRequest request, HttpServletResponse response)

```

protected void render(ModelAndView mv, HttpServletRequest request, HttpServletResponse response)
 // Determine locale for request and apply it to the response.
 Locale locale = locale: "en_US"
 (this.localeResolver != null ? this.localeResolver.resolveLocale(request) : request.
 response.setLocale(locale); response: ResponseFacade@16560 locale: "en_US"

 View view;
 String viewName = mv.getViewName(); mv: " ModelAndView [view="index"; model={today=2024-01-11}"]
 if (viewName != null) {
 // We need to resolve the view name.

```

Stack Trace:

```

render:1404, DispatcherServlet (org.springframework.web.servlet)
processDispatchResult:1167, DispatcherServlet (org.springframework.web.servlet)
doDispatch:1106, DispatcherServlet (org.springframework.web.servlet)
doService:979, DispatcherServlet (org.springframework.web.servlet)
processRequest:1014, FrameworkServlet (org.springframework.web.servlet)

```

Самото view тук се преобразува до тип Thymeleaf view

```

response.setLocale(locale); response: ResponseFacade@16560
 View view; view: ThymeleafView@16674
 String viewName = mv.getViewName(); viewName: "index"
 if (viewName != null) {
 // We need to resolve the view name.
 view = resolveViewName(viewName, mv.getModelInternal(), locale, request); mv: " ModelAndView [view="
 if (view == null)
 throw new ServletException(" in ser
 }

```

JavaDoc for resolveViewName:

```

Resolve the given view name into a View object (to be rendered).
The default implementation asks all ViewResolvers of this dispatcher.
Can be overridden for custom resolution strategies, potentially based on
specific model attributes or request parameters.

Params: viewName - the name of the view to resolve
model - the model to be passed to the view
locale - the current locale
request - current HTTP servlet request

Returns: the View object, or null if none found

Throws: Exception - if the view cannot be resolved (typically in case of
problems creating an actual View object)

See Also: ViewResolver.resolveViewName

```

What is servlet? – A front-end controller

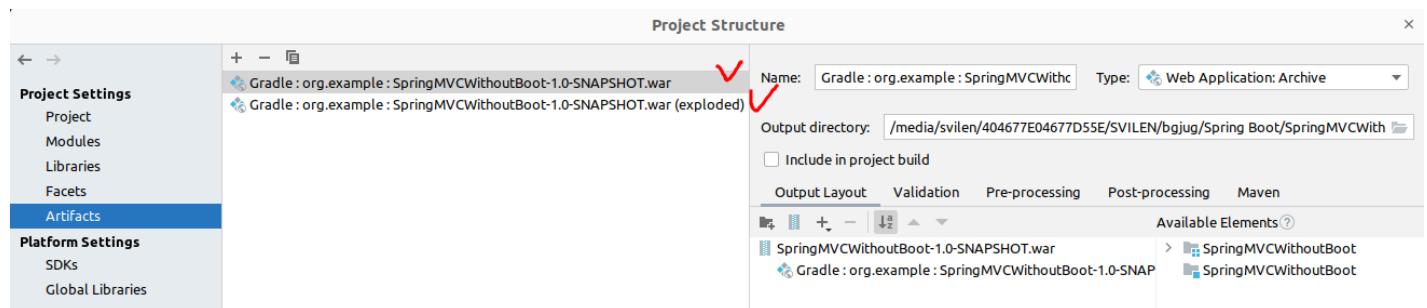
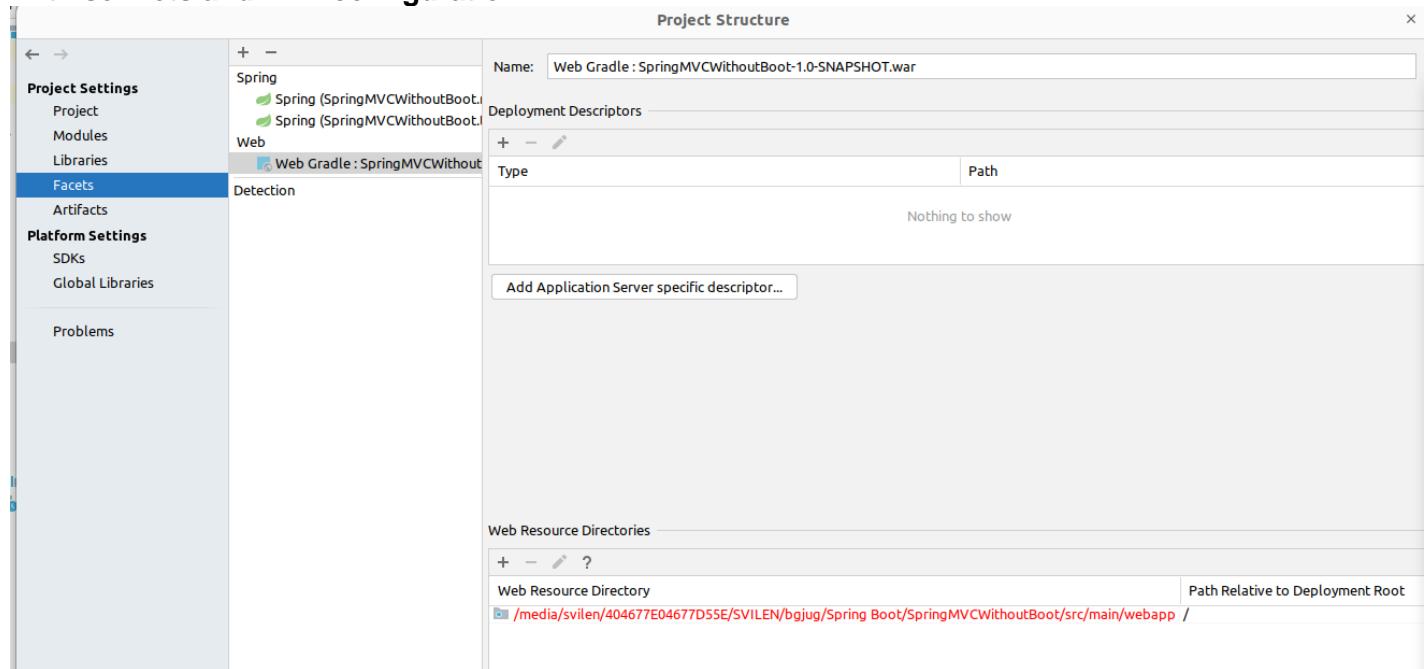
A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.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

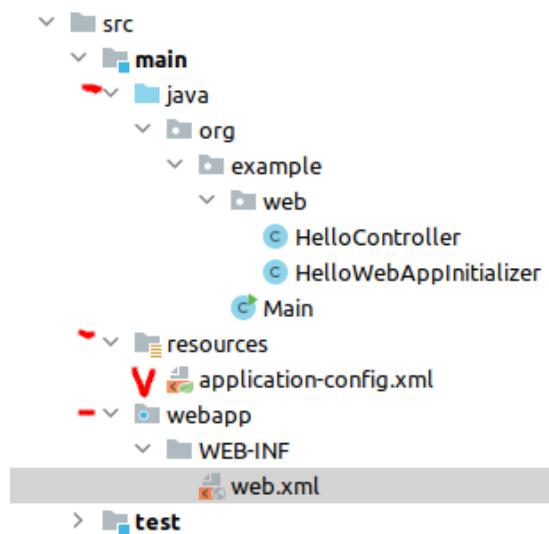
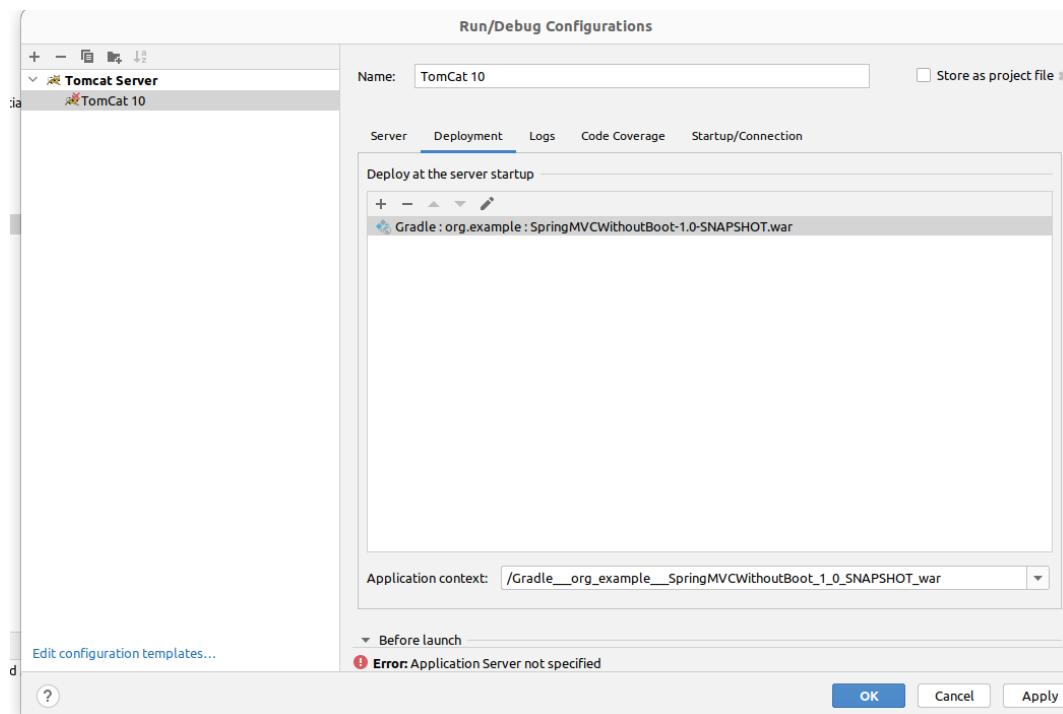
This chapter focuses on writing servlets that generate responses to HTTP requests.

## Demo Spring Web MVC without Spring Boot

### With servlets and XML configuration



Трябва да сме си инсталирали и самия Tomcat local server или някой друг сървър (WildFly/JBoss) за сървлети.



```

build.gradle
plugins {
 id 'java'
 id 'war' //да може да генерира war файл
}

group 'org.example'
version '1.0-SNAPSHOT'

repositories {
 mavenCentral()
}

dependencies {
 implementation 'org.springframework:spring-webmvc:6.0.9'
}

```

```

 implementation 'jakarta.servlet:jakarta.servlet-api:6.0.0'
 }

import org.springframework.beans.factory.InitializingBean;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController implements InitializingBean {

 @RequestMapping("/welcome")
 @ResponseBody
 public String welcome() {
 return "Welcome!";
 }

 @Override
 public void afterPropertiesSet() throws Exception {
 System.out.println("HelloController is initialized");
 }
}

import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.XmlWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class HelloWebAppInitializer implements WebApplicationInitializer {

 // Method
 @Override
 public void onStartup(ServletContext servletContext) throws ServletException {
 XmlWebApplicationContext webApplicationContext = new XmlWebApplicationContext();
 webApplicationContext.setConfigLocation("classpath:application-config.xml");

 // Creating a dispatcher servlet object
 DispatcherServlet dispatcherServlet = new
 DispatcherServlet(webApplicationContext);

 // Registering a dispatcher Servlet with Servlet Context
 ServletRegistration.Dynamic myCustomDispatcherServlet =
 servletContext.addServlet("myDispatcherServlet", dispatcherServlet);

 // Setting load on startup
 myCustomDispatcherServlet.setLoadOnStartup(1);

 // Adding mapping url (Custom URL)
 myCustomDispatcherServlet.addMapping("/example/*"); //да бъде достъпен на
/example

 System.out.println("Dispatcher Servlet is registered successfully.");
 }
}

```

```

application-config.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:context="http://www.springframework.org/schema/context"
 xsi:schemaLocation="
 http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="org.example.web"></context:component-scan>

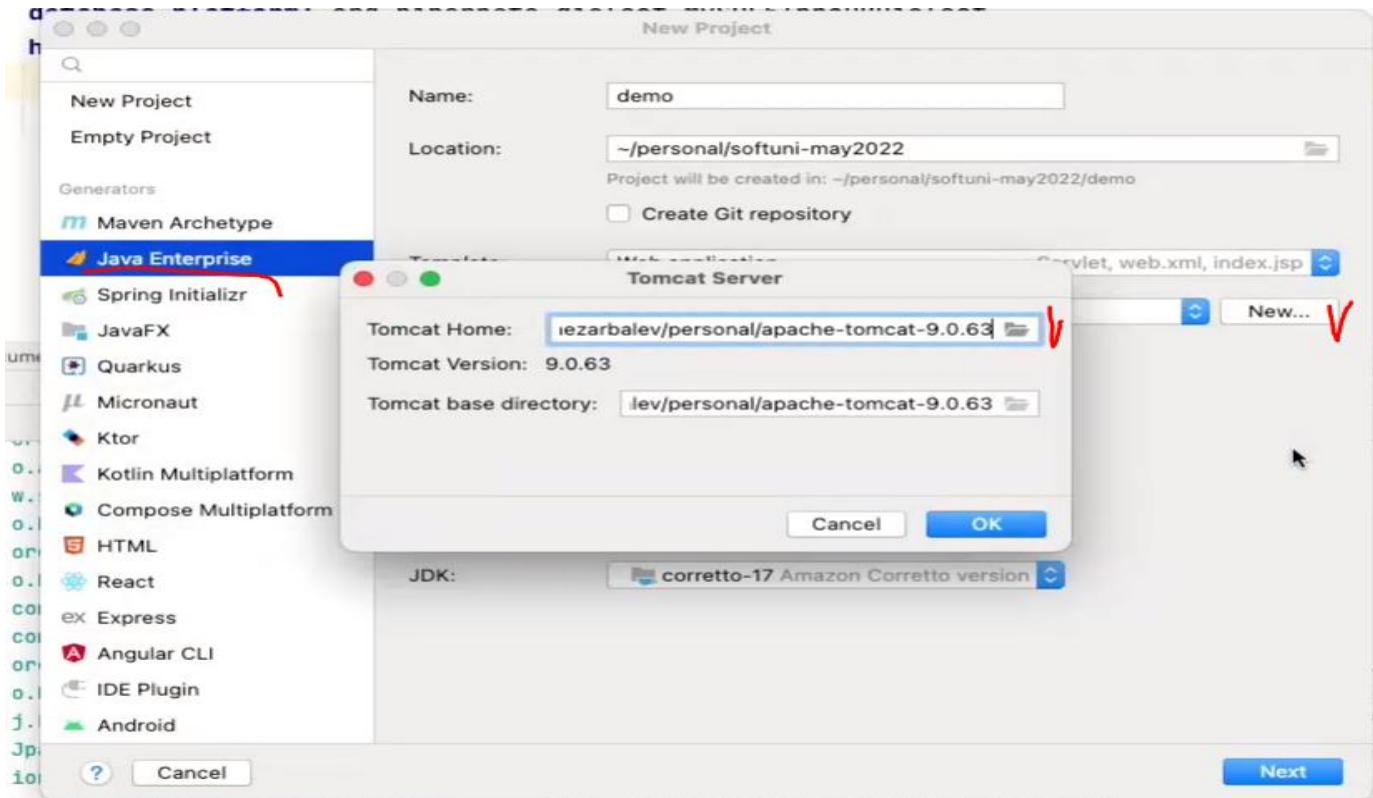
</beans>

web.xml
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
 <display-name>Sample app</display-name>
</web-app>
```

## 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!";
 }

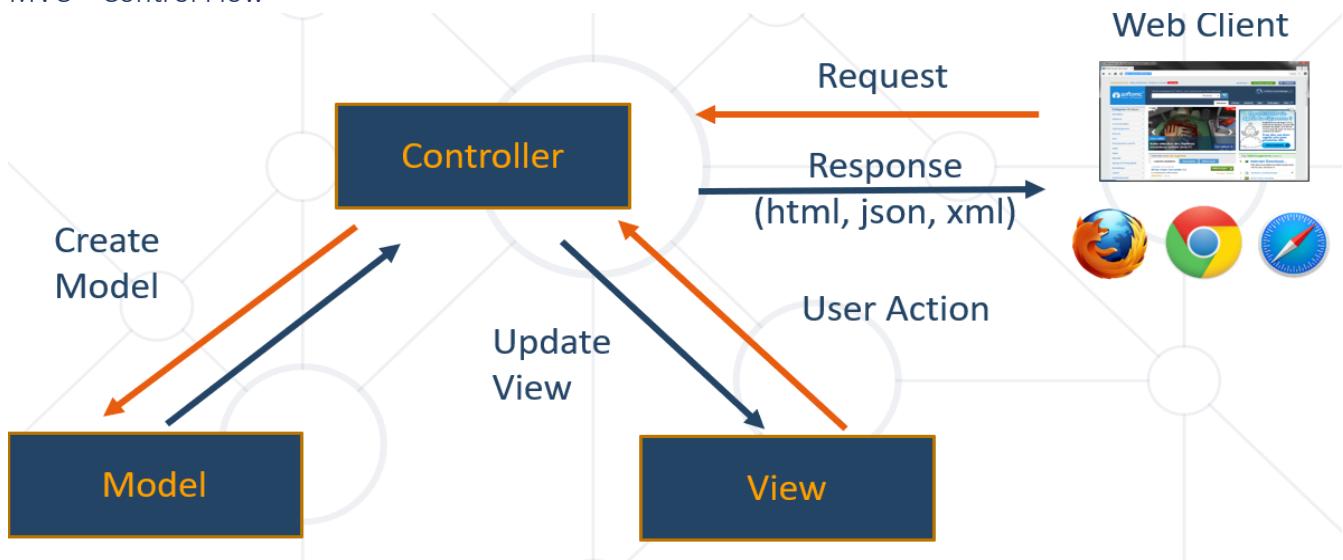
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException {
 response.setContentType("text/html");
 response.setStatus(HttpStatus.NotFound)

 // Hello
 PrintWriter out = response.getWriter();
 out.println("<html><body>");
 out.println("<h1>" + message + "</h1>");
 out.println("</body></html>");
 }

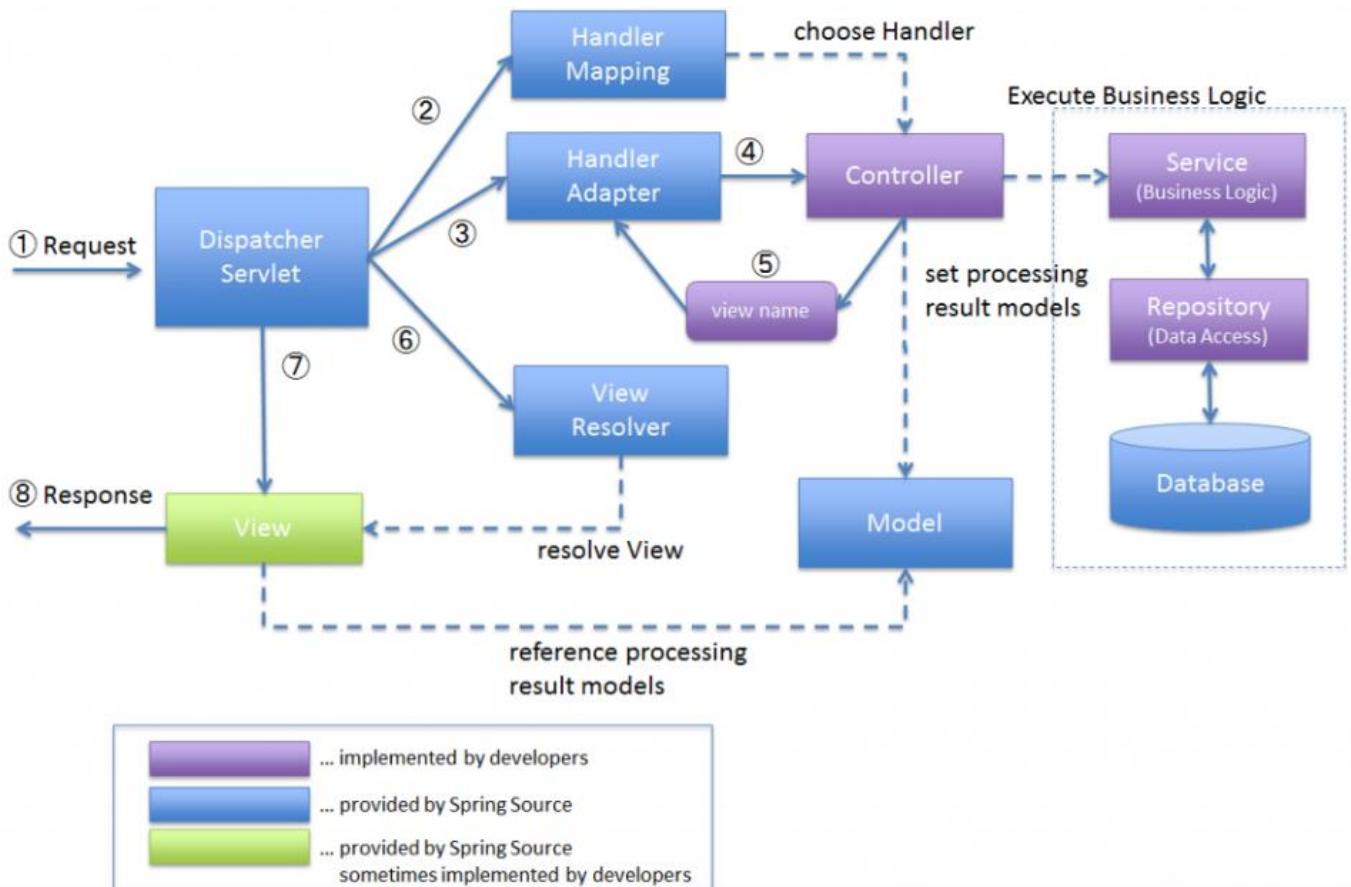
 public void destroy() {
}
}

```

## MVC – Control Flow



<https://www.upgrad.com/blog/spring-mvc-flow-diagram/#:~:text=Spring%20MVC%20is%20a%20Java,Dependency%2C%20Inversion%20of%20Control.>



## 6.2. 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-a
 }
}
```

- Controllers can contain multiple actions on different routes

**@Controller** - Indicates that an annotated class is a "Controller" (e.g. a web controller). **This annotation serves as a specialization of @Component**, allowing for implementation classes to be autodetected through classpath scanning.

### 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.

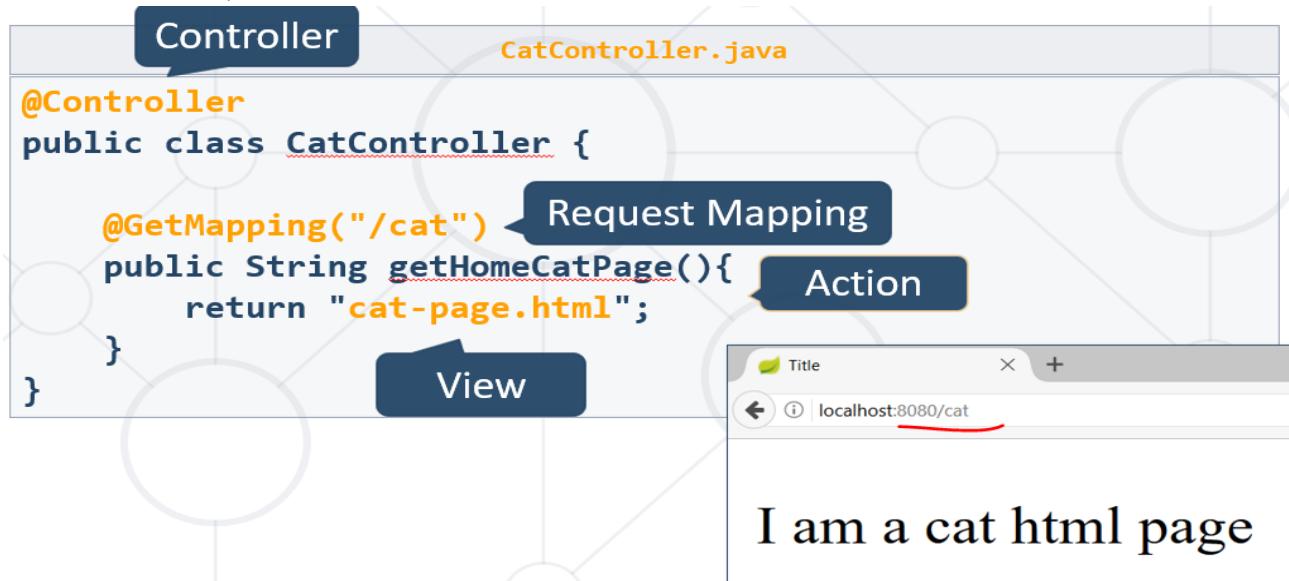
```
@RequestMapping(value="/home", method=RequestMethod.GET)
public String home() {
 return "home-view";
}
```

### 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"; //како работим с thymeleaf, то .html не е необходимо
}
```

## Actions – Get Requests



## 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";  
    }  
}
```

A screenshot of a web application showing a form titled "Add cat". The URL in the address bar is `localhost:8080/cat`. The form has two input fields: "Cat Name" containing "Tom" and "Cat Age" containing "20". Below the inputs is a button labeled "Add Cat".

Add cat

localhost:8080/cat

Cat Name Tom

Cat Age 20

Add Cat

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
    }
}
```

```
Cat Name: Tom, Cat Age: 20
```

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}" or howl
                    class="form-select mt-5" aria-label="Default select example">
                    <option value="" selected>Select one of the ships that are owned by the current
                    </option>
                    <option
                        th:each="ship : ${ownships}"
                        th:value="${ship.id}"
                        th:text="|${ship.name} -- ${ship.health} -- ${ship.power}|"
                        th:selected="*{attackerShipId} == ${ship.id}">
                </select>
            </div>
        </div>
    </div>

```

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-a
}

```

- 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

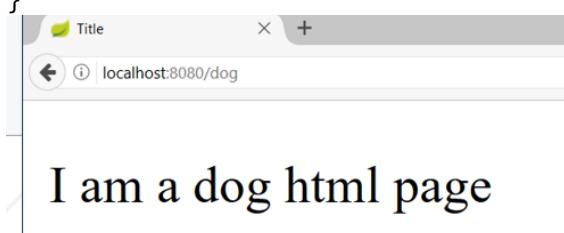
Models and Views

```

@Controller
public class DogController {

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

```



6.3. 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";
}

```

6.4. Request parameters

Info

Query string, 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`.

Query case example

- Getting a parameter from the query string

Пример 1:

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) {
    ...
}

```

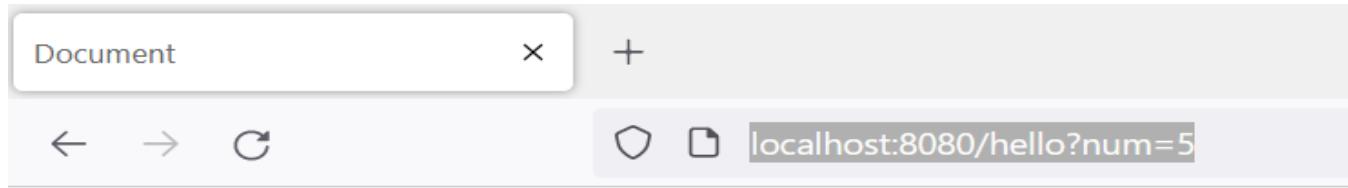
Пример 2:

```

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

```

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



Hello World

The number you passed to me is
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) {
    ...
}
```

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 = "Anonymous") 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) {
    ...
}
```

6.5. Request query parameters as a DTO

Подаваме на ендпойнта:

localhost:8080/add?currency1=1%20EUR¤cy2=2%20EUR



За това дипендиънси за Spring MVC се отнася реално:

```
implementation 'org.springframework.boot:spring-boot-starter-web'
```

```
-----
import com.example.demo.model.CurrencyDTO;
import org.springframework.web.bind.annotation.GetMapping;
```

```

import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CurrencyController {

    @GetMapping("/add")
    public CurrencyDTO add(@RequestParam CurrencyDTO currency1, @RequestParam CurrencyDTO currency2) {

        return currency1.add(currency2);
    }
}

-----
import java.util.Objects;

public record CurrencyDTO(String code, int amount) {

    public CurrencyDTO add(CurrencyDTO that) {
        if (!Objects.equals(this.code, that.code)) {
            throw new IllegalArgumentException("Codes are different");
        }

        return new CurrencyDTO(that.code, this.amount + that.amount);
    }
}

```

Автоматично захапва request parameter-а, за да може да го десериализира до CurrencyDTO!!!!

```

import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;

@Component
public class CurrencyConvertor implements Converter<String, CurrencyDTO> { //<source,
value>

    @Override
    public CurrencyDTO convert(String source) {
        var split = source.split(" ");

        return new CurrencyDTO(split[1], Integer.parseInt(split[0]));
    }
}

```

6.6. Request Body & 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.

What is the function of annotation `@RequestBody`? - To bind the HTTP body request to a domain/transfer object enabling automatic deserialization.

`@ResponseBody`

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

Сериализирай го като JSON и го сложи като част/цялото body на http response-a

```

@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"}

```

```

@Controller
public class DogController {

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

```

6.7. Form Objects

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

```

@PostMapping("/register")
public String register(@Valid 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.

```

    @PostMapping(value = "/calculator")
    public String calculate(@Valid CalculationRequestDTO calculationRequestDTO) {
        Model model = new Model();
        var result = exRateService.convert(calculationRequestDTO.getTo(),
                calculationRequestDTO.getFrom(),
                calculationRequestDTO.getAmount());
        model.addAttribute("result", result);
        return "calculator";
    }

```

6.8. Redirecting

After POST request

- 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
    }

```

6.9. @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 автоматично 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";
}
```

6.10. @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 е 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>

```

6.11. Inversion of Control for Spring MVC

Field Injection

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

```
@Autowired  
private ServiceA serviceA  
@Autowired  
private ServiceB serviceB  
@Autowired  
private ServiceC serviceC
```



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

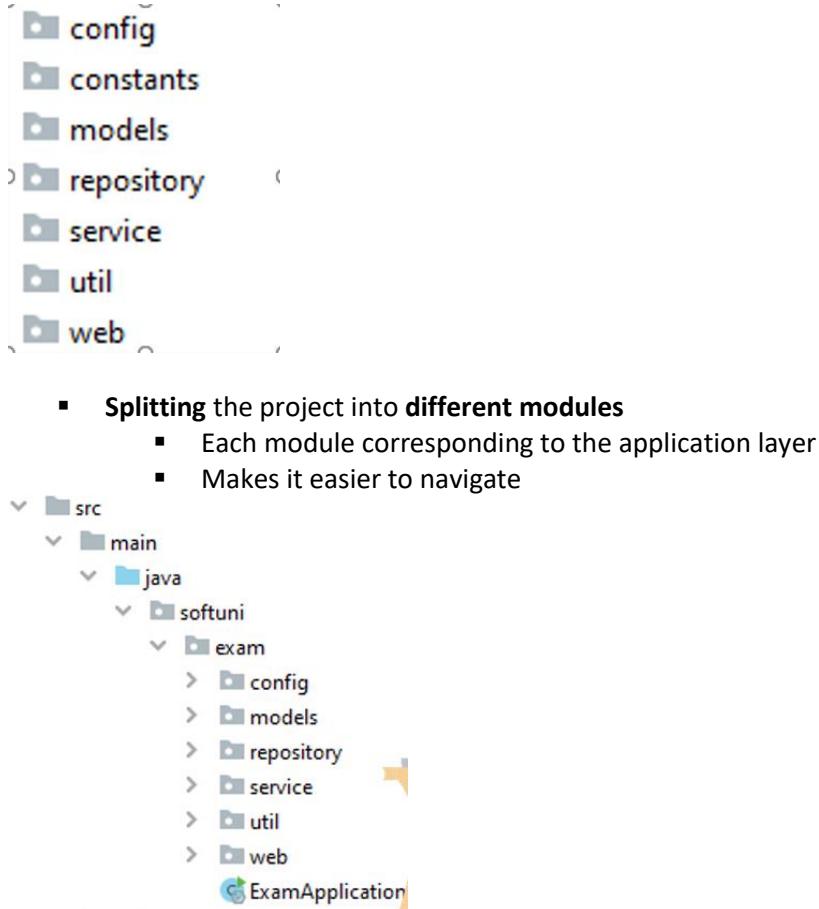
- Create setters for dependencies
- Can be combined easily with constructor injection
- Flexibility in dependency resolution or object reconfiguration!



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

6.12. Layers - The Correct Project Structure

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



6.13. 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.

7. State Management

7.1. 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

Name=Value

Attributes

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

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.com

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

Name:	ELOQUA
Content:	GUID=50B3A712CDAA4A208FE95CE1F2BA7063
Domain:	.oracle.com
Path:	/
Send for:	Any kind of connection
Accessible to script:	Yes
Created:	Monday, August 15, 2016 at 11:38:50 PM
Expires:	Wednesday, August 15, 2018 at 11:38:51 PM
Remove	

Examine Your Cookies 1

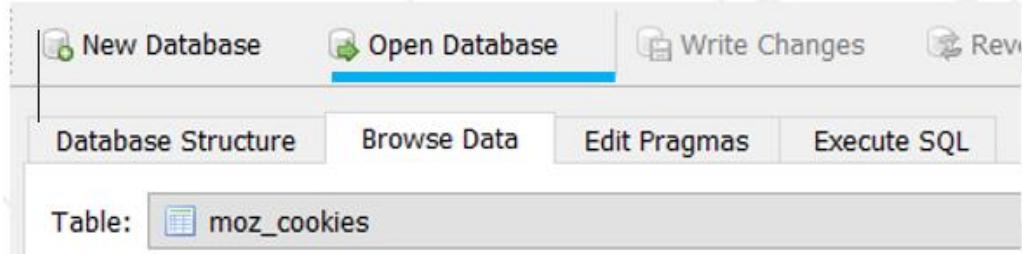
- Most cookies are stored in a **RDBMS**, usually **SQLite**
- Download DB Browser for SQLite (the **SQLite browser**) from [here - https://sqlitebrowser.org](https://sqlitebrowser.org)
- Location of Mozilla cookies

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

The screenshot shows the SQLite browser displaying the 'moz_cookies' table. Callouts point to specific columns with their corresponding names:

- Name: points to the 'name' column.
- Host: points to the 'host' column.
- Expiration date: points to the 'expiry' column.
- Created on: points to the 'creationTime' column.
- Value: points to the 'value' column.
- Paths: points to the 'path' column.
- Last accessed on: points to the 'lastAccessed' column.

baseDomain	originAttributes	name	value	host	path	expiry	lastAccessed	creationTime
1 softuni.bg		_ga	GA1.2.14749...	.softuni	/	1548331112	1485259173536000	1472458246652000
2 softuni.bg		cb-enabled	enabled	platform.soft...	/	1512124532	1485213524987000	1480588532898000
3 softuni.bg		cookies-notifi...	ok	judge.softuni...	/	1787818276	1485259172447000	1472458276862000
4 softuni.bg		cb-enabled	accepted	softuni.bg	/	1503994248	1485214353890000	1472458248921000

Examine Your Cookies 2

Using the **EditThisCookie**

<https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnplhplaeedifhcceomclgfbg?hl=en>

Kypc Spring Fundamentals - Document

http://localhost:8080/session

localhost | Idea-924ed50e

localhost | Idea-eee8a77a

localhost | io

localhost | JSESSIONID

Value
4A8A684F579F51771F5F7DE1C2E82CC4

Domain
localhost

Path
/

Expiration
Sun Jun 04 2023 15:25:03 GMT+0300 (Eastern European Summer Time)

SameSite

HostOnly Session Secure HttpOnly

 Help

Control Your Cookies – Firefox Browser

The screenshot shows the 'Privacy & Security' section of the Firefox preferences. It includes options for tracking protection, sending a 'Do Not Track' signal, and managing cookies and site data. A red box highlights the 'Cookies and Site Data' section, which shows stored data usage and provides options to clear or manage data.

Custom
Choose which trackers and scripts to block.

Send websites a "Do Not Track" signal that you don't want to be tracked [Learn more](#)

Always
 Only when LibreWolf is set to block known trackers

Cookies and Site Data

Your stored cookies, site data, and cache are currently using 46.0 KB of disk space. [Learn more](#)

Delete cookies and site data when LibreWolf is closed [Manage Exceptions...](#)

Logins and Passwords

Ask to save logins and passwords for websites [Exceptions...](#)
 Autofill logins and passwords [Saved Logins...](#)
 Suggest and generate strong passwords
 Show alerts about passwords for breached websites [Learn more](#)

Use a Primary Password [Learn more](#) [Change Primary Password...](#)
Formerly known as Master Password

History

LibreWolf will [Use custom settings for history](#)

Always use private browsing mode [Clear History...](#)

The screenshot shows the 'Cookies' dialog box. A search bar at the top contains 'softuni'. Below it, a table lists cookies from various sites, with one row for 'softuni.bg' selected. A blue box highlights the 'Remove Selected' button at the bottom left of the dialog.

Site	Cookie Name
judge.softuni.bg	cookies-notification
judge.softuni.bg	cookies-notification
platform.softuni.bg	cb-enabled
platform.softuni.bg	idsrv.session
platform.softuni.bg	idsrv
softuni.bg	_ym_uid

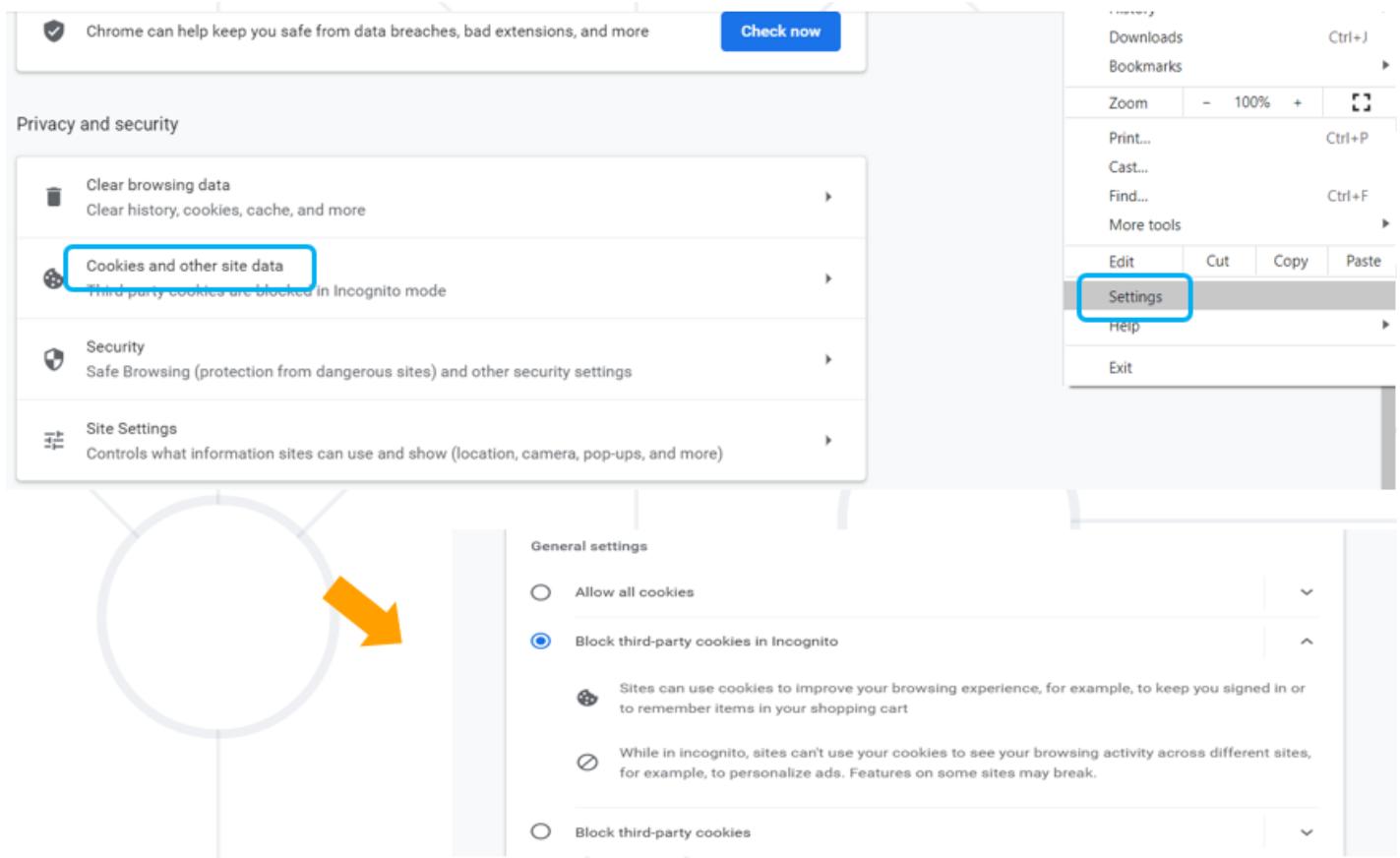
Name: _ym_uid
Content: 1501864502607682706
Domain: .softuni.bg
Path: /
Send For: Any type of connection
Expires: 25 July 2019, 19:35:01

[Remove Selected](#) [Remove All Shown](#) [Close](#)

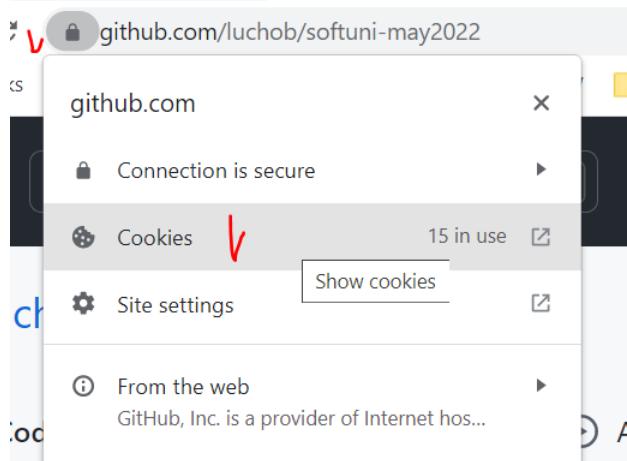
Browse cookies from a selected website

Delete a particular cookie or all cookies

Control Your Cookies – Chrome Browser



Или оттуда:



Cookies in use

	Allowed	Blocked
The following cookies were set when you viewed this page		
▾ github.com <ul style="list-style-type: none"> ▶ <input checked="" type="checkbox"/> Cookies ▶ <input checked="" type="checkbox"/> Local storage ▶ <input type="checkbox"/> Service Workers ▶ <input checked="" type="checkbox"/> Session storage ▶ <input type="checkbox"/> Shared Workers 		
Name	no cookie selected	
Content	no cookie selected	
Domain	no cookie selected	
Path	no cookie selected	

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!**

7.2. HTTP Sessions

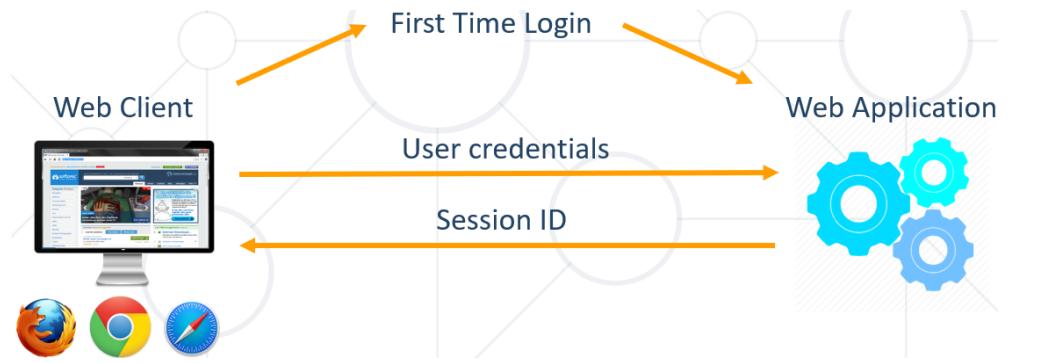
What Are Sessions?

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

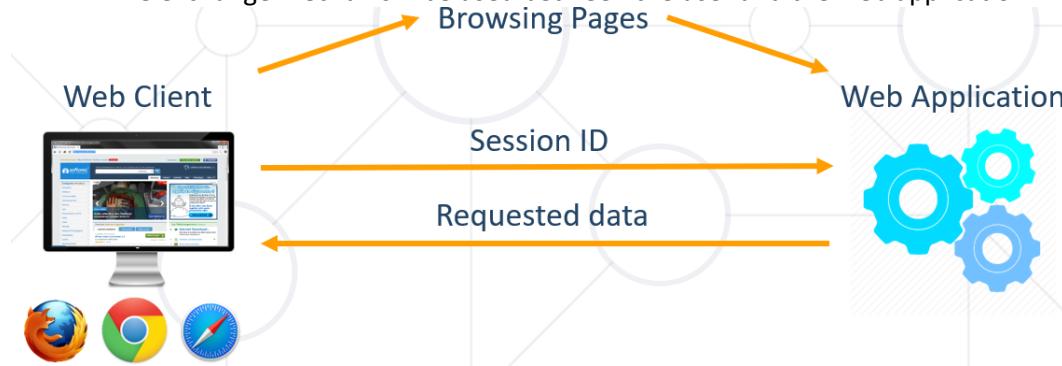


Session Management

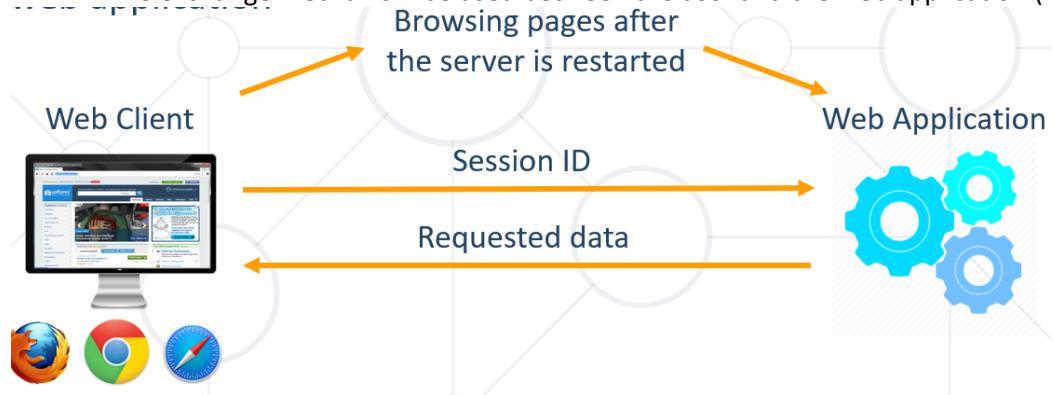
- The exchange mechanism be used between the user and the web application



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

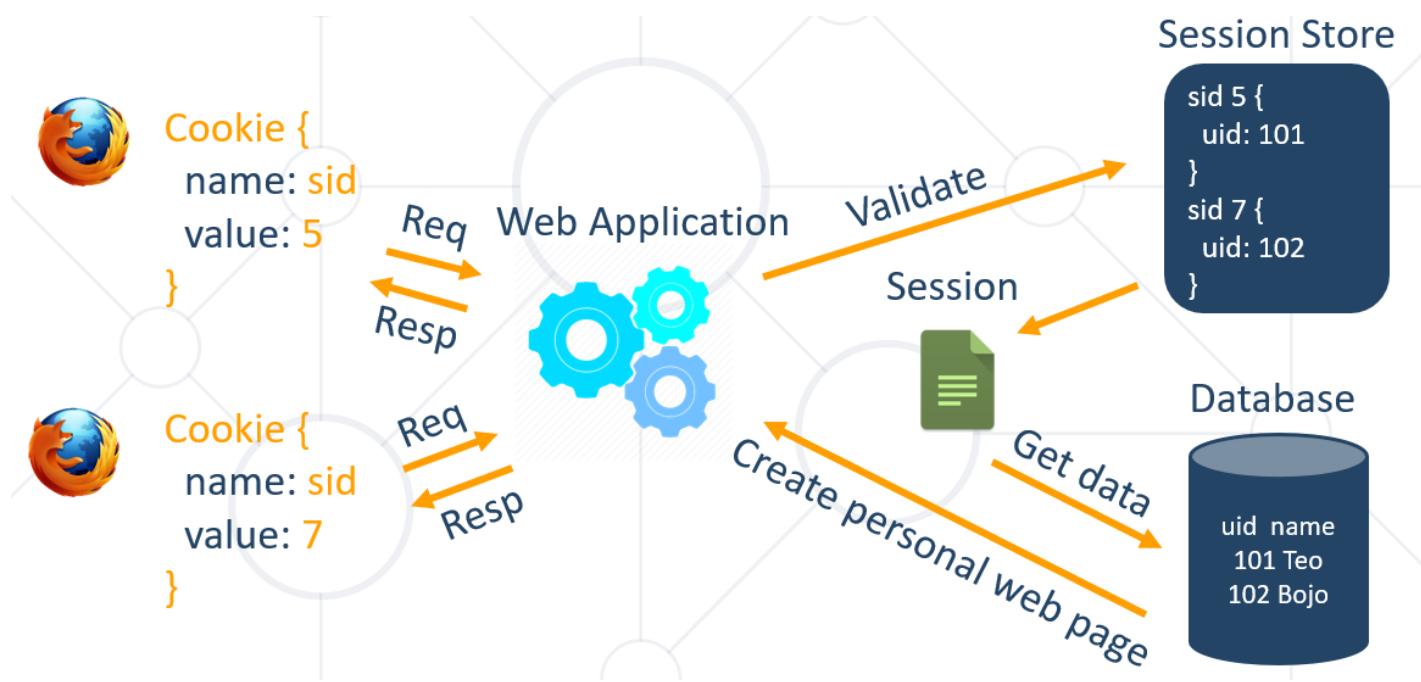


- The exchange mechanism be used between the user and the web application (when server is restarted)

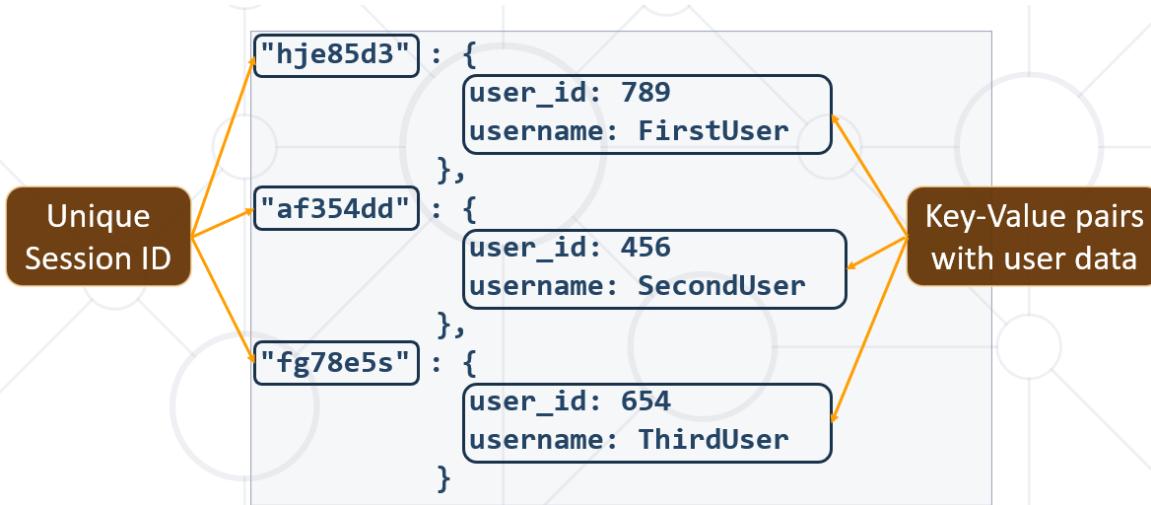


Relation with Cookies

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



Session Structure



7.3. 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

7.4. Demo cookies

screenshot

Добавено lang при заявка GET

The screenshot shows a browser developer tools window with the Network tab selected. A cookie named 'lang' is visible in the list, with its value set to 'de'. The cookie details show it was sent from 'localhost' with a path of '/' and 'Same-site connections only'. The Response Headers section includes 'Set-Cookie: lang=de'. The Request Headers section shows 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/

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
    ) {

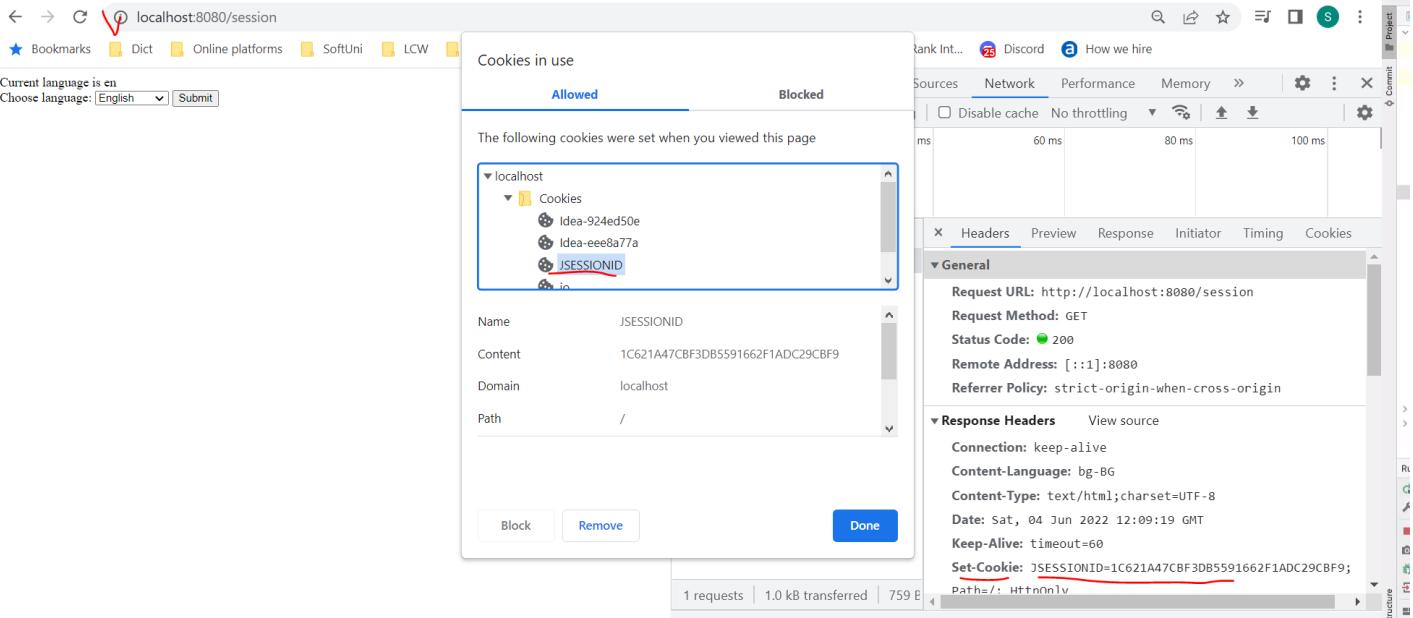
        Cookie cookie = new Cookie(LANG_COOKIE_NAME, language); //how we set a cookie - new
        Cookie(name, value)
        response.addCookie(cookie);
    response.setStatus(HttpStatus.NotFound)
        return "redirect:/cookies"; //редиректваме към този URL, не към html страница
    }
}

```

7.5. Demo HTTP session

screenshot

Добавено JSessionID при заявкa 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;
    
```

```

@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 празаписва реално
        session.setAttribute(LANG_SESSION_ATTRIBUTE, language);

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

```

8. Spring Essentials

8.1. Kotlin example

KotlinApplication.kt

```

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

@SpringBootApplication
class KotlinApplication

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

```

hello.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 дефалтна 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";
    }
}

```

8.2. 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 б
пътят 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>

```

8.3. 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.

```
  

```

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="${student.passExam}">Show results</div>  ако е true ќе се покаже само
<div th:unless="${student.passExam}">Not pass</div>  ако е false ќе 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 pl-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 генерираат 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

```

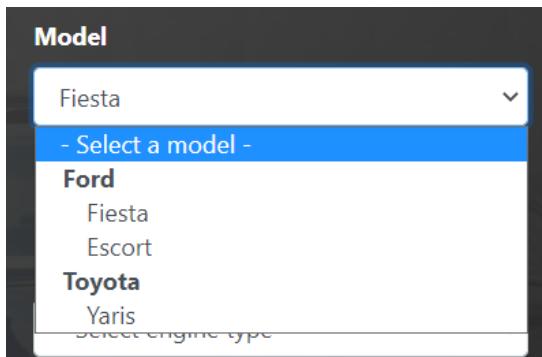
<form th:action="@{/offers/add}" th:method="POST" th:object="${addOfferModel}" class="main-form mx-auto col-md-8 d-flex flex-column justify-content-center">
    <div class="row">
        <div class="form-group col-md-6 mb-3">
            <label class="text-center text-white font-weight-bold" for="modelId">Model</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}">
                </option>
            </select>
        </div>
    </div>

```

```

        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="/"></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>
                </li>
            </ul>
        </div>
    </nav>
</div>
</body>
</html>
```

```

        <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>
        </div>
    </nav>
</header>
```

```

        
    </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>
...

```

8.5. 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
}

```

8.6. 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);
    ...
}

```

```

@Controller
public class HomeController {

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

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

```

Така го логваме потребителя:

```
httpSession.setAttribute("user", userServiceModel);
```

Така го разлогваме - invalidate

```

@GetMapping("/logout")
public String logout(HttpServletRequest 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(название 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(HttpServletRequest 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) {
    ...
}
```

8.7. 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 username;

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

```

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

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

public Long getId() {
    return id;
}

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

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}
}

```

```

@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!!!

```

@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>

```

9. 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

```

```

2 import javax.validation.constraints.Email;
3 import javax.validation.constraints.NotEmpty;
4 import javax.validation.constraints.NotNull;
5 import javax.validation.constraints.Size;
6
7 public class User {
8     @NotEmpty
9     @Email
10    private String email;
11
12    @NotEmpty
13    @Size(min = 2)
14    private String name;
15
16    @NotEmpty
17    @Size(min = 6)
18    private String password;
19
20    @NotEmpty
21    @Size(min = 6)
22    private String confirmPassword;
23
24    private String role;
25
26    public String getEmail() {
27        return email;
28    }
29
30    public void setEmail(String email) {
31        this.email = email;
32    }
33
34    public String getName() {
35        return name;
36    }
37
38    public void setName(String name) {
39        this.name = name;
40    }
41
42    public String getPassword() {
43        return password;
44    }
45
46    public void setPassword(String password) {
47        this.password = password;
48    }
49
50    public String getConfirmPassword() {
51        return confirmPassword;
52    }
53
54    public void setConfirmPassword(String confirmPassword) {
55        this.confirmPassword = confirmPassword;
56    }
57
58    public String getRole() {
59        return role;
60    }
61
62    public void setRole(String role) {
63        this.role = role;
64    }
65
66    @Override
67    public String toString() {
68        return "User{" +
69            "email=" + email +
70            ", name=" + name +
71            ", password=" + password +
72            ", confirmPassword=" + confirmPassword +
73            ", role=" + role +
74            '}';
75    }
76}

```

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>

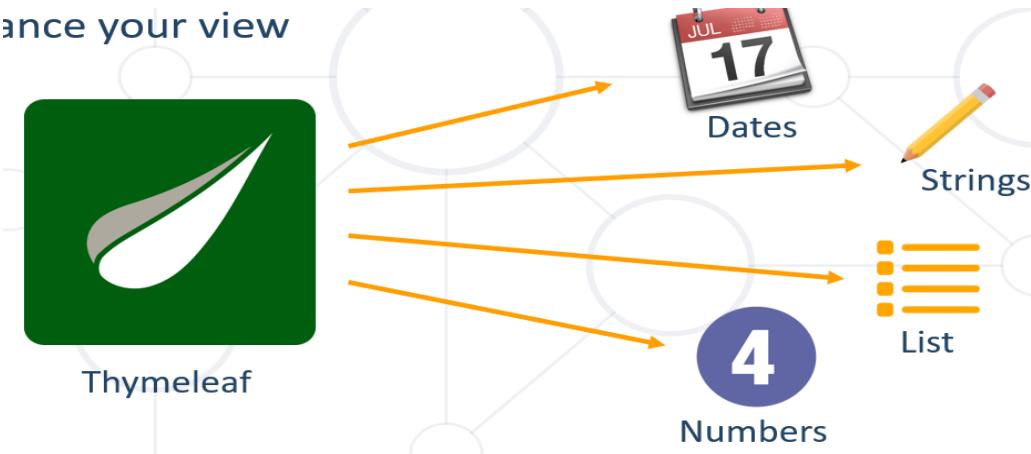
```

9.1. Thymeleaf Helpers

Helpers

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

Enhance your view



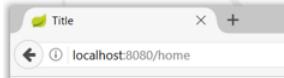
Date – Custom Format

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>
```



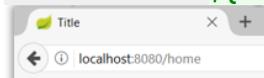
Date – Week Name of Day

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>
```

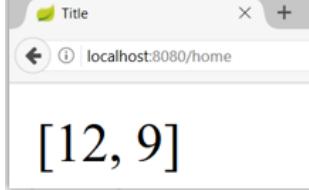


Date – List Days

```
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>
```

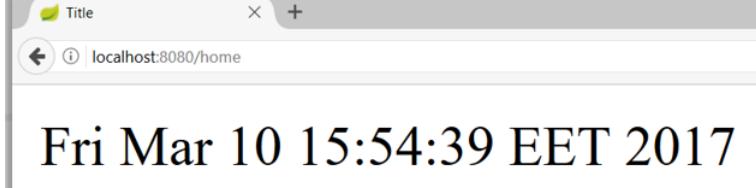


Date – Get Current Date

```
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>
```



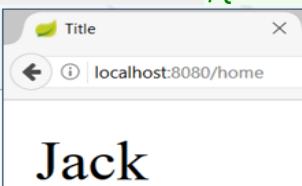
Strings – Substring

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>
```



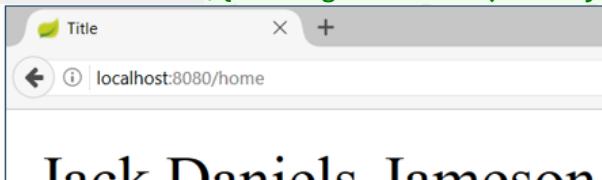
Strings – Join

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
```



Jack Daniels-Jameson

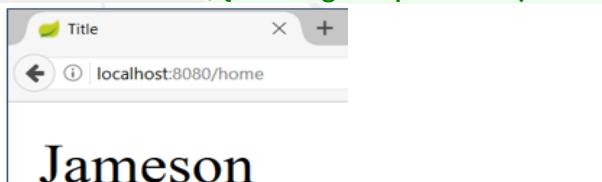
Strings – Capitalize

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
```



Jameson

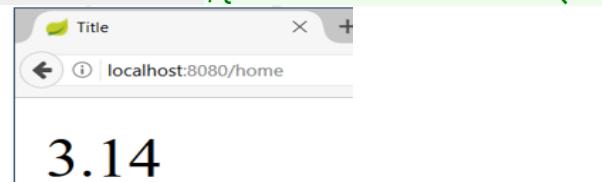
Numbers – Format

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>
```



3.14

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

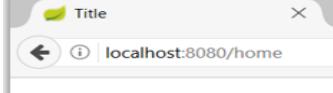
WhiskeyController.java

```

@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)}
```



83.95

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

Извикване на source js файл може да стане и през `th:src`

```
<script th:inline="javascript" th:src="@{/js/viewAndSearchOrders.js}"
type="text/javascript"></script>
```

9.2. 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>

```

Name

Invalid name

Пример 2:

AuthController.java

```

@Controller
public class AuthController {

    //Ако го има веќе 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">
        </div>
    </div>

```

```

        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 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="${userToLogin}">
                    <div class="text-center text-light">
                        <h3 th:if="${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>
                    </div>
                </form>
            </div>
        </div>
    </div>

```

```

        </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

```

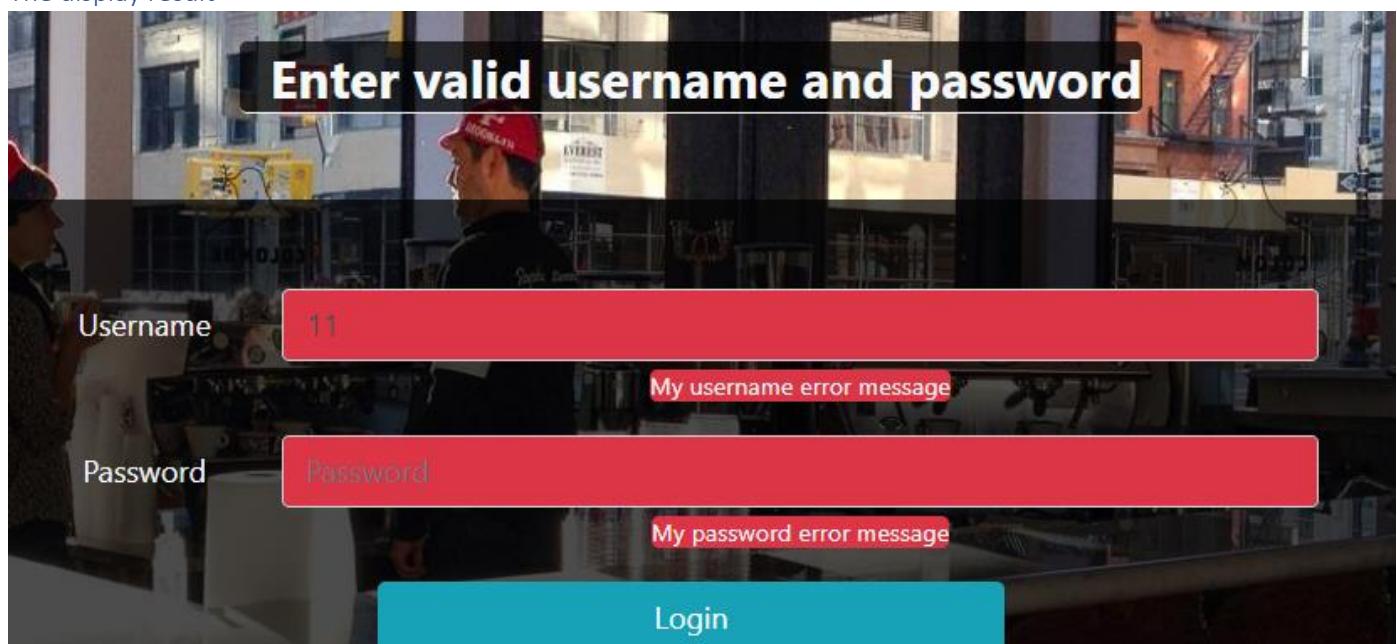
public class UserLoginBindingModelDTO {
    @Size(min = 5, max = 20, message = "My username error message")
    private String username;

    @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"/>

```

В)

//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>

```

С)

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">

```

```



```

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 р

```

<p class="invalid-feedback errors alert alert-danger">
    Passwords should match. //валидира се от сложната custom анотация за паролите - с таг
елемент р 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 искали листа от всички грешки

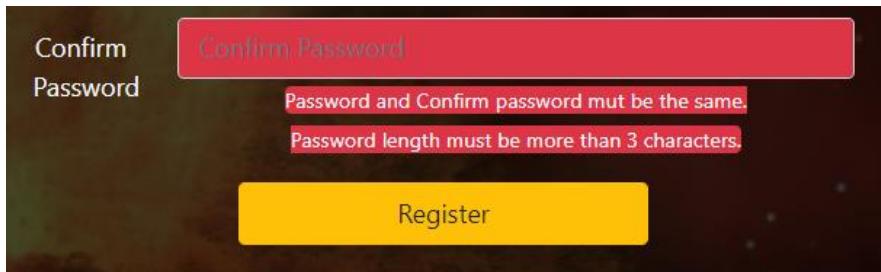
```

```
</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>
```



```
@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
                .Annotацията @FieldMatch на UserRegisterDto
                .addPropertyNode(this.second) //задай грешката на второто поле от класа
            UserRegisterDto
                .addConstraintViolation()
                .disableDefaultConstraintViolation(); //без default message при грешка
        }
    }

    return valid;
}
}

```

9.3. 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&createDatabaseIfNotExist=true&serverTimezone=UTC
    username: root
    password:
  mvc:
    hiddenmethod:
      filter:
        enabled: true
```

Каково 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 DTO за валидация на данни и едно DTO за презаписване на данните

Настойка в application.yml

```

spring:
  datasource:
    driverClassName: com.mysql.cj.jdbc.Driver
    url:
      jdbc:mysql://localhost:3306/mobilele?allowPublicKeyRetrieval=true&useSSL=false&createDatabaseIfNotExists=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">

```

9.4. 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();
    }

}

```

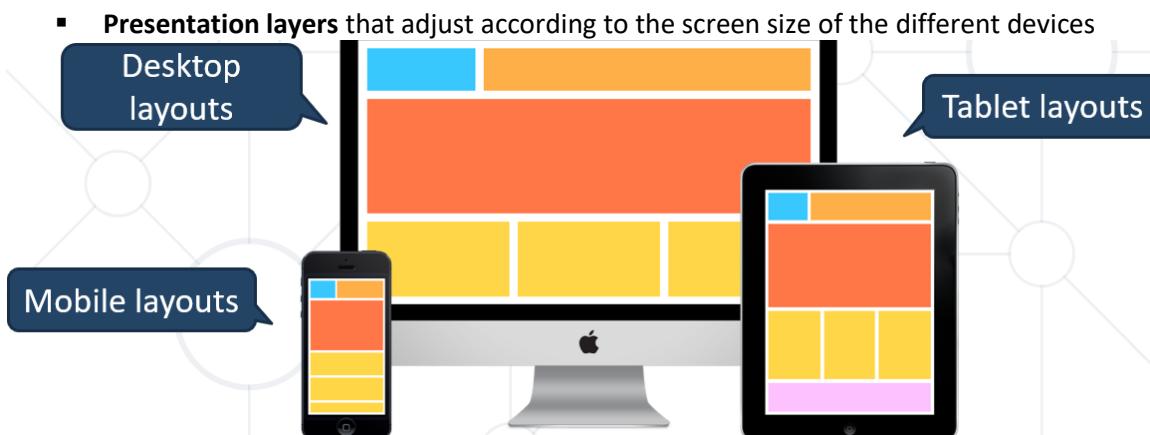
10. Bootstrap, Front-End Basics

10.1. JavaScript

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

10.2. Bootstrap

What is a Responsive Design?



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+OrBnVFBL6D0itfPri4tjfHxaWutUpFmBp4vmVor"
        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 тъло написани, някои неща в React, Vue и Angular вземат/копират JQuery.

10.3. 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>
```



Column one Column two Column three

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 <code>max-width</code>	None (auto)	540px	720px	960px	1140px	1320px
Class prefix	<code>.col-</code>	<code>.col-sm-</code>	<code>.col-md-</code>	<code>.col-lg-</code>	<code>.col-xl-</code>	<code>.col-xxl-</code>
# 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>
```

10.4. 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);  
    }  
}
```

10.5. Bootstrap Components

Button Groups

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

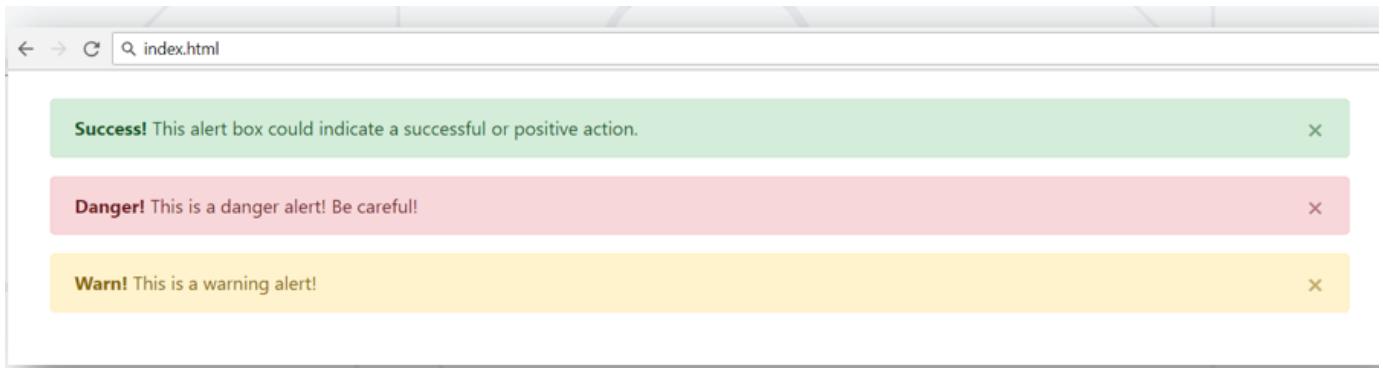


```
<button type="button" class="btn btn-primary">Primary</button>  
<button type="button" class="btn btn-secondary">Secondary</button>  
<button type="button" class="btn btn-success">Success</button>  
<button type="button" class="btn btn-danger">Danger</button>
```

Alerts

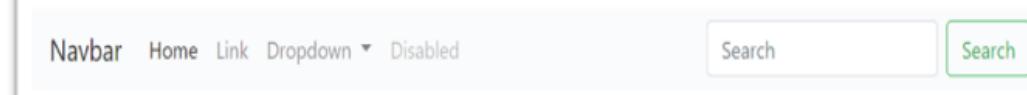
```
<div class="alert alert-success alert-dismissible">  
    <a class="close" data-dismiss="alert" aria-label="close">x</a>  
    <strong>Success!</strong> This alert box could indicate a successful or positive  
    action.  
</div>
```

И затваряме/изтриваме дом елемента без JS!



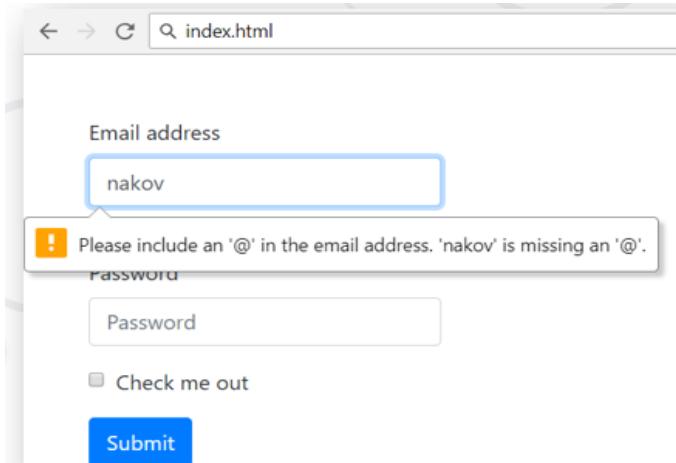
Nav and Navbar

- 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>

```

Hello, world!

This is a simple hero unit, a simple jumbotron-style component for calling extra attention to featured content or information.

It uses utility classes for typography and spacing to space content out within the larger container.

[Learn more](#)

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">
      
      <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">
      
      <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">
      
      <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>

```

```

        </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>

```

10.6. 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-hVVnYaiADRT02PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8=
" 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 technology';
    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

```
$('.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

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

```
//Mouse events  
//Form events  
//Keyboard 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
```

11. Web-API-and-REST-Controllers

11.1. 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.

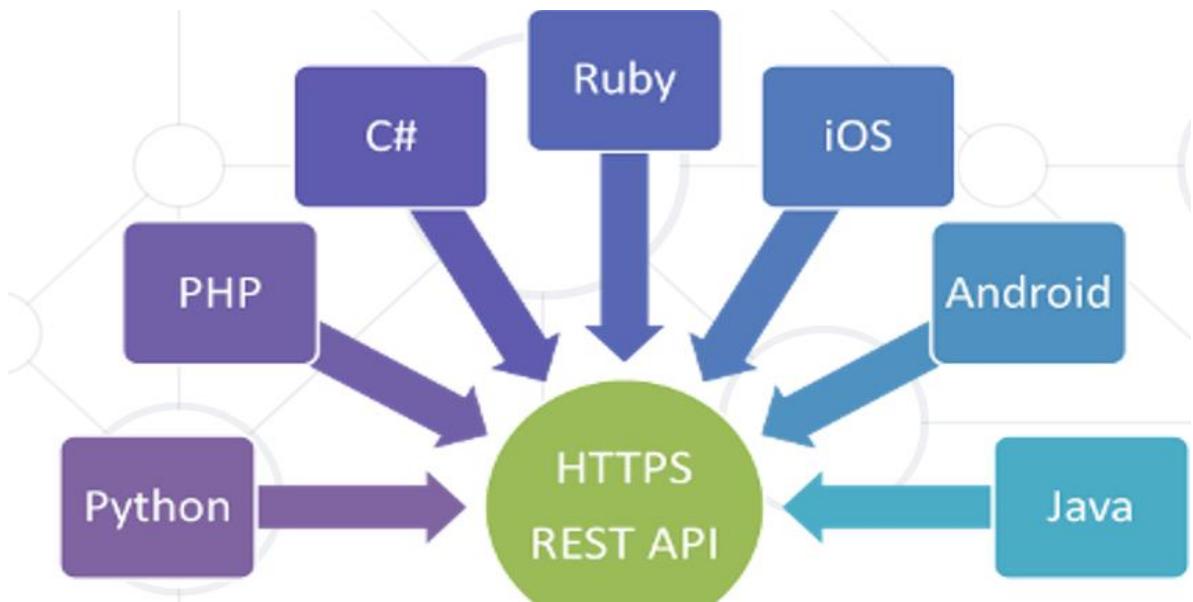
Пример:

```
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-a 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

11.2. 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);
}

```

```

(HttpServletRequest response)
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 `ResponseType` 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 –
        //пропъртита 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,
        @Valid @RequestBody BookDTO bookDTO){
        //todo
        throw new UnsupportedOperationException("coming soon");
    }
}

```

11.3. 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

```

```

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();
    }
}

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);
            }
        }
    }
}

@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; //наша 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

11.4. Spring Reactive Webclient as a Rest Template

The screenshot shows the Spring Initializr interface with the following configuration:

- Project:** Maven Project (selected), Gradle Project (unchecked).
- Language:** Java (selected), Kotlin (unchecked), Groovy (unchecked).
- Spring Boot:** 2.7.1 (selected), 3.0.0 (SNAPSHOT), 3.0.0 (M3), 2.7.2 (SNAPSHOT), 2.6.10 (SNAPSHOT), 2.6.9.
- Project Metadata:**
 - Group: bg.softuni
 - Artifact: webclient
 - Name: webclient
 - Description: Demo for webclient
 - Package name: bg.softuni.webclient
- Dependencies:**
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container. (Has a red checkmark)
 - Spring Reactive Web** (WEB): Build reactive web applications with Spring WebFlux and Netty. (Has a red checkmark)
- Buttons at the bottom:**
 - GENERATE CTRL + ⌘
 - EXPLORE CTRL + SPACE
 - SHARE...

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

application.yml

```
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!"  
    }  
}
```

```
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  
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 ще може да  
изчака, за да се expose-не за обслужването на някой друг request  
  
        System.out.println(user);  
    }  
}
```

```
import com.fasterxml.jackson.annotation.JsonProperty;  
import com.fasterxml.jackson.annotation.JsonInclude;  
  
@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 + '\'' +
            '}';
    }
}

```

11.5. 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);
```

▼
 Peter
 ▼
 Maria

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);
```

11.6. Browser Events and DOM Events – plain Vanilla JS

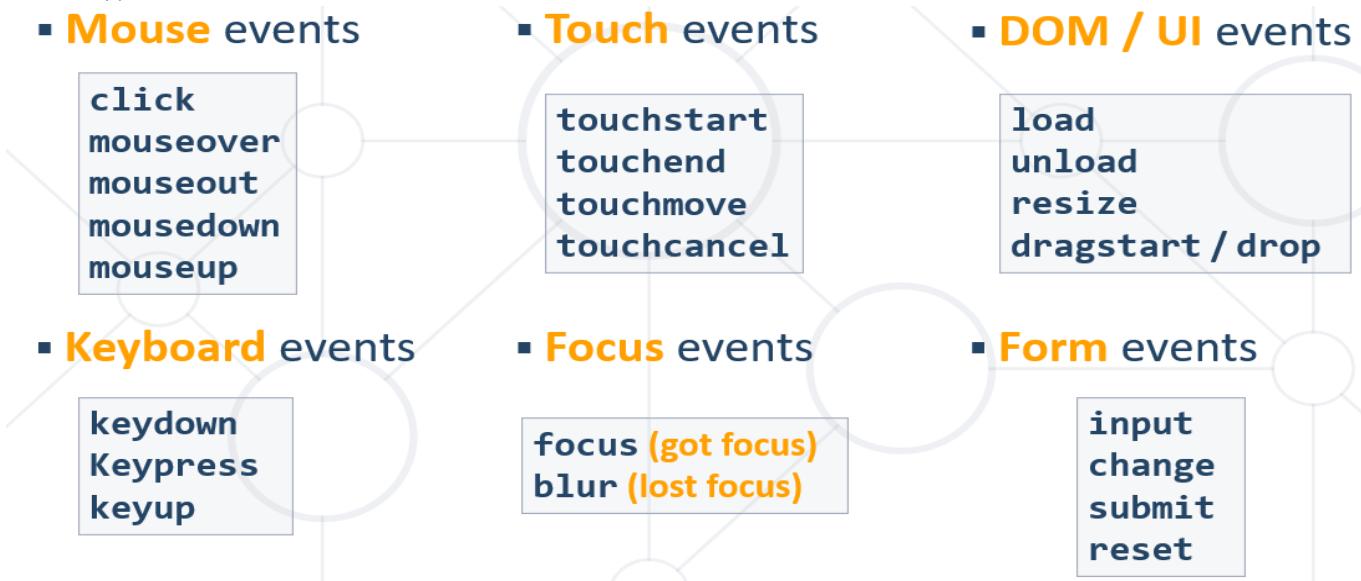
Handling Events in JS

- 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'.

11.7. 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);
    }));
});

```

12. Spring Security

12.1. Защита от следните атаки

- **Cross-Site Request Forgery (CSRF)** - is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. A CSRF attack exploits a vulnerability in a Web application if it cannot differentiate between a request generated by an individual user and a request generated by a user without their consent. Например ако се знаят/излагат name и value на някои DOM/HTML елементи.
- **Cross-Site Scripting (XSS)** - an attack in which an attacker injects malicious executable scripts into the code of a trusted application or website. Attackers often initiate an XSS attack by sending a malicious link to a user and enticing the user to click it.
- **Cross-Origin Resource Sharing (CORS)** - a mechanism for integrating applications. CORS defines a way for client web applications that are loaded in one domain, **to be allowed or not, to interact** with resources in a different domain.
- **Session Fixation** - In the session fixation attack, the attacker already has access to a valid session and tries to force the victim to use that particular session for his or her own purposes. The session fixation attack “fixes” an established session on the victim's browser, so the attack starts before the user logs in.
- **SQL Injection** - a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details.
- Timing атаки
- Etc.

12.2. Filters and Interceptors

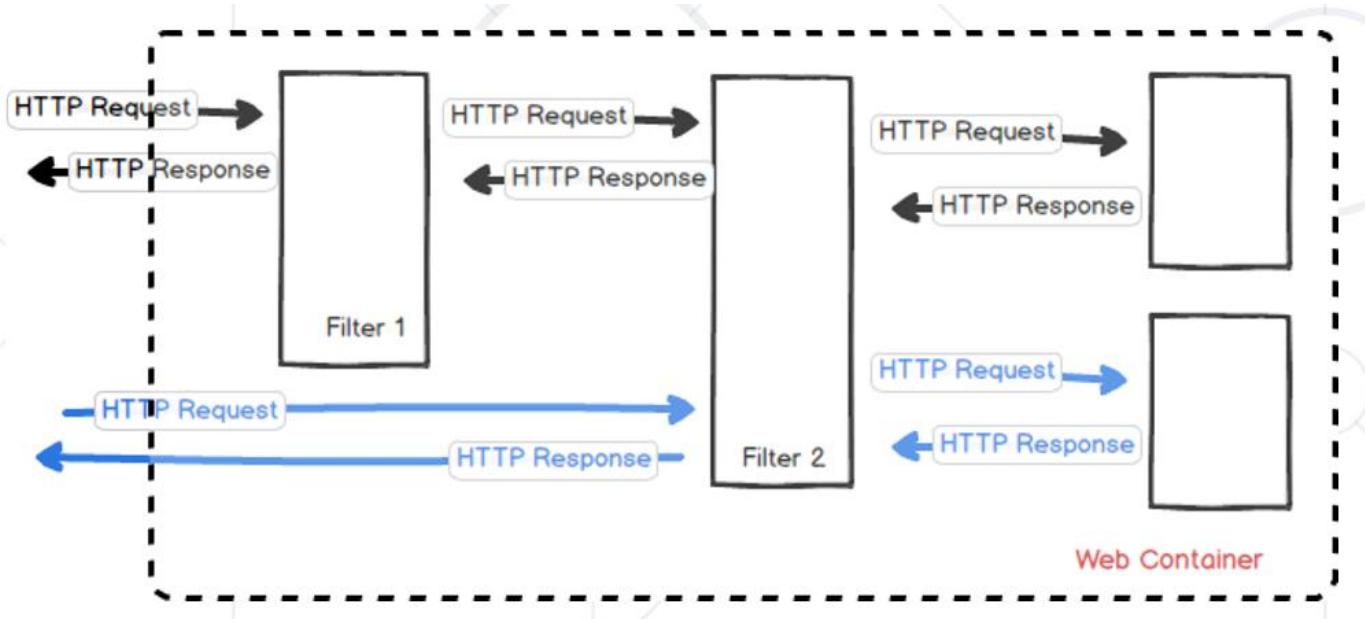
Филтрите работят върху Servlet-и.

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>

```



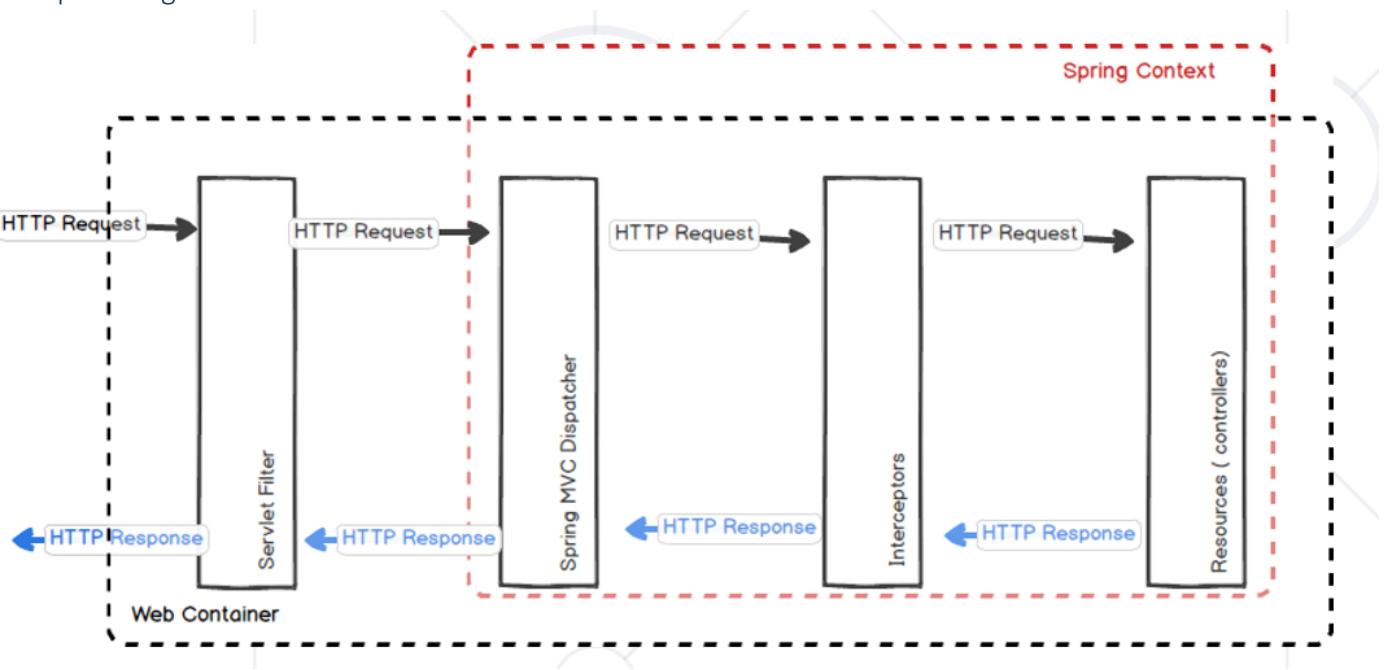
Hello, Pesho!

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 контролера
 - **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 LoggingInterceptor 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-a и response-a са минали примерно.

        return 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);
}
}

```

12.3. 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

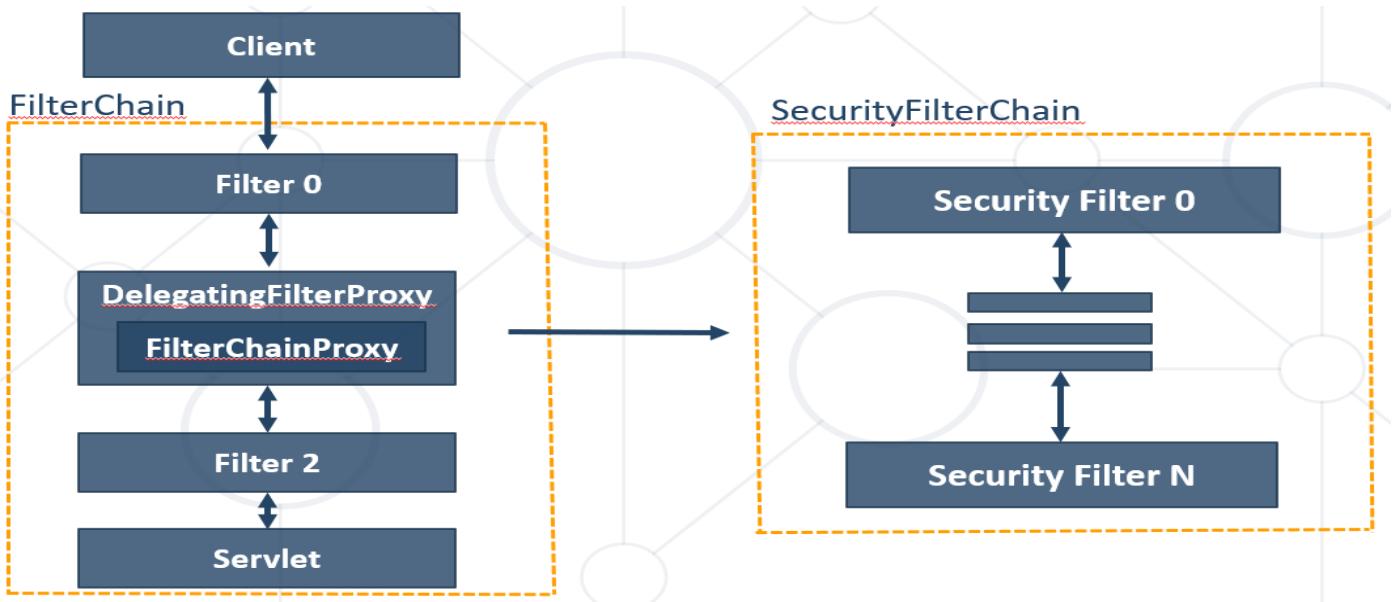


- **Authorization**
 - What you are allowed to do

**ACCESS
DENIED**

Spring Security Filter Chain

Spring Security реално е един филтър.

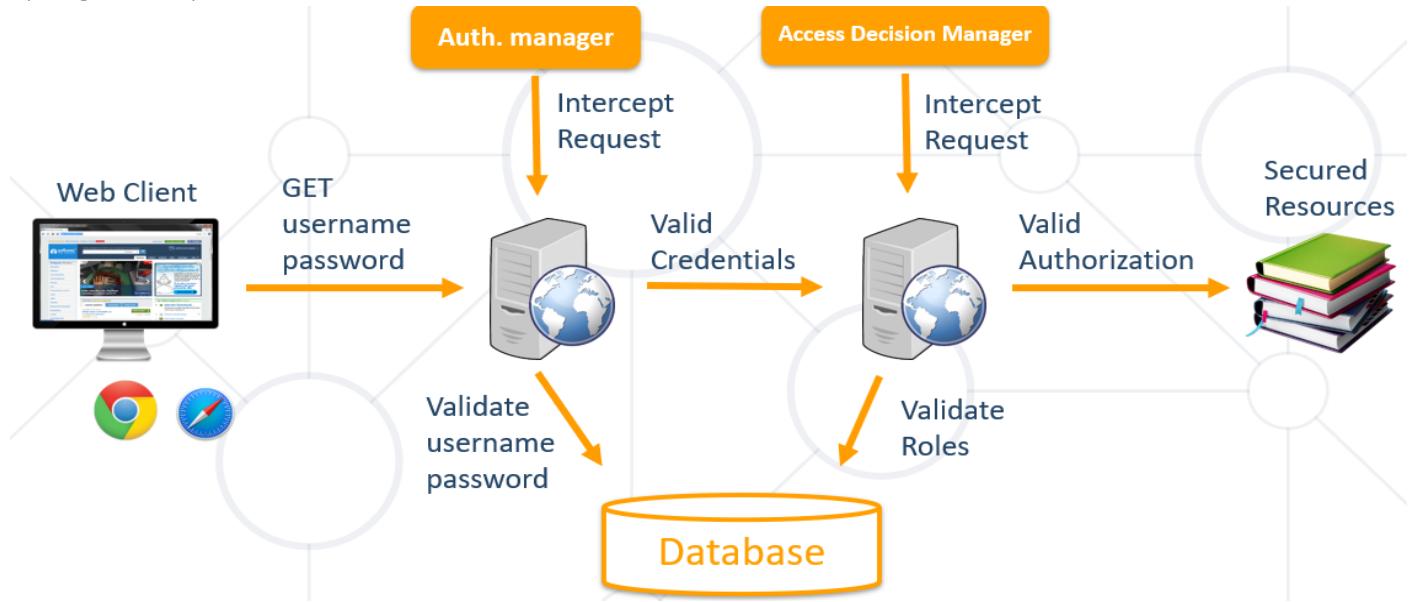


Security Context and Authentication

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



Spring Security Mechanism



Spring Security Maven/Gradle

build.gradle

```
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 WebSecurityConfigurerAdapter

- Extending the **WebSecurityConfigurerAdapter** class - deprecate-HAT

```
@Configuration
@EnableWebSecurity //Used with 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 SecurityFilterChain

- Creating the **SecurityFilterChain** bean.

```

@Configuration
//@EnableGlobalMethodSecurity(prePostEnabled = true) - we will not use @PreAuthorize on method level
@EnableWebSecurity - can be skipped when using SecurityFilterChain
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' />

```

SecurityConfig Logout and Remember Me demo

```

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeHttpRequests(
            //which urls are visible by which users
            authorizeRequests -> authorizeRequests
                //all static resources which are situated in js, images, css are visible
        for anyone
            .requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
                //allow anyone to see the pages
                .requestMatchers("/", "/home", "/about", "/destinations", "/contacts",
                    "/register", "/login", "/offers", "/offer-details", "/profile",
                    "profile-change", "/fonts/**", "/plugins/**", "/login-
error").permitAll()
                .requestMatchers("/destination-add", "/offer-add", "/admin-user-
management",
                    "/profile-change-admin", "/profile-change-admin/delete-user",
                    "/profile-change-admin-user").hasRole(UserRole.ADMIN.name())
    }
}

```

```

.antMatchers("/", "/login", "/register").permitAll()
requestMatchers е новия метод на мястото на antMatchers

        //all other requests are authenticated
        .anyRequest().authenticated()

).formLogin()
    formLogin -> {
        formLogin
            //redirect here when we access something which is not allowed
            // also, this is the page where we perform login and where we
            VALIDATE!!!
            .loginPage("/login")
            //the name of the input fields (in login.html)
            .usernameParameter("email")
            .passwordParameter("password")
            .defaultSuccessUrl("/home", true)
            .failureForwardUrl("/login-error");
    }
).logout()
    logout -> {
        logout
            // the url where we should POST something in order to perform the
        logout
            .logoutUrl("/logout")
            // where to go when logged out
            .logoutSuccessUrl("/home")
            // invalidate the http session
            .invalidateHttpSession(true);
    }
).rememberMe()
    rememberMe -> {
        rememberMe
            .key("rememberme")
            .rememberMeParameter("rememberme")
            .rememberMeCookieName("rememberme");
    }
};

return httpSecurity.build();
}

```

Application

- Manifest
- Service workers
- Storage

Storage

- Local storage
- Session storage
- IndexedDB
- Web SQL
- Cookies

http://localhost:8080

Private state tokens

Interest groups

Shared storage

Name	Value	Do...	Path	Ex...	Size	Ht...	Se...	Sa...	Par...	P...
cf_clearance	sXTVT68W2Wg...	.cl...	/	20...	114	✓	✓	No...	Me...	
rememberMeCookieName	c2ltb25hdGFm...	loc...	/	20...	116	✓			Me...	
JSESSIONID	rememberMeCookieName ...		/	Se...	42	✓			Me...	
_ga_NKMQJY2D2H	GS1.1.1699362...	.cl...	/	20...	51				Me...	
_uetvid	d08acdf073261...	.cl...	/	20...	39				Me...	
_clk	1h707dp[2fg5cl...	/	20...	25				Me...	
_ga	GA1.1.1952945...	.cl...	/	20...	30		✓	No...	Me...	
_gcl_au	1.1.102722541...	.cl...	/	20...	32				Me...	
_ces	v~13b5cbdf64...	.cl...	/	20...	252			Str...	Me...	
_conv_r	s%3Awwww.goo...	.cl...	/	20...	47			Lax	Me...	
_conv_v	vi%3A1*sc%3A...	.cl...	/	20...	147			Lax	Me...	
_fbp	fb.1.169823210...	.cl...	/	20...	32			Lax	Me...	

```

httpSecurity.authorizeHttpRequests(
    authorizeRequests -> authorizeRequests

```

```
.requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
```

No Access Handling

SecurityConfiguration.java

```
.and()  
    .exceptionHandling().accessDeniedPage( '/unauthorized' )
```

AccessController.java

```
@GetMapping( '/unauthorized' )  
public String unauthorized() {  
    return 'unauthorized';  
}
```

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

```

        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);
    }
```

12.4. Cross-Site Request Forgery (подправяне/фалшификация)

Spring CSRF 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>
```

Вариант при POST rest заявка и използване на CSRF

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}">
```

Автоматично си го взема токена

```
<meta charset="UTF-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<meta name="_csrf" content="${_csrf.token}" />
<meta name="_csrf_header" content="${_csrf.headerName}" />
<title>Pathfinder</title>
<link rel="stylesheet" href="/css/bootstrap.min.css"/>
<link rel="stylesheet" href="/css/styles.css">
<link rel="stylesheet" href="/css/mobile.css">
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.15.4/css/all.css"
      integrity="sha384-DyZ88mC6Up2uqS4h/KRgHuoeGwBcD4Ng9SiP4dIRy0EXTlnuz47vAwme6w"/>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.4.0/dist/leaflet.css"
      integrity="sha512-pu8pdR87980ZvTTbP4A8Ix/l+A4dHDD8DGqYX6RQ+9jxkRFcLaxxQb/SJA1"/>
```

```
@PostMapping(value = "/{routeId}/comments", consumes = "application/json", produces = "application/json")
public ResponseEntity<CommentDisplayView> createComment(@PathVariable("routeId") Long routeId,
                                                       @AuthenticationPrincipal UserDetails userDetails,
                                                       @RequestBody CommentMessageDto commentDto) {
    CommentCreationDto commentCreationDto = new CommentCreationDto(
        userDetails.getUsername(),
        routeId,
        commentDto.getMessage()
    );

    CommentDisplayView comment = commentService.createComment(commentCreationDto);

    return ResponseEntity
        .created(URI.create(String.format("/api/%d/comments/%d", routeId, comment.getId())))
        .body(comment);
}
```

Accept, a не accepts

```

const routeId = document.getElementById('routeId').value
const commentForm = document.getElementById('commentForm')
commentForm.addEventListener("submit", handleFormSubmission)

const csrfHeaderName = document.head.querySelector('[name=_csrf_header]').content
const csrfHeaderValue = document.head.querySelector('[name=_csrf]').content

const commentContainer = document.getElementById('commentCtnr')

async function handleFormSubmission(event) {
    event.preventDefault()

    const messageVal = document.getElementById('message').value

    fetch(`http://localhost:8080/api/${routeId}/comments`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Accepts': 'application/json',
            [csrfHeaderName]: csrfHeaderValue
        },
        body: JSON.stringify({
            message: messageVal
        })
    }).then(res => res.json())
    .then(data => {
        document.getElementById('message').value = ""
        commentContainer.innerHTML += commentAsHtml(data)
    })
}

function commentAsHtml(comment) {
    let commentHtml = '<div>\n'
    commentHtml += `<h4>${comment.authorName}</h4>\n'
    commentHtml += `<p>${comment.message}</p>\n'
    commentHtml += '</div>\n'

    return commentHtml
}

```

12.5. Thymeleaf Security

Functionality to Thymeleaf

pom.xml

```

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

```

```
build.gradle
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>
```

12.6. 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.AuthenticationBuilder;
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()
този код валидира Login само за @Controller анотация и само през html формулар
            //configure login with login html form with two input 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 -> kwrnfwfewkfjkdqwdqwpqfLsw (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>

```

12.7. 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 only 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 logout
        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);
}

@Bean
public UserDetailsService userDetailsService() {
    UserDetails user = User.withDefaultPasswordEncoder()
        .username("user").password("password").roles("ADMIN").build();

    return new InMemoryUserDetailsManager(user); //in memory
}

Functional style will be the only possible style in future SpringSecurity releases.

@Bean
public SecurityFilterChain filterChainFunctionalStyle(HttpSecurity http) throws Exception {
    http.authorizeRequests(authRequest ->
        authRequest.requestMatchers("/").permitAll()
            .anyRequest().authenticated())
        .formLogin(formLogin ->
            formLogin.loginPage("/login").permitAll())
        .logout(logout -> logout.permitAll());

    return http.build();
}
}

```

```

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 user_repository;

    public AppUserDetailsService(UserRepository user_repository) {
        this.user_repository = user_repository;
    }

    @Override
    public UserDetails loadUserByUsername(String username)
        throws UsernameNotFoundException {
        return user_repository.
            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";
    }
}

```

12.8. 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);
    }
}

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;
    }
}

```

```

    }
}

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);
    }
}

import bg.softuni.mobilele.service.StatsService;
import org.springframework.stereotype.Component;
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;
    }
}

```

```

@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;
    }
}

<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>

```



12.9. @Secured анотацията

```
import org.springframework.security.access.annotation.Secured;
```

```

@Secured(Role.ADMIN)
@GetMapping("/add")
public String add(Model model) {
    model.addAttribute("ships", shipService.getShips());

    return "contract-add";
}

```

12.10. SecurityLogoutHandler

Logout demo without SecurityConfig:

В моята апликация имам функционалност за смяна на паролата. Питането ми е как може след като променя паролата да накарам потребителя да излезе от профила си и да се логне отново с новата си парола. Има ли възможност това нещо да стане в конфигурацията на Spring Security с GET mapping request.

luchob commented 15 days ago

Owner



...

Здрави!

Може да го накараш с браузър редирект, обаче това не е твърде сигурно защото може и да не сработи навсякъде. Не съм го ползвал никого, на мисля, че можеш да го логаутнеш програмно използвайки `SecurityContextLogoutHandler`. Кодът би могъл да изглежда като нещо от сорт на:

```

public void logout(HttpServletRequest request, HttpServletResponse response) {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (auth != null) {
        new SecurityContextLogoutHandler().logout(request, response, auth);
    }
}

```

`request` и `response` сринг инжектира в контролерите.

```

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.support.RedirectAttributes;

@Controller
public class LoginController {

    @GetMapping("/login")
    public String login() {
        return "login";
    }

    @PostMapping("/login-error")

```

```

public String onFailedLogin(
    @ModelAttribute.UsernamePasswordAuthenticationFilter
        .SPRING_SECURITY_FORM_USERNAME_KEY) String username,
    RedirectAttributes redirectAttributes) {

    redirectAttributes.addFlashAttribute(UsernamePasswordAuthenticationFilter
        .SPRING_SECURITY_FORM_USERNAME_KEY, username);
    redirectAttributes.addFlashAttribute(
        "bad_credentials", true);

    return "redirect:/login";
}

@RequestMapping(value = "/logout")
public String logout(HttpServletRequest request,
                      HttpServletResponse response,
                      Authentication authentication) {
    if (authentication != null) {
        new SecurityContextLogoutHandler()
            .logout(request, response, authentication);
    }

    return "redirect:/login";
}
}

```

12.11. 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();
}

```

12.12. SecurityExpressionRoot and MethodSecurityExpressionOperations

Този подход бил най-добрия при използването на Spring security

Watch lection of October 2021 – Events for this specific security customized configuration and usage
 Watch also the 3d QA and Workshop lection of June 2022 (1:28)

web folder

```
import com.example.demo.service.OfferService;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.PathVariable;

import java.util.UUID;

@Controller
public class OfferController {

    private OfferService offerService;

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

    // @PreAuthorize("hasRole('ADMIN')") - only 1 user type can access it
    // @PreAuthorize("hasAnyRole('ADMIN', 'MODERATOR', 'TOZI_ONZI')") //several user types can
    // access it
    // @PreAuthorize("@offerService.isOwner(#principal.name, #uuid)") //isOwner here refers to
    // offerService - specific users can access it
    @PreAuthorize("isOwner(#uuid)") //isOwner here refers to OwnerSecurityExpressionRoot - more
    // logic which users to have access
    @DeleteMapping("/offers/{id}")
    public String deleteOffer(@PathVariable("id") UUID uuid) {
        offerService.deleteOfferById(uuid);

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

service folder

```
@Service
public class OfferService {
    public boolean isOwner(String userName, UUID offerId) {
        boolean isOwner = offerRepository
            .findById(offerId)
            .filter(o -> o.getSeller().getEmail().equals(userName))
            .isPresent();

        if (isOwner) {
            return true;
        }

        return userRepository.
            findByEmail(userName)
            .filter(this::isAdmin)
            .isPresent();
    }

    private boolean isAdmin(UserEntity user) {
        return user.getUserRoles()
            .stream()
            .anyMatch(r -> r.getUserRole() == UserRoleEnum.ADMIN);
```

```

    }

    public void deleteOfferById(UUID uuid) {
    }

}

```

config folder

```

import com.example.demo.service.OfferService;
import org.springframework.security.access.expression.SecurityExpressionRoot;
import org.springframework.security.access.expression.method.MethodSecurityExpressionOperations;
import org.springframework.security.core.Authentication;

import java.util.UUID;

public class OwnerSecurityExpressionRoot extends SecurityExpressionRoot implements
MethodSecurityExpressionOperations {

    private final Authentication authentication;
    private final OfferService offerService;
    private Object filterObject;
    private Object returnObject;

    public OwnerSecurityExpressionRoot(Authentication authentication, OfferService offerService)
    {
        super(authentication);
        this.authentication = authentication;
        this.offerService = offerService;
    }

    // This method is accessible in the @PreAuthorize annotation
    // When more complicated scenario for roles - admin can delete everything, but if we want
    // the user who created the article to be able to delete it also - this is the way
    public boolean isOwner(UUID id) {
        if (authentication.getPrincipal() == null) {
            return false; //Not Logged users cannot do anything
        }

        var userName = authentication.getName();

        return offerService.isOwner(userName, id);
    }

    @Override
    public void setFilterObject(Object filterObject) {
        this.filterObject = filterObject;
    }

    @Override
    public Object getFilterObject() {
        return filterObject;
    }

    @Override
    public void setReturnObject(Object returnObject) {

```

```

        this.returnObject = returnObject;
    }

    @Override
    public Object getReturnObject() {
        return returnObject;
    }

    @Override
    public Object getThis() {
        return this;
    }
}

-----
import com.example.demo.service.OfferService;
import org.aopalliance.intercept.MethodInvocation;
import
org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler;
import org.springframework.security.access.expression.method.MethodSecurityExpressionOperations;
import org.springframework.security.authentication.AuthenticationTrustResolverImpl;
import org.springframework.security.core.Authentication;

public class MobileleSecurityExpressionHandler extends DefaultMethodSecurityExpressionHandler {
    private final OfferService offerService;

    public MobileleSecurityExpressionHandler(OfferService offerService) {
        this.offerService = offerService;
    }

    @Override
    protected MethodSecurityExpressionOperations createSecurityExpressionRoot(Authentication authentication, MethodInvocation invocation) {
        OwnerSecurityExpressionRoot root = new OwnerSecurityExpressionRoot(authentication,
offerService);

        root.setPermissionEvaluator(getPermissionEvaluator());
        root.setTrustResolver(new AuthenticationTrustResolverImpl());
        root.setRoleHierarchy(getRoleHierarchy());

        return root;
    }
}

-----
import com.example.demo.service.OfferService;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.access.expression.method.MethodSecurityExpressionHandler;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import
org.springframework.security.config.annotation.method.configuration.GlobalMethodSecurityConfiguration;

@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class MobileleMethodSecurityConfig extends GlobalMethodSecurityConfiguration {

```

```

private final OfferService offerService;

public MobileleMethodSecurityConfig(OfferService offerService) {
    this.offerService = offerService;
}

@Override
protected MethodSecurityExpressionHandler createExpressionHandler() {
    return new MobileleSecurityExpressionHandler(offerService);
}
}

```

Name and password in application.yaml

```

spring:
  security:
    user:
      name: user
      password: password

```

13. (HATEOAS) Hypermedia As the Engine of Application State

13.1. What is HATEOAS

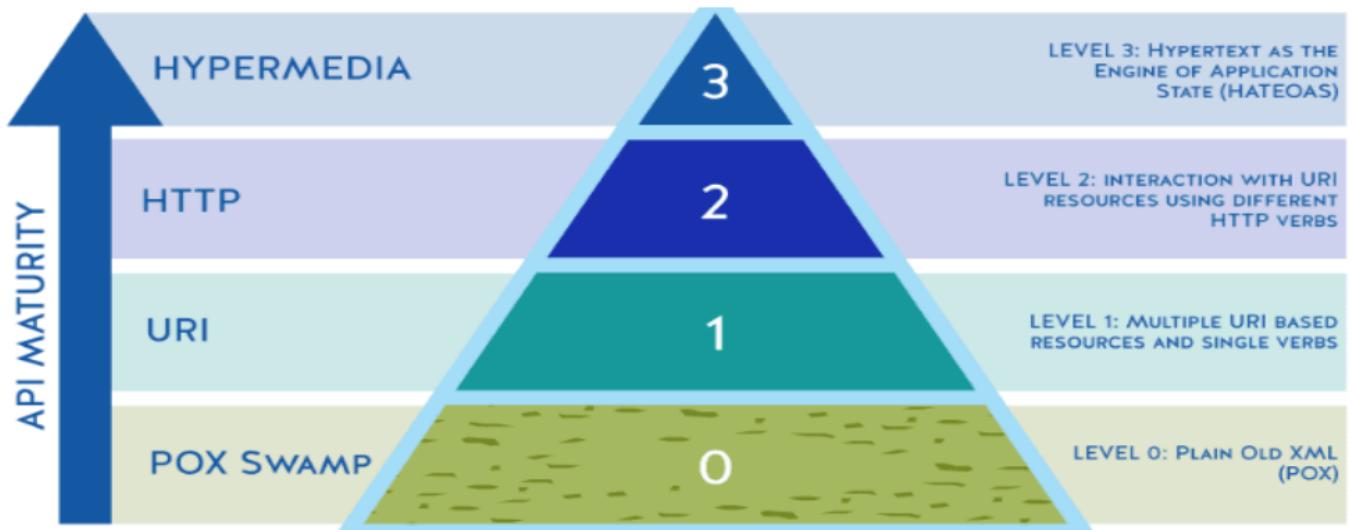
Hypermedia As the Engine of Application State

- **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 – **демек връщаме на клиенти като API response допълнителни данни в response body-то:**
 - **кое е следващото (следващият http ресурс)**
 - **или списък от следващи неща**
 - **или списък на всички възможни линкове в дадения сайт да речем**
- 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 HATEOAS в PayPal системата

<https://developer.paypal.com/api/rest/responses/>

13.2. 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

13.3. 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

The screenshot shows the Spring Initializr web application. On the left, there are sections for 'Project' (Maven Project selected), 'Language' (Java selected), and 'Spring Boot' (2.7.1 selected). On the right, under 'Dependencies', the 'H2 Database' dependency is selected. Other listed dependencies include 'Spring Web' (selected) and 'Spring Data JPA'. A red vertical bar highlights the 'H2 Database' section.

```
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>

The screenshot shows two side-by-side configurations for the H2 console. The left configuration is for 'Saved Settings: Generic H2 (Embedded)'. It includes fields for 'Setting Name' (Generic H2 (Embedded)), 'Driver Class' (org.h2.Driver), 'JDBC URL' (jdbc:h2:~/test), 'User Name' (sa), and 'Password'. The right configuration is for 'Saved Settings: Generic H2 (Embedded)'. It includes fields for 'Setting Name' (Generic H2 (Embedded)), 'Driver Class' (org.h2.Driver), 'JDBC URL' (jdbc:h2:mem:testdb), 'User Name' (sa), and 'Password'. Both configurations have 'Save' and 'Remove' buttons.

The screenshot shows the main interface of the H2 Console. On the left, there is a tree view of the database schema under 'jdbc:h2:mem:testdb'. The schema includes tables 'COURSES', 'ORDERS', 'STUDENTS', and 'INFORMATION_SCHEMA'. The 'COURSES' table has columns 'ID', 'ENABLED', 'NAME', 'PRICE', and 'Indexes'. The 'ORDERS' table has columns 'ID', 'COURSE_ID', and 'STUDENT_ID'. The 'STUDENTS' table has columns 'ID', 'AGE', 'NAME', and 'Indexes'. The 'INFORMATION_SCHEMA' table contains information about the database. On the right, there is a SQL editor window with the following content:

```
Run Run Selected Auto complete Clear SQL statement:
SELECT * FROM COURSES
```

Below the SQL editor, there is a section titled 'Important Commands' with the following table:

	Displays this Help Page
	Shows the Command History
	Ctrl+Enter Executes the current SQL statement
	Shift+Enter Executes the SQL statement defined by the text selection
	Ctrl+Space Auto complete
	Disconnects from the database

At the bottom, there is a section titled 'Sample SQL Script' with the following content:

Delete the table if it exists Create a new table with ID and NAME columns Add a new row Add another row Query the table Change data in a row Remove a row	<pre>DROP TABLE IF EXISTS TEST; CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255)); INSERT INTO TEST VALUES(1, 'Hello'); INSERT INTO TEST VALUES(2, 'World'); SELECT * FROM TEST ORDER BY ID; UPDATE TEST SET NAME='Hi' WHERE ID=1; DELETE FROM TEST WHERE ID=2;</pre>
---	---

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 ът 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

JSON	Raw Data	Headers
Save Copy Collapse All Expand All Filter JSON		
<pre> ▼ _embedded: ▼ studentDTOList: ▼ 0: id: 1 name: "Pesho" age: 33 ▼ _links: ▼ self: href: "http://localhost:8080/students/1" ▼ update: href: "http://localhost:8080/students/1" ▼ orders: href: "http://localhost:8080/students/1/orders" ▼ 1: id: 2 name: "Ana" age: 23 ▼ _links: ▼ self: href: "http://localhost:8080/students/2" ▼ update: href: "http://localhost:8080/students/2" ▼ orders: href: "http://localhost:8080/students/2/orders" </pre>		

```

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**

13.4. 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>

```



Edit Headers

http://localhost:8080/students

Go!

JSON Properties

```
{  
  "page": {  
    "size": 20,  
    "totalElements": 2,  
    "totalPages": 1,  
    "number": 0  
  }  
}
```

Links

Relation	Name	Title	HTTP	Doc
self				
profile			GET DELETE	POST PUT PATCH

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,  
        "_links": {  
          "self": {  
            "href": "http://localhost:8080/students/2"  
          },  
          "student": {  
            "href": "http://localhost:8080/students/2"  
          },  
          "orders": {  
            "href": "http://localhost:8080/students/2/orders"  
          }  
        }  
      }  
    ]  
  }  
}
```

14. Error Handling

Правим error handling-а само по един от начините. Не е добре да го смесваме.

14.1. 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**

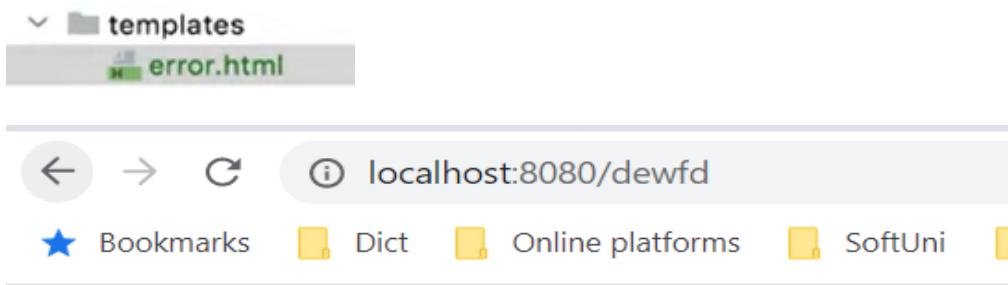
Error Handling in Spring

- 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

- **@ControllerAdvice** 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**



BasicErrorController

- Spring Boot maps **/error** to **BasicErrorController** which populates model with error attributes and then returns '**error**' as the view name

ErrorController Interface

- 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 **markir интерфейс**.

Т.е. да не използваме стандартния `spring /error` и да си направим наш:

```

server:
  error:
    whitelabel:
      enabled: false
    path: /error

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";
  }
}

```

14.2. @ResponseStatus - 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...

```

@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Product was not found.")
public class ProductNotFoundException extends RuntimeException {
  // Exception definition
}

```

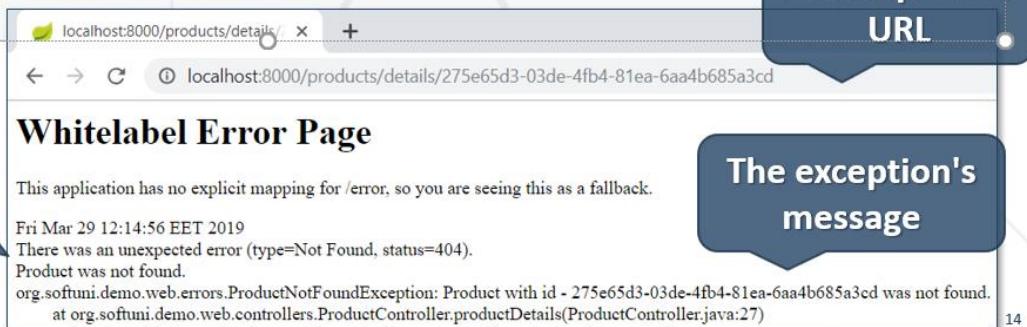
- And the controller action, throwing the exception

```
@GetMapping("/products/details/{id}")
public ModelAndView productDetails(@PathVariable String id, ModelAndView modelAndView) {
    Product product = this.productRepository.findById(id);

    if(product == null) throw new ProductNotFoundException(id);

    modelAndView.addObject("product", product);
    return this.view("product/details", modelAndView);
}
turn this.view("product/details", modelAndView);
```

The requested URL



14

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){
        //retrieve product from repository
        //productRepository.findById(productId).orElseThrow(new ProductNotFoundException());
        throw new ProductNotFoundException();
    }

    //Error 500
    @GetMapping("/products/{id}/error")
    public String boom(@PathVariable("id") String productid){
        //something wrong here
        //productRepository.findById(productId).orElseThrow(new ProductNotFoundException());
        throw new NullPointerException();
    }
}
```

```

    }
}

▼ └── error
    ├── 404.html 28.7.2022 г. 23:04, 398 B
    └── 500.html 28.7.2022 г. 23:05, 364 B

```

14.3. @ExceptionHandler - a Controller-Based Error Handling

Още един начин за handle-ване на exceptions

- 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_TRACE: 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){
        //retrieve product from repository
        //productRepository.findById(productId).orElseThrow(new ProductNotFoundException());
        throw new ObjectNotFoundException(productId);
    }

    //B ProductController2 е дефиниран @ExceptionHandler, и замова 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
        //замова задаваме тук http съмнуса 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
}
```

14.4. @ControllerAdvice Classes - Global Application Exception Handling -

Менаджиране на грешки от много на брой контролери

A. Global Exception Handling

- There is a way to achieve Global exception handling in Spring
 - This is done through the **@ControllerAdvice** annotation
- Any class annotated with **@ControllerAdvice** 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);
    }
}

```

B. 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

```

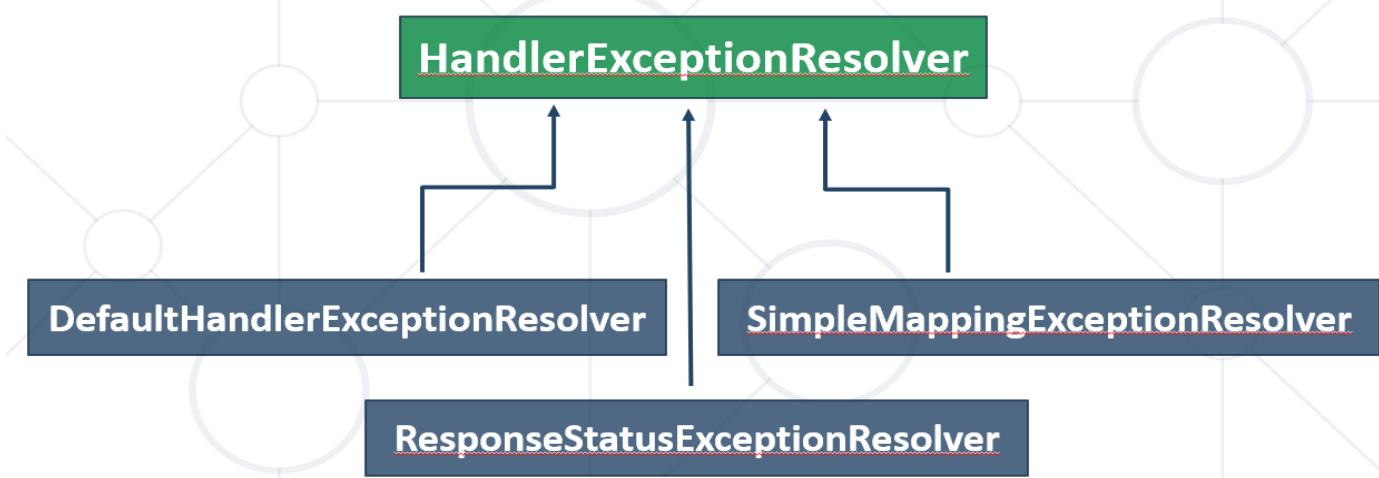
```

    ErrorInfo handleRESTErrors(HttpServletRequest req, DbException e) {
        return new ErrorInfo(req.getRequestURL(), ex);
    }
}

```

C. 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");

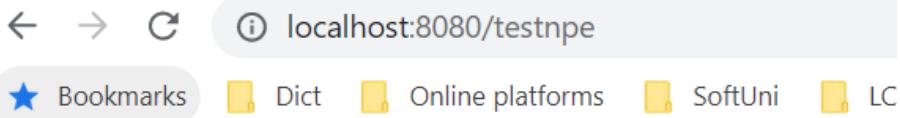
        resolver.setExceptionMappings(properties);

        return resolver;
    }
}

@Controller
public class TestController {

    @GetMapping("/testnpe")
    public String testNPE() {
        if (true) {
            throw new NullPointerException("npe");
        }
        return "npe";
    }
}

```



Ups, we threw NPE!

D. Using ResponseEntityExceptionHandler

```
import com.onlineacademy.onlineAcademy.exceptions.common.AccessDeniedException;
import com.onlineacademy.onlineAcademy.exceptions.common.ApiException;
import com.onlineacademy.onlineAcademy.exceptions.common.ConstraintValidationException;
import com.onlineacademy.onlineAcademy.exceptions.common.InternalServerErrorException;
import com.onlineacademy.onlineAcademy.exceptions.user.UserLoginException;
import com.onlineacademy.onlineAcademy.models.dto.response.ExceptionResponse;
import com.onlineacademy.onlineAcademy.utils.ApiExceptionParser;
import jakarta.validation.ConstraintViolationException;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.InternalAuthenticationServiceException;
import org.springframework.transaction.TransactionException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@ControllerAdvice
public class CustomExceptionHandler extends ResponseEntityExceptionHandler {
    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<ExceptionResponse> handleRuntimeExceptions(RuntimeException exception) {
        // Log data
        exception.printStackTrace();
        return handleApiExceptions(new InternalServerErrorException());
    }

    @ExceptionHandler(InternalAuthenticationServiceException.class)
    public ResponseEntity<ExceptionResponse>
    handleInternalExceptions(InternalAuthenticationServiceException exception) {
        Throwable cause = exception.getCause();

        if (cause instanceof ApiException) {
            return handleApiExceptions((ApiException) cause);
        }

        return handleRuntimeExceptions(exception);
    }

    @ExceptionHandler(ConstraintViolationException.class)
    public ResponseEntity<ExceptionResponse>
    handleConstraintValidationExceptions(ConstraintViolationException exception) {
        return handleApiExceptions(
            new ConstraintValidationException(
                exception.getConstraintViolations()
            )
        );
    }

    @ExceptionHandler(TransactionException.class)
    public ResponseEntity<ExceptionResponse> handleTransactionExceptions(TransactionException exception) {
```

```

    if (exception.getRootCause() instanceof ConstraintViolationException validationException) {
        return handleConstraintValidationExceptions(validationException);
    }

    return handleRuntimeExceptions(exception);
}

@ExceptionHandler(BadCredentialsException.class)
public ResponseEntity<ExceptionResponse> handleBadCredentialsExceptions() {
    return handleApiExceptions(new UserLoginException());
}

@ExceptionHandler(org.springframework.security.access.AccessDeniedException.class)
public ResponseEntity<ExceptionResponse> handleAccessDeniedExceptions() {
    return handleApiExceptions(new AccessDeniedException());
}

@ExceptionHandler(ApiException.class)
public ResponseEntity<ExceptionResponse> handleApiExceptions(ApiException exception) {
    ExceptionResponse apiException = ApiExceptionParser.parseException(exception);

    return ResponseEntity
        .status(apiException.getStatus())
        .body(apiException);
}
}

public class ApiExceptionParser {
    public static ExceptionResponse parseException(ApiException exception) {
        return ExceptionResponse
            .builder()
            .dateTime(LocalDateTime.now())
            .message(exception.getMessage())
            .status(exception.getStatus())
            .statusCode(exception.getStatusCode())
            .build();
    }
}
}

```

14.5. 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 them
- For Controller-specific exceptions, **@ExceptionHandler** methods should be added alongside the actions
- For all other exceptions, **@ExceptionHandler** methods in **@ControllerAdvice** classes should be implemented

15. Events in Spring

15.1. 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());
}

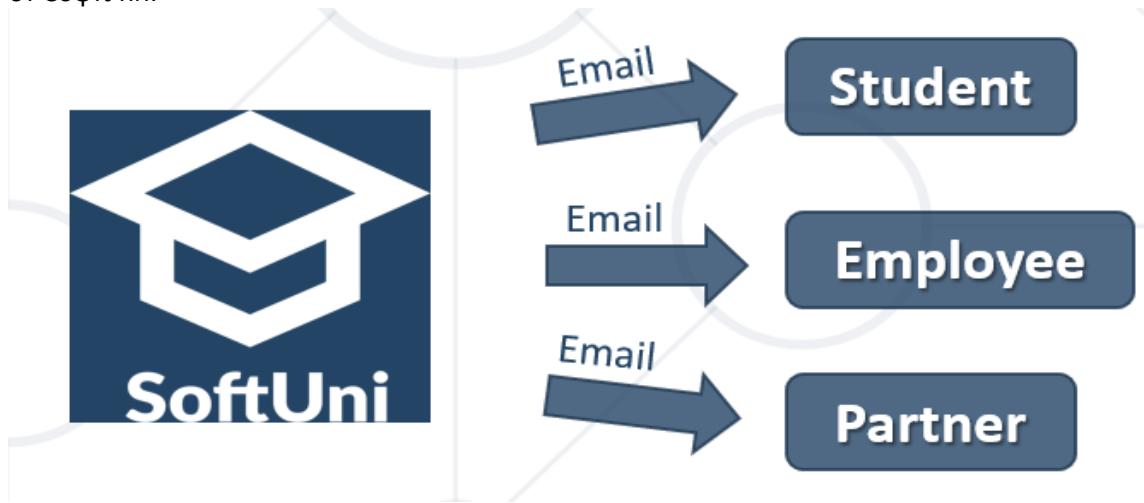
```

15.2. 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
- Realization of the so called **Publish and Subscribe mechanism**

Софтуни fire-ва event, реализирайки loose coupling – единия студент не знае дали/какво е получил другия студент от Софтуни.



Events in Spring

- Realization of the so called **Publish and Subscribe mechanism**
- 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 прихватем тези събития.

Spring Build-in Events – които се публикуват от ApplicationContext-a

- **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

15.3. Listening for Events

Listening for Events

- There are ways to listen 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

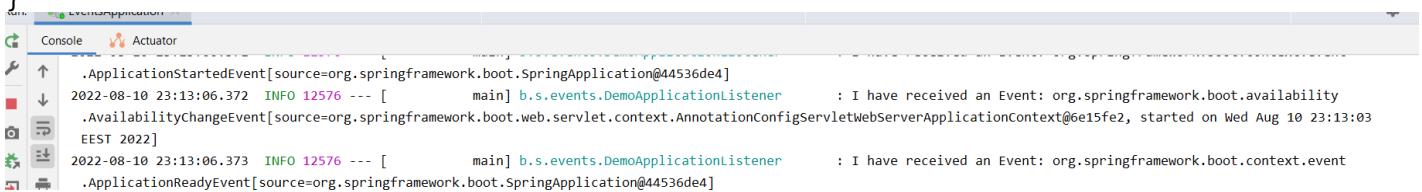
Demo using ApplicationListeners

- 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);
}
}

```

```

client=[0:0:0:0:0:0:0:1]; method=[GET]; servlet=[dispatcherServlet]; session=[null]; user=[null]; time=[29ms]; status=[OK]
2022-08-10 23:14:05.055 INFO 12576 --- [nio-8080-exec-1] b.s.events.DemoApplicationListener : I have received an Event: ServletRequestHandledEvent: url=[/error];
client=[0:0:0:0:0:0:0:1]; method=[GET]; servlet=[dispatcherServlet]; session=[null]; user=[null]; time=[75ms]; status=[OK]
2022-08-10 23:14:08.140 INFO 12576 --- [nio-8080-exec-3] b.s.events.DemoApplicationListener : I have received an Event: ServletRequestHandledEvent: url=[/test];
client=[0:0:0:0:0:0:0:1]; method=[GET]; servlet=[dispatcherServlet]; session=[null]; user=[null]; time=[27ms]; status=[OK]
2022-08-10 23:14:08.277 INFO 12576 --- [nio-8080-exec-4] b.s.events.DemoApplicationListener : I have received an Event: ServletRequestHandledEvent: url=[/favicon
.ico]; client=[0:0:0:0:0:0:0:1]; method=[GET]; servlet=[dispatcherServlet]; session=[null]; user=[null]; time=[6ms]; status=[OK]
2022-08-10 23:14:08.390 INFO 12576 --- [nio-8080-exec-4] b.s.events.DemoApplicationListener : I have received an Event: ServletRequestHandledEvent: url=[/error];
client=[0:0:0:0:0:0:0:1]; method=[GET]; servlet=[dispatcherServlet]; session=[null]; user=[null]; time=[112ms]; status=[OK]

```

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!");
}

```

15.4. 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);  
    }  
}
```

```

2   } //we publish the event
3   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

```

15.5. 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<ConcurrentMapCacheManager>**
- Create Bean CacheManager that returns new **ConcurrentMapCacheManager**

```
@Component
public class MyCacheCustomizer implements CacheManagerCustomizer<ConcurrentMapCacheManager> {
@Override
public void customize(ConcurrentMapCacheManager cacheM) {
    cacheM.setCacheNames(asList("students", "courses"));
}
}
```

16. Aspect Oriented Programming (AOP)

16.1. 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)!

16.2. 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

16.3. 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 – може 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.

16.4. 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="aspectLogging" 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("execution(public * *(..))")
private void trackStudentActions() {}
```

Pointcut Expressions – език за описание 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. *(..))")
```

16.5. 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 aroundAdvises(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; }
}
```

16.6. 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 {
}

```

17. Unit testing and integration testing

Не трябва да тестваме passwordencoder - 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 @PostConstruct annotated method beginInit()
- second, copy the real CLOUDINARY_SECRET in the application.yml in the test section // or other option is to set Environmental 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>
```

17.1. 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)

17.2. 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



17.3. 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);
    }
}
```

17.4. Integration Testing the Web Layer / the Controller

Testing Controller Examples

With Integration Tests!!!

@SpringBootTest

@AutoConfigureMockMvc

@Autowired

@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("$.size()", hasSize(2)))
            .andExpect(jsonPath("$.[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"));
}

```

17.5. 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 introducing 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.

black box testing – отиваме и се мъчим да хакнем дадена система (да направим security test) без да знаем как тя работи отвътре

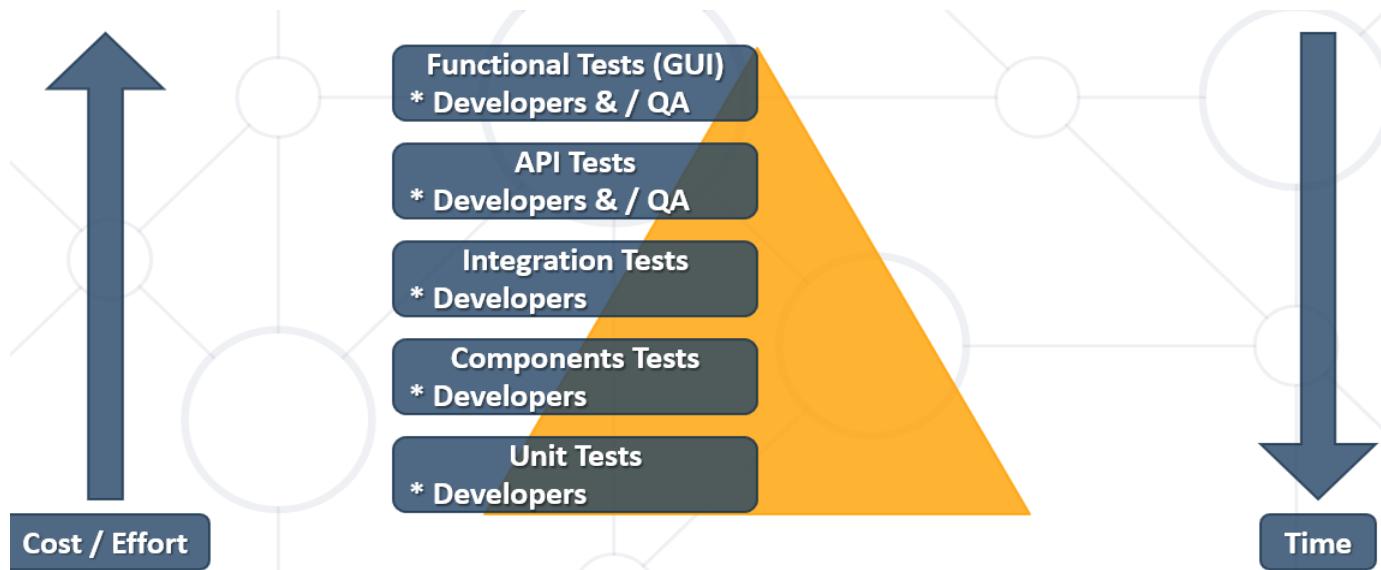
white box testing – когато имаме достъп до целия source код и можем да разгледаме и въз основа на придобитите от този source код знания , чак тогава да изпълним security test.

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



17.6. 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 добавявме 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 ProductItemBindingDTO 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 незапочнат 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 апликаций 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_cloudinary_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'");
}
}

//{@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);
}
}

```

```

@SpringBootTest
@AutoConfigureMockMvc
class RegistrationControllerTest {
    private static final String USER_CONTROLLER_PREFIX_REGISTER = "/users/register";
    @Autowired
    private MockMvc mockMvc;

```

```

@Test
void registerConfirmTestSuccessfull() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.post(USER_CONTROLLER_PREFIX_REGISTER)
        .param("username", "pesh")
        .param("email", "Pesho@mail.bg")
        .param("firstName", "Pesho")
        .param("lastName", "Peshovich")
        .param("password", "123456")
        .param("confirmPassword", "123456")
        .with(csrf())
        .contentType(MediaType.APPLICATION_FORM_URLENCODED))
        .andExpect(status().is3xxRedirection());
}

```

```

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-httpproblems"))
        .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
    //Задаваме 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));
}

```

ObjectMapper

```

1 usage
@Autowired
private MockMvc mockMvc;

1 usage
@MockBean
private CommentService commentService;

@Test
public void getComments_twoCommentsExist_commentsReturnedAsJsonAndStatusIsOk() {
    Long routeId = 1L;
    when(commentService.getAllCommentsForRoute(routeId)).thenReturn(List.of(
        new CommentDisplayView( id: 1L, authorName: "John Doe", message: "This is comment #1"),
        new CommentDisplayView( id: 2L, authorName: "Foo Bar", message: "This is comment #2")
    ));

    mockMvc.perform(get("/api/" + routeId + "/comments"))
}

```

```

    @Test
    @WithMockUser(username = "testUsername")
    public void createComment_sampleData_commentIsReturnedAsExpected() throws Exception {
        when(commentService.createComment(any())).thenAnswer(interaction -> {
            CommentCreationDto commentCreationDto = interaction.getArgument(0);
            return new CommentDisplayView(1L, commentCreationDto.getUsername(), commentCreationDto.getMessage());
        });
        CommentMessageDto commentMessageDto = new CommentMessageDto("This is comment #1");
        ObjectMapper objectMapper = new ObjectMapper();

        mockMvc.perform(post(urlTemplate: "/api/" + ROUTE_ID + "/comments")
                .content(objectMapper.writeValueAsString(commentMessageDto))
                .with(csrf())
                .contentType("application/json")
                .accept(...mediaTypes: "application/json"))
                .andExpect(status().is(201))
                .andExpect(jsonPath(expression: "$.message", is(value: "This is comment #1")))
                .andExpect(jsonPath(expression: "$.authorName", is(value: "testUsername")));
    }
}

```

17.7. 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!

Експешън, 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 различни 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 внето 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.

17.8. Using H2 console during test debugging in Spring

<https://hrrbrt.medium.com/using-h2-during-test-debugging-in-spring-f6a3db355e3a>

17.9. Coverage testing

И добавяш coverage от текущия тест към общия coverage - IntelliJ те пита като мине всеки тест клас за това.

По-хитър начин, за да не пускаме един по един: Направо върху тест папката кликаш, и после run with coverage.

18. Documentation - Swagger 3

OpenApi

Спецификации как да описваме нашите http rest api в json формат, в yml формат и т.н.

Използвало се – тази огромна документация с много редове

Swagger

- With the Swagger we can **simplifies** API development for users, teams, and enterprises
- Why we need documentations:
 - front-end and back-end components often **separate** a web application
 - usually, we expose APIs as a back-end component for the front-end component
- Reference documentation should **simultaneously describe** every change in the API

SpringDoc

- Spring Boot, using the **SpringDoc** implementation of the **Swagger 3 specification**.
- It's enough to add a single **springdoc-openapi-ui** dependency:

```
<dependency>
<groupId>org.springdoc</groupId>
<artifactId>springdoc-openapi-ui</artifactId>
<version>1.6.8</version>
</dependency>

dependencies {
    implementation 'org.springdoc:springdoc-openapi-ui'
}
```

<http://localhost:8080/swagger-ui.html>

- Go to <http://localhost:8080/swagger-ui/index.html> to test it – when openapi
- Go to <http://localhost:8080/webjars/swagger-ui/index.html> to test it – when openapi webflux

```
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Contact;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class OpenAPIConfig {
    @Bean
    public OpenAPI openAPI() {
        return new OpenAPI().
            info(new Info().
                title("Our open book API").
                version("1.0.0").
                contact(new Contact().name("Lachezar Balev").
                    email("lachezar.balev@gmail.com")).
                description("Our book API")));
    }
}

@CrossOrigin("*")
@RestController
@RequestMapping("/api/books")
public class BooksController {

    private final BookService bookService;

    @Autowired//Autowired not required on constructor
    public BooksController(BookService bookService) {
        this.bookService = bookService;
    }

    @GetMapping
```

```

public ResponseEntity<List<BookDTO>> getAllBooks() {
    return ResponseEntity.ok(bookService.getAllBooks());
}

@Tag(name = "Get book by ID", description = "Returns the book details by its id")
@Parameter(name = "id", description = "The ID of the book", required = true)
@ApiResponse(responseCode = "200", description = "If the book was retrieved successfully")
@ApiResponse(responseCode = "404", description = "If the book was not found")
@GetMapping("/{id}")
public ResponseEntity<BookDTO> getBookById(@PathVariable("id") Long bookId) {
    Optional<BookDTO> bookOpt = bookService.getBookById(bookId);
    if (bookOpt.isEmpty()) {
        return ResponseEntity.notFound().build();
    } else {
        return ResponseEntity.ok(bookOpt.get());
    }
}

@PostMapping
public ResponseEntity<BookDTO> createBook(@RequestBody BookDTO newBook, UriComponentsBuilder uriComponentsBuilder) {
    long newBookID = bookService.createBook(newBook);

    return ResponseEntity.created(uriComponentsBuilder.path("/api/books/{id}").build(newBookID)).build();
}

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

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

@PostMapping("/{id}")
public ResponseEntity<BookDTO> updateBook(@PathVariable("id") Long id, @RequestBody BookDTO bookDTO) {
    BookDTO updatedBookDTO = this.bookService.persistBook(id, bookDTO);

    if (updatedBookDTO == null) {
        return ResponseEntity.notFound().build();
    }

    return ResponseEntity.ok(updatedBookDTO);
}
}

```

Using Swagger UI example

The screenshot shows the Swagger UI interface for an OpenAPI definition. At the top, it says "OpenAPI definition v0 OAS3". Below that, there's a "Servers" section with a dropdown set to "http://localhost:8080 - Generated server url". The main area is titled "comment-rest-controller". It contains two API endpoints: "GET /api/{routeId}/comments" and "POST /api/{routeId}/comments". Underneath, there's a "Schemas" section with a "GrantedAuthority" schema listed.

От Swagger UI можем да си генерираме или сървър или клиент спрямо описаните

The screenshot shows the Swagger Editor interface. On the left, there's a code editor window displaying an OpenAPI specification. The code starts with "openapi: 3.0.1" and includes information about the API, contact details, and a server at "http://localhost:8080". To the right of the code editor is a "Generate Client" dropdown menu. The menu lists several client generation options: "csharp" (php), "csharp-dotnet2" (python), "dart" (r), "dynamic-html" (ruby), "go" (scala), and "html" (swift3). A red arrow points from the text above to the "Generate Client" dropdown, and another red arrow points to the "csharp" entry in the list.

Swagger UI

- **Swagger UI** allows anyone to **visualize** and **interact** with the API's resources without having any of the implementation logic in place.
- It's **automatically generated** from your OpenAPI (formerly known as Swagger) Specification, with the visual documentation making it easy for back-end implementation and client-side consumption.

19. Deployment

19.1. Deploy with Git

We can delete the test folder

Deploying On Heroku with Git – part A

- Download **Heroku CLI** - <https://devcenter.heroku.com/articles/heroku-cli>
- In the bash terminal, write the command

heroku login

- For creating a new Git repository
 - Go to the directory of the project
 - In the bash terminal, write

git init

- Create a new Git repository

git add .

git commit -m "some message"

git push

git push origin master

- Create a new Heroku project and bind it with the git repository

heroku create

```
Bo svilk@DESKTOP-SVILEN MINGW64 /c/Temp projects/Spring Advanced/14 Deployment, Hosting and Monitoring/books_heroku (master)
```

```
$ heroku create
```

```
Creating app... done, ⚡ calm-plains-37845
```

```
https://calm-plains-37845.herokuapp.com/ | https://git.heroku.com/calm-plains-37845.git
```

git remote -v

Вече мога да си push-вам и да си fetch-вам, но е празно локалното

```
Bookmark Structure
```

```
$ git remote -v
heroku https://git.heroku.com/calm-plains-37845.git (fetch)
heroku https://git.heroku.com/calm-plains-37845.git (push)
origin https://github.com/svilkata/books-heroku-demo.git (fetch)
origin https://github.com/svilkata/books-heroku-demo.git (push)
```

heroku git: remote -a <project-name>

- Add the PostgreSQL addon

heroku addons:create heroku-postgresql

```
$ heroku addons:create heroku-postgresql
Creating heroku-postgresql on ⬤ calm-plains-37845... free
Database has been created and is available
! This database is empty. If upgrading, you can transfer
! data from another database with pg:copy
Created postgresql-spherical-49894 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
```

19.2. Deployment

What is Deployment?

- **Deployment** means to push changes or updates from one environment to another



Where to Deploy?

- We can deploy **one project** onto **multiple websites**
- Some of the deployment websites
 - [Heroku](#)
 - [Amazon Web Services \(AWS\)](#)
 - [Google Cloud Platform](#)

Deploying On Heroku

- There are **3 ways** to deploy a project on Heroku
 - Using **Git** (Heroku Git, Heroku CLI)
 - Using **Github**
 - Using the **Container Registry** (Heroku **CLI** Command Line Interface)

Gradle or Maven – making the .jar files

Всичко в папка **build** или папка **target** не трябва и не се качва в Github!

МОЖЕ да изтрием тестовете при build-ване

Самият ти няма нужда да я изпълняваш

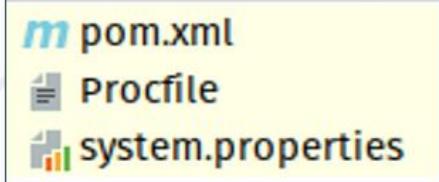
`mvn clean package` или `./mvnw -DskipTests clean dependency:list install`

Хероку ще го направи този jar, но трябва да посочиш правилния път на jar-а, който ще се появи в target директорията.

Procfile and system.properties

- Before running our project, we should add **3 important keys** to deploy the project
- Create 2 new files in our **project folder**
 - **Procfile**

- system.properties



System.properties

- system.properties
 - Holds **all of the system configuration properties** needed to run the project
 - By default, Heroku uses JDK Version 1.8
 - To specify specific version:

```
java.runtime.version={version}
java.runtime.version=17
java.runtime.version=11
```

Procfile

- Procfile
 - Holds the executed commands by the application on startup
 - Should include:

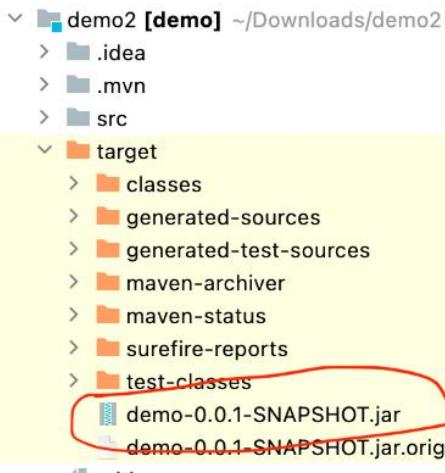
Където се намира jar-а

web: java -jar target/{name}-{version}.jar - за Maven
web: java -jar build/libs/{name}-{version}.jar - за Gradle

web: java -jar build/libs/books.jar

```
build.gradle
group = 'bg.softuni'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'
```

```
pom.xml
<version>1.0.0-SNAPSHOT</version>
<name>project_name</name>
```



```

2 <project xmlns="http://maven.apache.org/POM/
3   xsi:schemaLocation="http://maven.apache.org/ 
4     4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.7.2</version>
9     <relativePath/> 
10  </parent>
11  <groupId>com.example</groupId>
12  <artifactId>demo</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>demo</name>
```

application.properties or application.yml

application.properties

```

spring.datasource.url=${JDBC_DATABASE_URL}
spring.datasource.username=${JDBC_DATABASE_USERNAME}
spring.datasource.password=${JDBC_DATABASE_PASSWORD}
server.port=${PORT:8080}
spring.datasource.hikari.connection-timeout=30000
spring.datasource.hikari.maximum-pool-size=10
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

application.yml

```

spring:
  datasource:
    driverClassName: org.postgresql.Driver
    url: ${JDBC_DATABASE_URL}
    username: ${JDBC_DATABASE_USERNAME}
    password: ${JDBC_DATABASE_PASSWORD}
  jpa:
    #For MySQL
    #  database-platform: org.hibernate.dialect.MySQL8Dialect
    #For PostgreSQL
    database-platform: org.hibernate.dialect.PostgreSQLDialect
    hibernate:
      ddl-auto: update
      open-in-view: false
    properties:
      hibernate:
        format_sql: true
      defer-datasource-initialization: true

  server:
    port: ${PORT:8080}
```

build.gradle

```
dependencies {  
// runtimeOnly 'mysql:mysql-connector-java'  
    runtimeOnly 'org.postgresql:postgresql'  
}
```

Native MySQL queries will not work on PostgreSQL. Heroku offers free PostgreSQL database.

19.3. Deploying On Heroku with Git – part B

A - Managing config vars

Using the Heroku CLI (Command Line Runner)

- View current config var values

```
heroku login
```

```
$ heroku config
```

DATABASE_URL:

```
postgres://odqkbivsnfitu:fc64522599af9ea35a85eea06d34ac8ad87815ad3279a05df0eb3aca1a06f173@ec2-44-209-158-64.compute-1.amazonaws.com:5432/d1k9ara4igfkna
```

GITHUB_USERNAME: ivan

OTHER_VAR: student

```
$ heroku config:get GITHUB_USERNAME
```

```
ivan
```

```
$ heroku config:get CLOUDINARY_SECRET
```

```
mySecret
```

- Set a config var

```
$ heroku config:set CLOUDINARY_SECRET = mySecret
```

Adding config vars and restarting myapp... done, v12

SOME_SECRET: mySecret

Using the Heroku Dashboard

Config Variables

Config vars change the way your app behaves. In addition to creating your own, some addons come with their own.

Config Vars	
GITHUB_USERNAME	joesmith
OTHER_VAR	production
KEY	VALUE

Cancel Save

(-) (+)

Config Vars

Config vars change the way your app behaves.
In addition to creating your own, some add-ons come with their own.

Config Vars

[Hide Config Vars](#)

CLOUDINARY_SECRET	muhaha		
DATABASE_URL	postgres://fgtdqoxxvszze:7f9b214d9af5f62		
HEROKU_POSTGRESQL_SILVER_URL	postgres://vvhvsmncn1ful:a8e0dcf872bce93		
KEY	VALUE	Add	

B - Using the Heroku Dashboard

- You can manage your app's config vars programmatically with the [Heroku Platform API](#) using a simple HTTPS REST client and JSON data structures

C – push heroku

- Push the project to Heroku

master е локалната версия на heroku

git push heroku master



```
Terminal: Local X Git Bash X + ▾
heroku https://git.heroku.com/calm-plains-37845.git (fetch)
Enumerating objects: 53, done.
Counting objects: 100% (53/53), done.
Delta compression using up to 4 threads
Compressing objects: 100% (38/38), done.
Writing objects: 100% (53/53), 66.20 KiB | 3.89 MiB/s, done.
Total 53 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source...
remote:
remote: -----> Building on the Heroku-22 stack
remote: -----> Determining which buildpack to use for this app
remote: -----> Gradle app detected
remote: -----> Spring Boot detected
remote: -----> Installing OpenJDK 17... done
remote: -----> Building Gradle app...
remote: -----> executing ./gradlew build -x check
remote: Downloading https://services.gradle.org/distributions/gradle-7.4.1-bin.zip
remote: .....10%.....20%.....30%.....40%.....50%.....60%.....70%.....80%.....90%.....100%
remote: To honour the JVM settings for this build a single-use Daemon process will be forked. See https://docs.gradle.org/7.4.1/userguide/gradle\_daemon.html#sec:disabling\_the\_daemon
.
remote: Daemon will be stopped at the end of the build
remote: > Task :compileJava
remote:
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/calm-plains-37845.git
 * [new branch]      master -> master
```

- Change the ps:scale

heroku ps:scale web=1

```
$ heroku ps:scale web=1
```

```
Scaling dynos... done, now running web at 1:Free
```

- Check logs

heroku logs --tail

дава логовете на моя app и гледаме за грешки – дава някаква тъпа грешка за favicon

Structure

```
2022-09-11T21:46:35.545131+00:00 app[web.1]: at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
2022-09-11T21:46:35.545131+00:00 app[web.1]: at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
2022-09-11T21:46:35.545131+00:00 app[web.1]: at java.base/java.lang.reflect.Method.invoke(Method.java:568)
2022-09-11T21:51:21.371903+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/favicon.ico" host=calm-plains-37845.herokuapp.com request_id=0cdaf57d-8be6-43a6-9e95-d47558fa0225 fwd="87.227.215.196" dyno=connect service= status=503 bytes= protocol=https
2022-09-11T21:59:56.351065+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/" host=calm-plains-37845.herokuapp.com request_id=0acfecf-d75e-4c06-b989-c156f72666 cc fwd="87.227.215.196" dyno= connect= service= status=503 bytes= protocol=https
2022-09-11T21:59:56.646350+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/favicon.ico" host=calm-plains-37845.herokuapp.com request_id=a86b2d50-7a43-4667-958b-2d383f8acf6d fwd="87.227.215.196" dyno= connect= service= status=503 bytes= protocol=https
```

Bookmarks

Structure

19.4. Deploy with Github

- Create a registration in the Heroku website

The screenshot shows the Heroku website's landing page. At the top, there's a purple header bar with the Heroku logo, a search bar, and a 'New' button. Below the header, a large purple banner says 'Welcome to Heroku' and 'Now that your account has been set up, here's how to get started.' It features two main calls-to-action: 'Create a new app' (with a single user icon) and 'Create a team' (with a group icon). Below these are sections for language guides: 'Looking for help getting started with your language?' followed by icons for Node.js, Ruby, Java, PHP, Python, Go, Scala, and Clojure.

- Create a new App

Create New App

App name

 ✓

test-app-9871 is available

Choose a region

 ▼

➕ Add to pipeline...

Create app

- Add to the installed add-ons:



Personal ⋮ > ➕ test-app-9871

Overview

Resources

Deploy

Metrics

Activity

Access

Settings

Installed add-ons

\$0.00/month

[Configure Add-ons](#) ⊕

There are no add-ons for this app

You can add add-ons to this app and they will show here. [Learn more](#)

Dyno formation

\$0.00/month

[Configure Dynos](#) ⊕

This app has no process types yet

Add a Procfile to your app in order to define its process types. [Learn more](#)

Collaborator activity ⓘ

[Manage Access](#) ⊕

There is no recent activity on this app

Collaborator activity will be shown when there are recent deploys

Salesforce Platform

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Personal > mobilelele

Open app More

Overview Resources Deploy Metrics Activity Access Settings

Dynos

This app has no process types yet
Add a Procfile to your app in order to define its process types. [Learn more](#)

Add-ons

Find more add-ons

repo hero

Heroku Postgres

Heroku Connect

Apache Kafka on Heroku

Heroku Redis

VIDEOS

- Go to deploy tab, check "Github" and add your repository

repo-name

Search

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more](#).

Choose a branch to deploy

master

Deploy Branch

Heroku Git Use Heroku CLI

GitHub Connected ✓

Container Registry Use Heroku CLI

Connected to luchob/mobilele-heroku by luchob

Disconnect...

Releases in the [activity feed](#) link to GitHub to view commit diffs

Manual deploy

Deploy a GitHub branch

Deploy the current state of a branch to this app.

This will deploy the current state of the branch you specify below. [Learn more](#).

Choose a branch to deploy

Receive code from GitHub

Build main b7254fbb

```
[INFO] -----  
[INFO] Total time: 16.236 s  
[INFO] Finished at: 2022-09-18T09:04:15Z  
[INFO] -----  
----> Discovering process types  
      Procfile declares types -> web  
----> Compressing...  
      Done: 127.4M
```

Autoscroll with output

View logs

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Personal > mobilelele

GitHub luchob/mobilele-heroku

Overview Resources Deploy Metrics Activity Access Settings

Activity Feed > Build Log ID 6c3504

> Task :classes
> Task :bootJarMainClassName
> Task :bootJar

Open app More
View logs **View webhooks**
Run console Production check
Add to pipeline...
Restart all dynos

19.5. Deploying

За да се дисплайват страниците(за да работи изобщо) при деплоинат проект в Heroku
/For displaying the html pages in a Heroku deployed project

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    private final StatsInterceptor statsInterceptor;
    private LocaleChangeInterceptor localeChangeInterceptor;

    public WebConfig(StatsInterceptor statsInterceptor, LocaleChangeInterceptor localeChangeInterceptor) {
        this.statsInterceptor = statsInterceptor;
        this.localeChangeInterceptor = localeChangeInterceptor;
    }
}
```

```

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(statsInterceptor);
    registry.addInterceptor(localeChangeInterceptor);
}

import org.thymeleaf.templateresolver.ClassLoaderTemplateResolver;

@Bean
public ClassLoaderTemplateResolver myTemplateResolver(){
    ClassLoaderTemplateResolver configurer = new ClassLoaderTemplateResolver();
    configurer.setPrefix("templates/");
    configurer.setSuffix(".html");
    configurer.setTemplateMode(TemplateMode.HTML);
    configurer.setCharacterEncoding("UTF-8");
    configurer.setOrder(0);
    configurer.setCacheable(false);
    configurer.setCheckExistence(true);

    return configurer;
}
}

```

19.6. Actuator - Monitor and manage your application

Actuator

- Spring Boot includes a number of **additional features** to help you **monitor** and **manage** your application when you push it to production
- You can choose to manage and monitor your application by using **HTTP endpoints** or with **JMX**
- Auditing, health, and metrics gathering can also be **automatically applied** to your application

Actuator dependency

- The recommended way to enable the features is to add a dependency on the spring-boot-starter-actuator 'Starter'.

```

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>

```

implementation 'org.springframework.boot:spring-boot-starter-actuator'

build.gradle

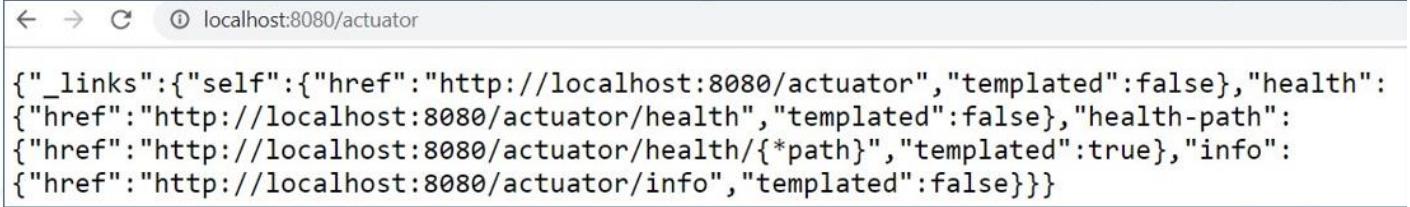
```
implementation 'org.springframework.boot:spring-boot-actuator-autoconfigure'
```

Actuator HTTP Endpoints

- HTTP Endpoints let you monitor and interact with your application
- Spring Boot includes a number of **built-in endpoints** and lets you add your own
- Each individual endpoint can be enabled or disabled and exposed

Actuator example

- For example, by default, the health endpoint is mapped to **/actuator**



A screenshot of a web browser window showing the JSON response for the `/actuator` endpoint. The URL in the address bar is `localhost:8080/actuator`. The response body contains the following JSON:

```
{"_links": {"self": {"href": "http://localhost:8080/actuator", "templated": false}, "health": {"href": "http://localhost:8080/actuator/health", "templated": false}, "health-path": {"href": "http://localhost:8080/actuator/health/{*path}", "templated": true}, "info": {"href": "http://localhost:8080/actuator/info", "templated": false}}}
```

Expose **all** actuator endpoints

- To expose **all** actuator **endpoints** you need to add in application.properties file:
`management.endpoints.web.exposure.include=*`

```
application.yaml
management:
  endpoints:
    web:
      exposure:
        include: "*"
```

← → ⌂ localhost:8080/actuator

Online platforms Soft Polezno casino kt jboxers-cl23 Predictor...

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```
_links:
  self:
    href: "http://localhost:8080/actuator"
    templated: false
  beans:
    href: "http://localhost:8080/actuator/beans"
    templated: false
  caches-cache:
    href: "http://localhost:8080/actuator/caches/{cache}"
    templated: true
  caches:
    href: "http://localhost:8080/actuator/caches"
    templated: false
  health:
    href: "http://localhost:8080/actuator/health"
    templated: false
  health-path:
    href: "http://localhost:8080/actuator/health/{*path}"
    templated: true
  info:
    href: "http://localhost:8080/actuator/info"
    templated: false
  conditions:
    href: "http://localhost:8080/actuator/conditions"
    templated: false
  configprops:
    href: "http://localhost:8080/actuator/configprops"
    templated: false
  configprops-prefix:
    href: "http://localhost:8080/actuator/configprops/{prefix}"
    templated: true
  env:
    href: "http://localhost:8080/actuator/env"
    templated: false
  env-toMatch:
    href: "http://localhost:8080/actuator/env/{toMatch}"
    templated: true
  loggers:
    href: "http://localhost:8080/actuator/loggers"
    templated: false
  loggers-name:
    href: "http://localhost:8080/actuator/loggers/{name}"
    templated: true
  heapdump:
    href: "http://localhost:8080/actuator/heapdump"
    templated: false
  threaddump:
    href: "http://localhost:8080/actuator/threaddump"
    templated: false
  scheduledtasks:
    href: "http://localhost:8080/actuator/scheduledtasks"
    templated: false
  mappings:
    href: "http://localhost:8080/actuator/mappings"
    templated: false
```

```

{
  "contexts": [
    "application"
  ],
  "beans": [
    "endpointCachingOperationInvokerAdvisor",
    "defaultServletHandlerMapping",
    "applicationTaskExecutor",
    "observationRegistryPostProcessor",
    "characterEncodingFilter",
    "forceAutoProxyCreatorToUseClassProxying",
    "healthEndpointGroups",
    "management.endpoint.health.org.springframework.boot.actuate.autoconfigure.health.HealthEndpointProperties",
    "webEndpointDiscoverer",
    "org.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguration",
    "org.springframework.boot.autoconfigure.web.DispatcherServletAutoConfiguration$DispatcherServletRegistrationConfiguration",
    "preserveErrorControllerTargetClassPostProcessor"
  ]
}

```

Enabling Endpoints

`application.properties`

- If you prefer all endpoints to be disabled
 - Set the **management.endpoints.enabled-by-default = false**
- Use individual endpoint enabled properties
 - For example, enable info endpoint

`management.endpoints.enabled-by-default=false`

`management.endpoint.info.enabled=true`

`application.yml`

```

management:
  endpoints:
    web:
      exposure:
        include: ["*"]
      base-path: "/admin"
    server:
      port: 8081

```

White listing – когато всичко е забранено, иказваме кое е разрешено само

Black listing – когато всичко е разрешено, иказваме кое е забранено само

Securing HTTP Endpoints

- You should take care to **secure HTTP endpoints** in the same way that you would any other sensitive URL
- If Spring Security is present, endpoints are **secured by default**
- Example of **custom security configuration** for HTTP endpoints
- EndpointRequest идва от актуатор

- Генерира линкове за бърза справка за разни/всички системни неща

```
@Configuration()
public class ActuatorSecurity extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.requestMatcher(EndpointRequest.toAnyEndpoint()).authorizeRequests((requests) ->
            requests.anyRequest().hasRole("ROLE_ADMIN"));
    }
}
```

ИЛИ ТАКА:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.
        authorizeRequests().
        // with this line we allow access to all static resources
        requestMatchers(PathRequest.toStaticResources().atCommonLocations()).per
        // allow actuator endpoints
        requestMatchers(EndpointRequest.toAnyEndpoint()).permitAll(). ✓
```

Implementing Custom Endpoints

- If you add a **@Bean** annotated with **@Endpoint**, any methods annotated with **@ReadOperation**, **@WriteOperation**, or **@DeleteOperation** are automatically exposed over JMX and, in a web application, over HTTP

```
@Component
@Endpoint(enableByDefault = true, id = "custom")
public class CustomEndpoint {
    @ReadOperation
    public String getMyEndpoint() {
        return "My custom endpoint";
    }
}
```

- If we want we can create Endpoints with **@RestControllerEndpoint** annotation

```
@Component
@RestControllerEndpoint( id="myRestEndpoint" )
public class MyRestEndpoint {

    @GetMapping("/test")
    @ResponseBody
    public String test(){
        return "My custom rest endpoint";
    }
}
```

Customizing properties

- Customizing the Management Endpoint Paths
 - `management.endpoints.web.base-path=/manage`
- Customizing the Management Server Port
 - `management.server.port=8081`
- Disabling HTTP Endpoints
 - `management.server.port=-1`

Vizualization Tools

- Using the Spring Boot Actuator give us a lot of **information** our application, but it's **not** very **user-friendly**
- Can be integrated with **Spring Boot Admin** for visualization, but it has its limitations and it's less popular
- Tools like **Micrometer**, **Prometheus** and **Grafana** are more commonly used for the monitoring and visualization and are language/framework-independent
 - These tools have their own set of data formats and converting the metrics data

19.7. Micrometer

Micrometer

- Solves the problem of being a **vendor-neutral data provider**
- Automatically **exposes /actuator/metrics data** into something your monitoring system can **understand**
- You need to include a vendor-specific micrometer dependency

Micrometer Dependency

- Micrometer is a separate open-sourced project and is not in the Spring ecosystem, so we have to explicitly add it as a dependency
- If using Prometheus, add its **specific** dependency

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

implementation 'io.micrometer:micrometer-registry-prometheus'

Micrometer Example

- After adding the micrometer dependency, we have a new endpoint - `/actuator/prometheus`
- The data is formatted in specific for Prometheus format

```

# HELP process_cpu_usage The "recent cpu usage" for the Java Virtual Machine process
# TYPE process_cpu_usage gauge
process_cpu_usage 2.560625718003065E-4
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds summary
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 1.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 0.6256954
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/"}, 2.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/"}, 0.0566226
# HELP http_server_requests_seconds_max
# TYPE http_server_requests_seconds_max gauge
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 0.0
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/"}, 0.0
# HELP jvm_classes_loaded_classes The number of classes that are currently loaded in the Java virtual machine
# TYPE jvm_classes_loaded_classes gauge
jvm_classes_loaded_classes 7273.0
# HELP jvm_threads_live_threads The current number of live threads including both daemon and non-daemon threads
# TYPE jvm_threads_live_threads gauge
jvm_threads_live_threads 28.0
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="G1 Survivor Space",} 3145728.0
jvm_memory_committed_bytes{area="heap",id="G1 Old Gen",} 2.62144E7
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 3.2555008E7
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-nmethods'",} 2555984.0
jvm_memory_committed_bytes{area="heap",id="G1 Eden Space",} 2.62144E7
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 4849664.0
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-profiled nmethods'",} 7077888.0
# HELP jvm_memory_max_bytes The maximum amount of memory in bytes that can be used for memory management
# TYPE jvm_memory_max_bytes gauge

```

19.8. Prometheus

Prometheus

- Time-series database that **stores** the **metric** data by pulling it (using a built-in data scraper) **periodically** over HTTP
- Intervals between pulls can be configured
- Has a simple user interface where we can **visualize/query** on all of the **collected metrics**
- To configure Prometheus more precisely we use the **prometheus.yaml** file

Download and Configure Prometheus

- You can download Prometheus from [here](https://prometheus.io/download/) - <https://prometheus.io/download/>
- Configure Prometheus with **prometheus.yaml** file

```

global:
  scrape_interval: 15s # By default, scrape targets every 15 seconds.
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'
    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 5s
    static_configs:
      - targets: [ 'localhost:9090' ]

```

Prometheus Dashboard

- After starting Prometheus, we can access it on <http://localhost:9090>

Prometheus Query Language – PromQL

- Prometheus provides a functional query language called **PromQL** (Prometheus Query Language)
- Let's the user **select** and **aggregate** time series data in real time
- Result of an expression can either be shown as a **graph**, viewed as **tabular data** in Prometheus' expression browser, or consumed by external systems via the **HTTP API**
- Return all time series with the metric `http_requests_total` and the **given job** and handler labels
`http_requests_total{job="apiserver", handler="/api/comments"}`

▪ Return a whole **range of time** for the same vector
`http_requests_total{job="apiserver", handler="/api/comments"}[5m]`

▪ Using **regular expressions**
`http_requests_total{job=~".*server"}`
`http_requests_total{status!~"4.."}`

20. Reactive programming

20.1. What is Reactive Programming

Reactive Programming

- Is a model of coding where **communication happens** through a **non-blocking stream of data**
- Makes code "reactive", **reacting to change** and **not being blocked**, such as performing operations that read/wait for responses from a database or file
- Can react to **events** as data **becomes available**
- Using it we can use **effective resources utilization** (CPU cores) for computing
- Its principles are based on the **Reactive Manifesto**
- It is build around **publisher-subscriber pattern** (observer pattern)

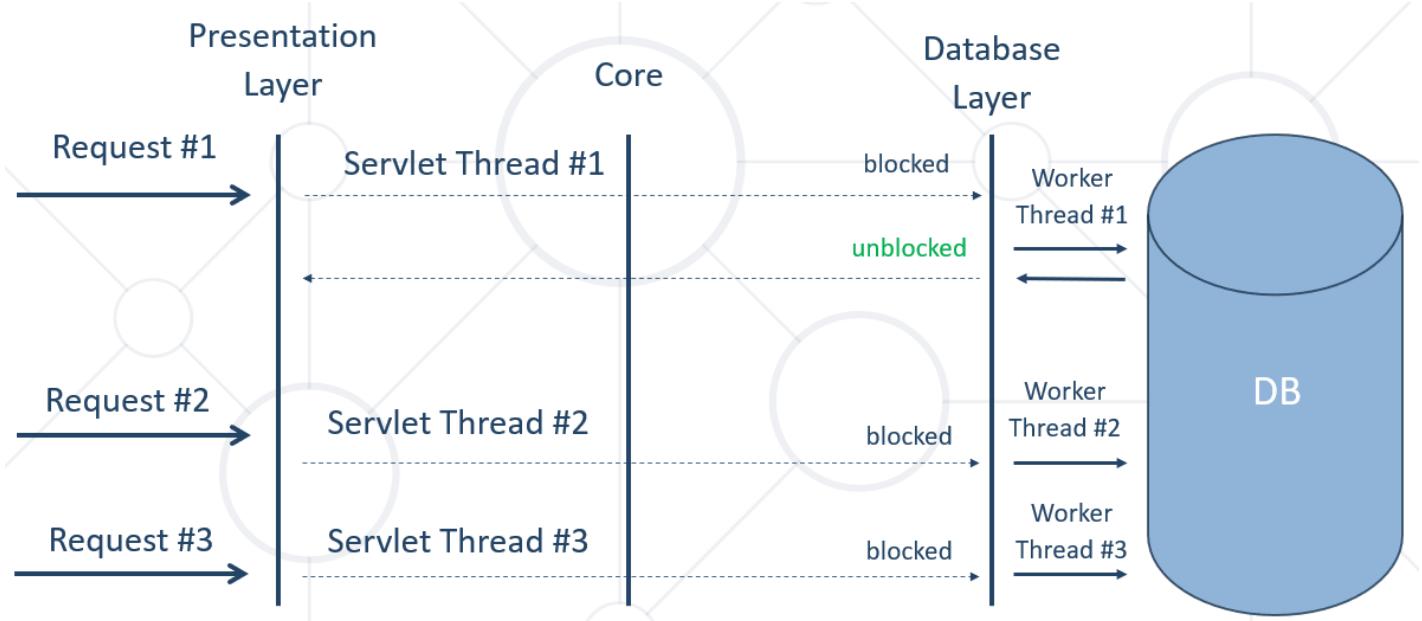
- In non-blocking code, the **Back pressure** becomes important to **control the rate of events** so that a fast producer does **not overwhelm** its destination

Една щайга ябълка в устата – ми не. Вземам една ябълка, като я изям подавам сигнал, че искам следваща ябълка. **Backpressure** е количеството информация, която нашия **consumer** желае да получи.

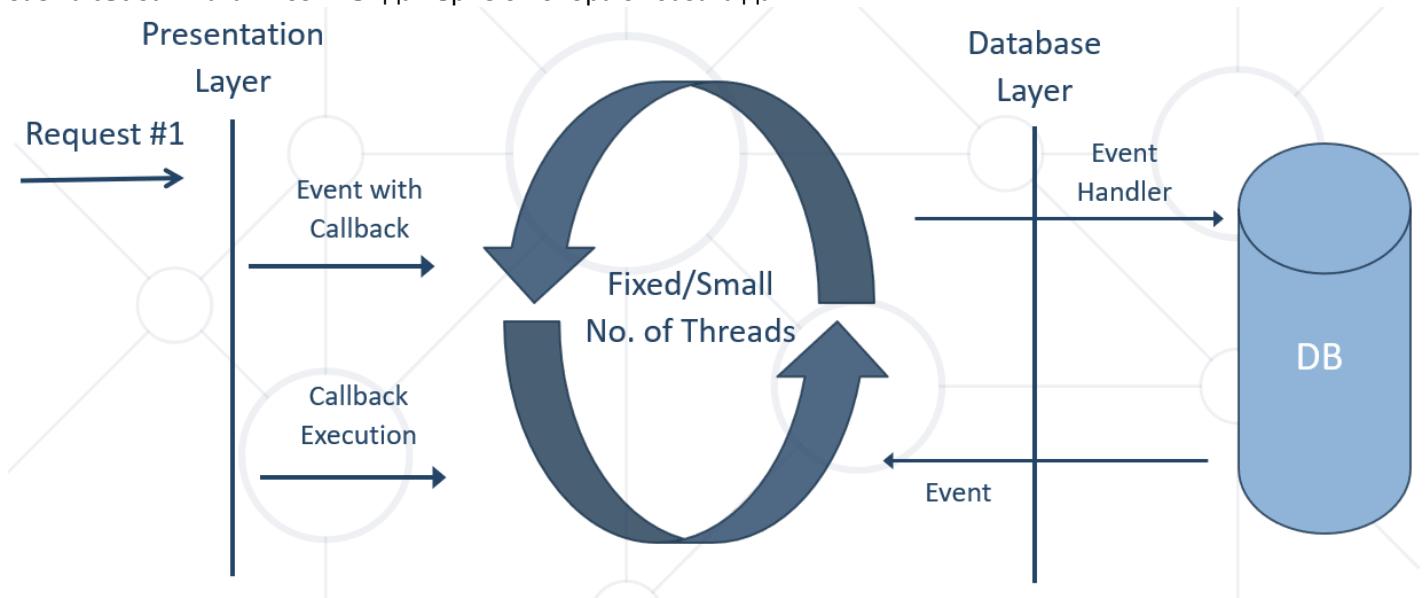
Traditional MVC is Blocking vs Non-blocking

Първа нишка при традиционния вариант се отбlocира навреме/веднага.

Но втора и трета нишка стоят блокирани при традиционния blocking вариант за доста време.



Изпращаме заявка към базата данни, след като нишката изпрати заявката, то спира да служи тук, и прави процесорно нещо друго. След като данните са взети от базата, то тази съща нишка или друга нишка отново се заема със заявката – този път да върне отговора от базата данни.



Reactive Streams and Reactor

- **Reactive Streams:** is a **specification** that defines how an API that implements and follows the Reactive Programming paradigm should work
- **Reactor:** is a Java implementation of the Reactive Streams specification
- **Spring WebFlux** is the "reaction" of Spring for this paradigm to use on web applications

Reactive Streams VS Java 8 Streams

- The core difference is that **Reactive** is a **push model**, whereas the **Java 8 Streams** are a **pull model**
 - In reactive events are pushed to the subscribers as they come in
- **Java 8 Streams** - pulling all the data and returning a result
- With **Reactive** we could have an **infinite stream** coming in from an external resource, with **multiple subscribers** attached

Reactive Stream APIs

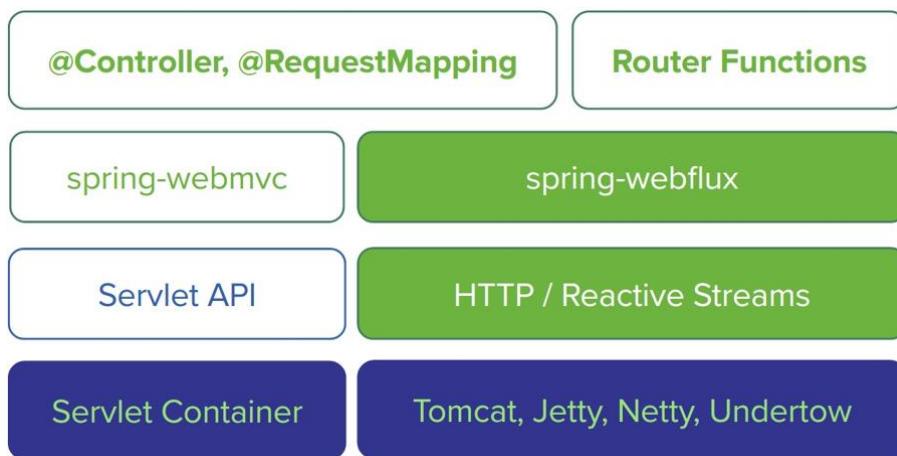
- As per **Java 9 and reactive specification** below are API we need to use for reactive implementation
 - **Publisher** - Emits a sequence of events to subscribers according to the demand received from its subscribers
 - **Subscriber(Consumer)** - Receives&Processes events emitted by a Publisher
 - **Subscription** - Defines a one-to-one relationship between a Publisher and a Subscriber
 - **Processors** - Represents a processing stage consisting of both a Publisher and a Subscriber (Consumer) and obeys the contracts of both = **a pipe from a Publisher and a Subscriber**

Spring 5 Reactive Building Blocks

- WebFlux
- Spring Data reactive library
- Reactive IO
- Nonblocking Servlet container
- Spring security reactive API

Spring 5 Reactive Building Blocks

- [Source - https://docs.spring.io/](https://docs.spring.io/)



20.2. Spring WebFlux

Spring WebFlux

- Spring WebFlux е **алтернатива** на стандартния Spring MVC (макар че могат да се използват и заедно)
- **Web framework** that brings support for the **reactive** programming model
- Implemented **using** the **Project Reactor**, and its publisher implementations — **Flux** and **Mono**, the library chosen by Spring
- WebFlux is **not a replacement for Spring MVC** they can actually complement each other, working together on the same solution

Spring WebFlux Dependencies

- Adding Spring WebFlux Dependency in pom.xml
- The Reactive Web dependency includes Spring WebFlux dependency

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

Mono and Flux

Mono – може да emit-ва 0 или 1 парче данни

Flux – може да emit-ва наведнъж повече от едно парче данни

- In WebFlux, the data returned from any operation is packed into a **reactive stream**
- There are **two types** that embody this approach and are the building blocks in WebFlux:
- **Mono** - is a stream which returns zero items or a single item (0..1)
- **Flux** - is a stream which returns zero or more items (0..n)

Working with Mono and Flux

- **Mono/Flux.just()**
 - The easiest way to emit an element is using the **just** method

```
Mono.just(1).subscribe(System.out::println);
Flux.just(1,2,3).subscribe(System.out::println);
```

- Mono/Flux can have more than one **subscribers**

```
Flux<Integer> flux = Flux.just(1,2,3);
flux.subscribe(s->System.out.println("Subscr One-"+ s));
flux.subscribe(s->System.out.println("Subscr Two-"+ s));
```

- Example of **two subscribers** with delay

```
Flux<Integer> flux = Flux.just(1, 2, 3);
flux
    .map(n -> ++n)
    .delayElements(Duration.ofMillis(500))
    .subscribe(System.out::println);      //2, 3, 4

flux
```

```
.delayElements(Duration.ofMillis(1000))
.subscribe(s -> System.out.println("Subscriber One-" + s)); //1, 2, 3
```

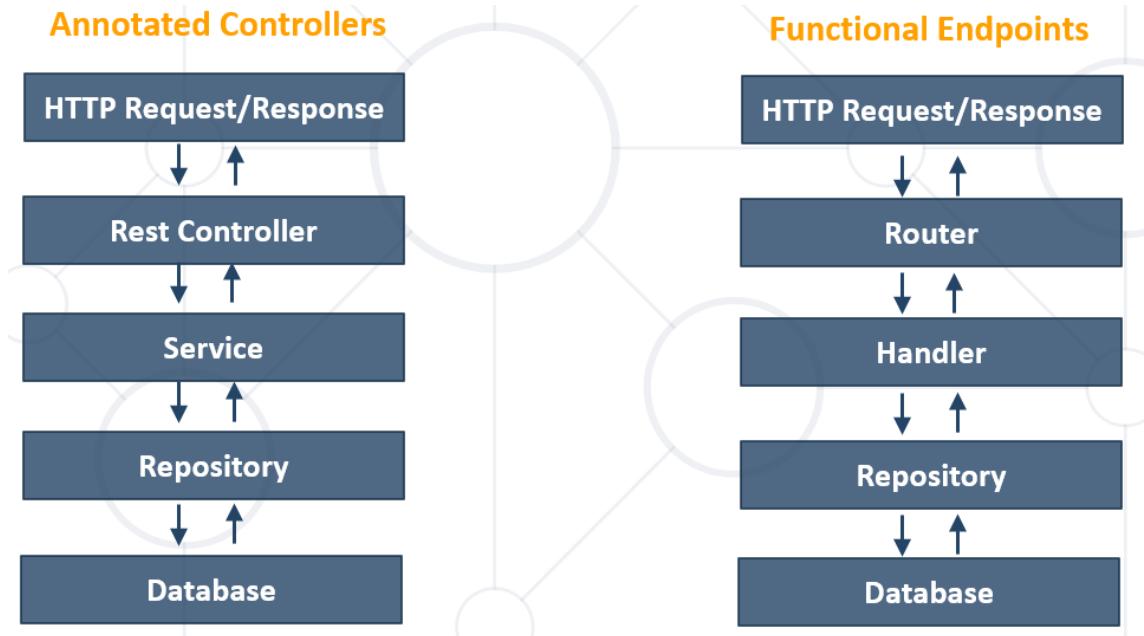
- The **subscribe** method could accept other parameters as well to handle the **error** and **completion** calls
- ```
Flux.just(1, 2, 3)
 .subscribe(
 i -> System.out.println("Received :: " + i),
 err -> System.out.println("Error :: " + err),
 () -> System.out.println("Successfully completed") //completion - моментът когато няма повече инфо в нашия flux
);
```

## 20.3. Programming Models

### Programming Models

- We can implement it in two ways:
  - Annotated Controllers**
    - using Spring MVC annotations with minimum modifications
    - The old applications can also be converted to reactive nature and can use its features and benefits
  - Functional Endpoints**
    - functional lambda style of programming

### Annotated Controllers VS Func Endpoints



### Annotated Controllers

- Similar to how we use controllers in **classic Spring MVC**

- To mark a class as a controller, we use the **@RestController** annotation on a class level.
- Having Spring WebFlux and the Reactor Core dependencies, in the class path, will let Spring know that the **@RestController** is in fact a reactive component and add support for **Mono** and **Flux**

## Annotated Controllers Example

- Annotated Controller Example

```
@RestController
@RequestMapping("/students")
public class StudentsControllers {
 //Inject studentsService in constructor

 @GetMapping("/all")
 public Flux<Students> findAll(){
 return studentsService.findAll();
 }
}
```

## Functional Endpoints

- Spring WebFlux includes **WebFlux.fn**, a lightweight functional programming model in which functions are used to **route** and **handle requests** and contracts are designed for immutability
- An HTTP request is handled with a **HandlerFunction** that takes **ServerRequest** and returns a delayed **ServerResponse** - **Mono<ServerResponse>**
- Incoming requests are routed to a handler function with a **RouterFunction** - takes **ServerRequest** and returns a delayed **HandlerFunction** - **Mono<HandlerFunction>**

## Functional Endpoints Example – Handler

- Example of Student Handler

```
public class StudentHandler {
 //...
 public Mono<ServerResponse> getStudent(ServerRequest request) {
 int studentId = Integer.valueOf(request.pathVariable("id"));
 return repository.getStudent(studentId)
 .flatMap(student -> ok().contentType(APPLICATION_JSON).bodyValue(student))
 .switchIfEmpty(ServerResponse.notFound().build());
 } //...
}
```

## Functional Endpoints Example – Router

- Example of Router Function

//... Inject PersonHandler **handler** in constructor

```
RouterFunction<ServerResponse> router = router()
 .GET("/student/{id}", accept(APPLICATION_JSON), handler::getStudent)
 .GET("/student ", accept(APPLICATION_JSON), handler::listStudents)
 .POST("/student ", handler::createStudent)
 .build();
```

## Spring WebFlux Configuration – за по-стари Spring версии

- The **@Configuration** and **@EnableWebFlux** annotations mark a class as a configuration class and Spring's bean management will register it
- To use or extend the existing WebFlux configuration API, you can implement **WebFluxConfigurer**

```
@Configuration
@EnableWebFlux
public class Configuration implements WebFluxConfigurer {...}
```

## 20.4. Demo – the 4 interfaces of reactive streams

```
import java.util.LinkedList;
import java.util.List;
import java.util.concurrent.Flow.Subscription;
import java.util.concurrent.Flow.Subscriber;

//I.e. this is the consumer
public class SimpleSubscriber<T> implements Subscriber<T> {
 private List<T> consumedElements = new LinkedList<>();
 private Subscription subscription;

 @Override
 public void onSubscribe(Subscription subscription) {
 this.subscription = subscription;
 subscription.request(1);
 }

 @Override
 public void onNext(T item) {
 consumedElements.add(item);

 //Applying backpressure - искам вече следващото
 subscription.request(1);
 }

 @Override
 public void onError(Throwable throwable) {
 throwable.printStackTrace();
 }

 @Override
 public void onComplete() {
 System.out.println("I'm complete!!!");
 consumedElements.forEach(System.out::println);
 System.out.println("-----");
 }

 public int getConsumedElementsCount(){
 return consumedElements.size();
 }

 public List<T> getConsumedElements() {
 return consumedElements;
 }
}
```

```

import java.util.concurrent.Flow.Processor;
import java.util.concurrent.Flow.Subscriber;
import java.util.concurrent.Flow.Subscription;
import java.util.concurrent.SubmissionPublisher;
import java.util.function.Function;

public class TransformationProcessor<T, R> implements Processor<T, R> {
 private final Function<T, R> transformation;
 private final SubmissionPublisher<R> submissionPublisher;
 private Subscription subscription;

 //нешо като pipe - ще получаваме от нашия publisher едни елементи, и ще ги превръщаме в
 //други елементи
 public TransformationProcessor(Function<T, R> transformation) {
 this.transformation = transformation;
 this.submissionPublisher = new SubmissionPublisher<R>();
 }

 //publisher
 @Override
 public void subscribe(Subscriber<? super R> subscriber) {
 submissionPublisher.subscribe(subscriber);
 }

 //publisher
 @Override
 public void onSubscribe(Subscription subscription) {
 this.subscription = subscription;
 subscription.request(1); //the backpressure
 }

 //Subscriber/producer
 @Override
 public void onNext(T item) {
 R transformed = transformation.apply(item);
 submissionPublisher.submit(transformed); //събmittваме надолу по веригата
 this.subscription.request(1); //the backpressure
 }

 //Subscriber/producer
 @Override
 public void onError(Throwable throwable) {
 throwable.printStackTrace();
 }

 //Subscriber/producer
 @Override
 public void onComplete() {
 System.out.println("Transformation is complete, closing down.");
 }
}

import java.util.List;
import java.util.concurrent.SubmissionPublisher;
import java.util.concurrent.TimeUnit;

```

```

import java.util.function.Function;

import static org.awaitility.Awaitility.await;
import static org.junit.jupiter.api.Assertions.*;

class ReactiveTests {

 @Test
 void testAllItemsConsumed() {
 //The producer!!!
 SubmissionPublisher<String> publisher = new SubmissionPublisher<>();

 SimpleSubscriber<String> mySubscriber = new SimpleSubscriber<>();

 //set new consumer for producer publisher
 publisher.subscribe(mySubscriber);

 assertEquals(1, publisher.getNumberOfSubscribers()); //Returns the number of current
subscribers.

 List<String> names = List.of("Anna", "John", "Pesho");
 names.forEach(item -> publisher.submit(item)); //submit = publishes the given items to
each current subscriber
 publisher.close();

 //Make it synchronous with the awaitility Library
 await()
 .atMost(1, TimeUnit.SECONDS)
 .until(
 () -> mySubscriber.getConsumedElementsCount() == names.size()
);
 assertEquals(3, mySubscriber.getConsumedElementsCount());
 }

 @Test
 public void testTransformation() {
 //arrange = given part
 Function<String, String> transFunc = String::toUpperCase;
 TransformationProcessor<String, String> transformationPipe = new
TransformationProcessor<>(transFunc);

 SubmissionPublisher<String> startPublisher = new SubmissionPublisher<>();
 SimpleSubscriber<String> finishSubscriber = new SimpleSubscriber<>();

 List<String> items = List.of("SenKo", "Pesho", "lilly"); //от едната страна на pipe-a
 List<String> expectedItems = List.of("SENKO", "PESHO", "LILLY"); //какво излизза от
другата страна на pipe-a

 //act = when
 startPublisher.subscribe(transformationPipe); //началната точка на pipe-a
 transformationPipe.subscribe(finishSubscriber); //крайната точка на Pipe-a

 items.forEach(item -> startPublisher.submit(item));
 startPublisher.close();
 }
}

```

```

 //Make it synchronous with the awaitility library
 await()
 .atMost(1, TimeUnit.SECONDS)
 .until(
 () -> expectedItems.equals(finishSubscriber.getConsumedElements())
);

 //assert
 assertEquals(expectedItems, finishSubscriber.getConsumedElements());
}
}

```

## 20.5. Demo – Mono and Flux

### Spring Reactive Web

```

dependencies {
 implementation 'org.springframework.boot:spring-boot-starter-webflux'
 testImplementation 'org.springframework.boot:spring-boot-starter-test'
 testImplementation 'io.projectreactor:reactor-test'
}

import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import java.util.List;
import java.util.Optional;
import java.util.Random;

public class FluxMonoExplained {
 private User user1 = new User().setFirstName("A").setLastName("B");
 private User user2 = new User().setFirstName("A1").setLastName("B1");

 public User getUser() {
 if (isAuthenticated()) {
 return user1;
 } else {
 return null;
 }
 }

 public Mono<User> getUserReactive() {
 if (isAuthenticated()) {
 return Mono.just(user1);
 } else {
 return Mono.empty();
 }
 }

 public Optional<User> getUserOpt() {
 if (isAuthenticated()) {
 return Optional.of(user1);
 } else {
 return Optional.empty();
 }
 }
}

```

```
public List<User> getAllUsers() {
 return List.of(user1, user2);
}

public Flux<User> getAllUsersReactive() {
 return Flux.just(user1, user2);
}

private boolean isAuthenticated() {
 return new Random().nextBoolean();
}

}

class User {
 private String firstName, lastName;

 public String getFirstName() {
 return firstName;
 }

 public User setFirstName(String firstName) {
 this.firstName = firstName;
 return this;
 }

 public String getLastName() {
 return lastName;
 }

 public User setLastName(String lastName) {
 this.lastName = lastName;
 return this;
 }
}

import org.junit.jupiter.api.Test;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import java.util.function.Consumer;
import java.util.stream.Collectors;

class WebfluxGeneralTest {
 @Test
 public void fluxToStream() {
 Flux<String> springProjectsFlux = Flux.just(getSpringProjects());

 springProjectsFlux
 .toStream()
 .map(String::toUpperCase)
 .forEach(System.out::println);
 }
}
```

```

@Test
public void subscribeToFlux() {
 Flux<String> springProjectsFlux = Flux.just(getSpringProjects());

 springProjectsFlux.subscribe(
 System.out::println
);
}

@Test
public void doOnEach() {
 Flux<String> springProjectsFlux = Flux.just(getSpringProjects());

 springProjectsFlux.doOnEach(
 signalConsumer -> {
 if (signalConsumer.isOnNext()) {
 System.out.println(signalConsumer.get());
 }
 }
)
 .subscribe();
}

@Test
public void mapAndFilter() {
 Flux<String> springProjectsFlux = Flux.just(getSpringProjects());

 springProjectsFlux
 .map(String::toUpperCase)
 .filter(s -> s.contains("REST"))
 .subscribe(System.out::println);
}

@Test
public void collect() {
 Flux<String> springProjectsFlux = Flux.just(getSpringProjects());

 springProjectsFlux
 .map(String::length)
 .collect(Collectors.summarizingInt(Integer::intValue)) //статистика от дължината
на всички заглавия
 .subscribe(System.out::println);

// IntSummaryStatistics{count=7, sum=85, min=10, average=12.142857, max=16}
}

@Test
public void subscribe() {
 Flux<String> springProjectsFlux = Flux.just(getSpringProjects());

 Consumer<String> onNextConsumer = System.out::println;
 Consumer<Throwable> onErrorConsumer = Throwable::printStackTrace;
 Runnable onDone = () -> System.out.println("We are done!");

 springProjectsFlux.subscribe(
 onNextConsumer,
 onErrorConsumer,
 onDone
);
}

```

```

 onDone
);
}

@Test
public void testOnError() {
 Flux<Integer> numbers = Flux.just("1", "two", "3")
 .map(Integer::parseInt);

 Consumer<Integer> onNextConsumer = System.out::println;
 Consumer<Throwable> onErrorConsumer = Throwable::printStackTrace;
 Runnable onDone = () -> System.out.println("We are done!");

 numbers.subscribe(
 onNextConsumer,
 onErrorConsumer, //throws error this time
 onDone
);
}

@Test
public void mono(){
 Mono.just("TEST").map(String::toUpperCase).subscribe(System.out::println);
}

private String[] getSpringProjects() {
 return new String[]{
 "Spring REST",
 "Spring DATA REST",
 "Spring Batch", //Java Batching
 "Spring MVC",
 "Spring Webflux",
 "Spring JMS", //Java Messaging Service
 "Spring Kafka"};
}
}

```

```

import org.junit.jupiter.api.Test;
import reactor.core.publisher.Flux;
import reactor.core.scheduler.Schedulers;

public class AsyncFlux {

 //синхронно
 @Test
 public void syncFlux() {
 MyInteger sum = new MyInteger(0);

 Flux.just(1, 2, 3, 4)
 .reduce(MyInteger::sum)
 .subscribe(sum::set);

 System.out.println(sum);
 }
}
```

```

// Output
// Summing 1 and 2
// Summing 3 and 3
// Summing 6 and 4
// MyInteger{initialValue=10}
}

//асинхронно
@Test
public void asyncFlux() {
 MyInteger sum = new MyInteger(0);

 Flux.just(1, 2, 3, 4)
 .subscribeOn(Schedulers.boundedElastic())
 .reduce(MyInteger::sum)
 .subscribe(sum::set);

 System.out.println(sum);

 //Output
 MyInteger{initialValue=0}
 Summing 1 and 2
 Summing 3 and 3
 Summing 6 and 4
}
}

class MyInteger {
 private Integer initialValue;

 public MyInteger(Integer initialValue) {
 this.initialValue = initialValue;
 }

 public static int sum(int a, int b) {
 System.out.println("Summing " + a + " and " + b);
 return a + b;
 }

 public void set(Integer aNumber) {
 this.initialValue = aNumber;
 }

 @Override
 public String toString() {
 return "MyInteger{" +
 "initialValue=" + initialValue +
 '}';
 }
}

```

## 20.6. Demo – webflux service

//При реактивното програмиране в Spring, за endpoint-и използваме бийнове вместо @Controller, @RequestMapping, и т.н.

See in my repo in GitHub

```
2022-10-19 23:37:43.891 INFO 14744 --- [main] b.s.w.WebfluxserviceApplication : No active profile set, falling back to 1 default profile: "default"
2022-10-19 23:37:45.571 INFO 14744 --- [main]
o.s.b.web.embedded.netty.NettyWebServer : Netty started on port 8080
2022-10-19 23:37:45.584 INFO 14744 --- [main] b.s.w.WebfluxserviceApplication : Started WebfluxserviceApplication in 2.201 seconds (JVM running for
```

```
public class ExchangeRate {
 private String fromCurrency;
 private String toCurrency;
 private Instant time;
 private BigDecimal rate;

 public ExchangeRate(String fromCurrency, String toCurrency, Instant time, BigDecimal rate) {
 this.fromCurrency = fromCurrency;
 this.toCurrency = toCurrency;
 this.time = time;
 this.rate = rate;
 }
}

@Service
public class ExchangeRateService {

 private List<ExchangeRate> exchangeRates = new ArrayList<>();

 public ExchangeRateService() {
 exchangeRates.add(new ExchangeRate("EUR", "BGN", Instant.now(), BigDecimal.valueOf(1.96)));
 exchangeRates.add(new ExchangeRate("USD", "BGN", Instant.now(), BigDecimal.valueOf(1.74)));
 exchangeRates.add(new ExchangeRate("GBN", "BGN", Instant.now(), BigDecimal.valueOf(2.16)));
 exchangeRates.add(new ExchangeRate("RSD", "BGN", Instant.now(), BigDecimal.valueOf(0.017)));
 }

 public Flux<ExchangeRate> getExchangeRateStream(int durationInterval) {
 Flux<ExchangeRate> ret = Flux.generate(() -> 0,
 (index, sink) -> {
 ExchangeRate updateRate = randomize(exchangeRates.get(index));
 sink.next(updateRate);
 System.out.println("Generated a new exchange rate...");
 return (++index) % exchangeRates.size();
 });

 if (durationInterval > 0) {
 return ret.delayElements(Duration.ofSeconds(durationInterval));
 } else {
 return ret;
 }
 }
}
```

```

 }
 }

@Component
public class ExchangeRateHandler {
 private final ExchangeRateService exchangeRateService;

 public ExchangeRateHandler(ExchangeRateService exchangeRateService) {
 this.exchangeRateService = exchangeRateService;
 }

 public Mono<ServerResponse> getExchangeRates(ServerRequest request) {
 int size = Integer.parseInt(request.queryParam("size").orElse("10"));

 //Вземане на моно от отложен server response
 return ok()
 .contentType(MediaType.APPLICATION_JSON)
 .body(this.exchangeRateService.getExchangeRateStream(0).take(size),
 ExchangeRate.class);
 }

 public Mono<ServerResponse> streamExchangeRates(ServerRequest request) {
 int size = Integer.parseInt(request.queryParam("size").orElse("10"));

 //Вземане на моно от отложен server response
 return ok()
 .contentType(MediaType.APPLICATION_STREAM_JSON)
 .body(this.exchangeRateService.getExchangeRateStream(1).take(size),
 ExchangeRate.class);
 }
}

//При реактивното програмиране в Spring, за endpoint-и използваме бийнове вместо @Controller,
//@RequestMapping, и т.н.
@Configuration
public class ExchangeRateRouter {

 @Bean
 public RouterFunction<ServerResponse> route(ExchangeRateHandler exchangeRateHandler) {
 return RouterFunctions
 .route(GET("/rates").and(accept(MediaType.APPLICATION_JSON)),
exchangeRateHandler::getExchangeRates)
 .andRoute(GET("/rates").and(accept(MediaType.APPLICATION_STREAM_JSON)), sr ->
exchangeRateHandler.streamExchangeRates(sr));
 }
}

```

## 20.7. Demo – webflux client with Reactive MongoDB driver

//При реактивното програмиране в Spring, за endpoint-и използваме бийнове вместо @Controller,  
//@RequestMapping, и т.н.

**Project**

- Gradle Project
- Maven Project

**Language**

- Java
- Kotlin
- Groovy

**Spring Boot**

- 3.0.0 (SNAPSHOT)
- 3.0.0 (M5)
- 2.7.5 (SNAPSHOT)
- 2.7.4
- 2.6.13 (SNAPSHOT)
- 2.6.12

**Project Metadata**

**Dependencies**

**ADD DEPENDENCIES... CTRL + B**

---

**Spring Reactive Web** WEB

Build reactive web applications with Spring WebFlux and Netty.

**Spring Data Reactive MongoDB** NOSQL

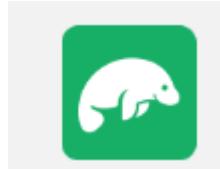
Provides asynchronous stream processing with non-blocking back pressure for MongoDB.

```
docker-compose.yaml
```

```
version: 3.1
services:
 mongo:
 image: mongo
 restart: always
 ports:
 - 27017:27017
```

в терминала команда docker-compose up

Mongo graphic user interface GUI  
Robo 3T is now Studio 3T Free



```
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Configuration
@EnableConfigurationProperties
@ConfigurationProperties("softuni.webflux.client")
public class ClientConfig {
 private String schema;
 private String host;
 private String port;
}

application.yml
softuni:
 webflux:
 client:
 schema: http
 host: localhost
 port: 8000

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
```

```

import java.math.BigDecimal;
import java.time.Instant;

{@Document //като @Entity анотацията за релационните база данни, но за MongoDB
public class ExchangeRate {
 @Id
 private String id;

 private String fromCurrency;
 private String toCurrency;
 private Instant time;
 private BigDecimal rate;
}

@Component
public class ExchangeRateClient {
 private final String SERVICE_URL;
 private final String API_PATH = "/rates";

 public ExchangeRateClient(ClientConfig clientConfig) {
 SERVICE_URL = clientConfig.getSchema() + "://" + clientConfig.getHost() + ":" +
clientConfig.getPort();
 }

 public Flux<ExchangeRate> getRateStream(){
 return WebClient.builder()
 .baseUrl(SERVICE_URL)
 .build()
 .get()
 .uri(API_PATH)
 .accept(MediaType.APPLICATION_STREAM_JSON)
 .retrieve()
 .bodyToFlux(ExchangeRate.class);
 }
}

@Repository
public interface ExchangeRateRepository extends ReactiveMongoRepository<ExchangeRate, String> {

}

@Service
public class ExchangeRateInit implements ApplicationListener<ContextRefreshedEvent> {

 private final ExchangeRateClient client;
 private final ExchangeRateRepository repository;

 public ExchangeRateInit(ExchangeRateClient client, ExchangeRateRepository repository) {
 this.client = client;
 this.repository = repository;
 }

 @Override

```

```

public void onApplicationEvent(ContextRefreshedEvent event) {
 client.getRateStream()
 .subscribe(exchangeRate -> {
 Mono<ExchangeRate> exchangeRateMono = repository.save(exchangeRate);
 exchangeRateMono.subscribe(er -> System.out.println("Saved " + er));
 });
}
}

```

## 26. 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

.....

```

<properties>
 <org.mapstruct.version>1.5.3.Final</org.mapstruct.version>
</properties>

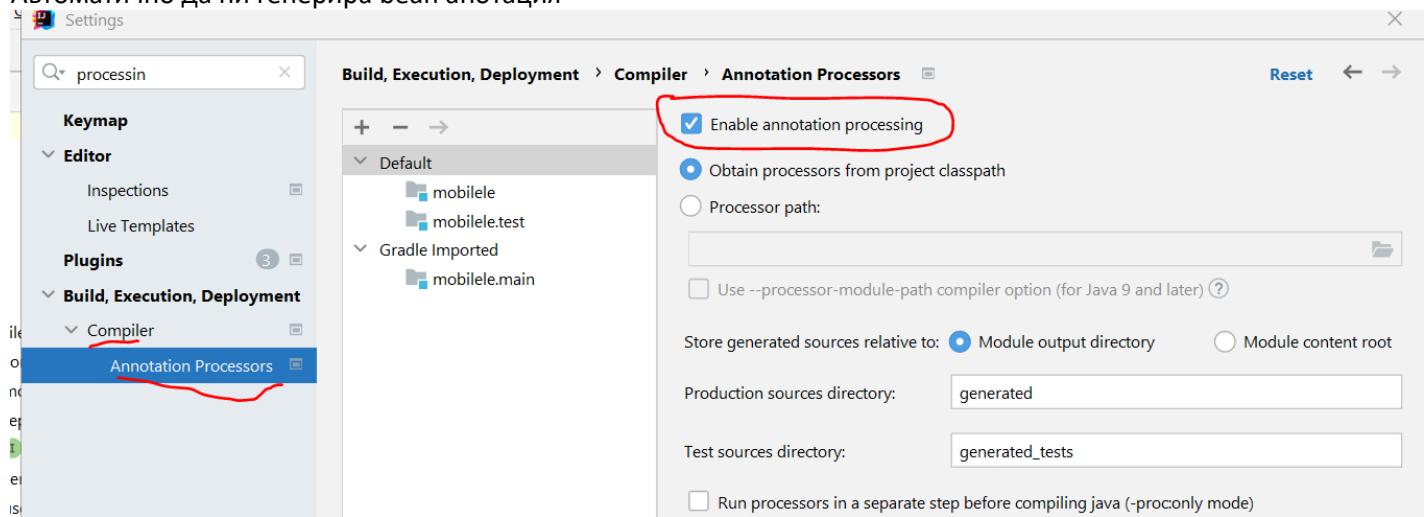
...
<dependencies>
 <dependency>
 <groupId>org.mapstruct</groupId>
 <artifactId>mapstruct</artifactId>
 <version>${org.mapstruct.version}</version>
 </dependency>
</dependencies>
...
<build>
 <plugins>
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</artifactId>
 <version>3.8.1</version>
 <configuration>
 <source>1.8</source><!-- depending on your project -->
 <target>1.8</target> <!-- depending on your project -->
 <annotationProcessorPaths>
 <path>
 <groupId>org.mapstruct</groupId>
 <artifactId>mapstruct-processor</artifactId>
 <version>${org.mapstruct.version}</version>
 </path>
 <!-- other annotation processors -->
 </annotationProcessorPaths>
 </configuration>
 </plugin>
 </plugins>
</build>

```

```
</plugin>
</plugins>
</build>
```

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 експоузата като Spring bean
public interface UserMapper {

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

}
```

Даваме Build project,  
и от този бутон

```

8 @Mapper(componentModel = "spring")
9 public interface UserMapper {
10 Navigate to autowired candidates
11 Navigate to the Spring bean declaration(s)
12 * UserEntity userDtoToUserEntity(UserRegisterDto registerDto);
13
14 }
15

```

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"

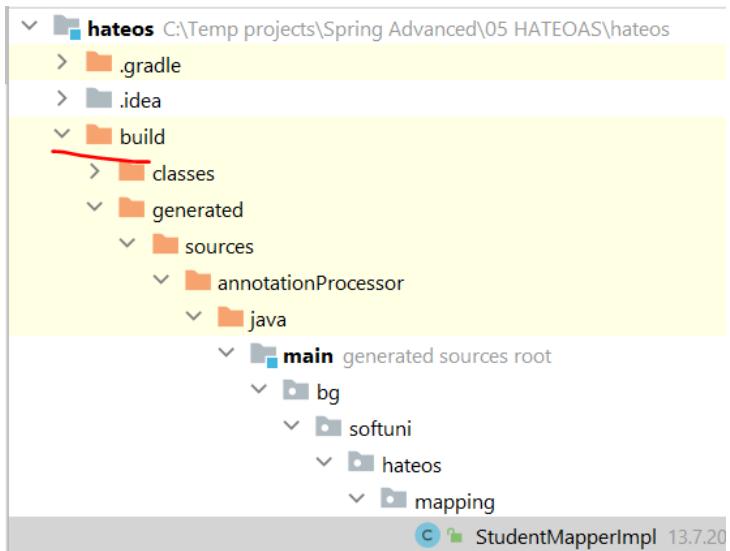
```

@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);
```

## 27. Pagination and Search with Specification and CriteriaBuilder at the server-side rendering

```
public interface JpaSpecificationExecutor<T> {
List<T> findAll(@Nullable Specification<T> spec);
Page<T> findAll(@Nullable Specification<T> spec, Pageable pageable);
List<T> findAll(@Nullable Specification<T> spec, Sort sort);
```

We can not implement specification in a @Query() JPQL, so we need to use only through the itemRepository.findAll - **not sure actually**

```
@Repository
public interface AllItemsRepository extends
PagingAndSortingRepository<ItemEntity, Long>,
JpaSpecificationExecutor<ItemEntity> {
 Page<ItemEntity> findAllByType(String type, Pageable pageable);
}
}
```

```

import bg.softuni.computerStore.model.binding.product.SearchProductItemDTO;
import bg.softuni.computerStore.model.entity.products.ItemEntity;
import org.springframework.data.jpa.domain.Specification;

import javax.persistence.criteria.*;

public class ProductItemSpecification implements Specification<ItemEntity> {
 private final SearchProductItemDTO searchProductItemDTO;
 private final String type;

 public ProductItemSpecification(SearchProductItemDTO searchProductItemDTO, String type) {
 this.searchProductItemDTO = searchProductItemDTO;
 this.type = type;
 }

 @Override
 public Predicate toPredicate(Root<ItemEntity> root,
 CriteriaQuery<?> query,
 CriteriaBuilder cb) {
 Predicate predicate = cb.conjunction();

 predicate.getExpressions().add(cb.equal(root.get("type"), type));

 if (searchProductItemDTO.getModel() != null &&
!searchProductItemDTO.getModel().isBlank()) {
 Path<Object> model = root.get("model");
 predicate.getExpressions().add(
 //!!!!!! when we have two relationally connected tables
// cb.and(cb.equal(root.join("model").get("name"),
// searchProductItemDTO.getModel())));
 //when all fields are from the same table ItemEntity:::: the Like works case
insensitive
 cb.and(cb.like(root.get("model").as(String.class), "%" + searchProductItemDTO.getModel() + "%"))
);
 }

 if (searchProductItemDTO.getMinPrice() != null) {
 predicate.getExpressions().add(
 cb.and(cb.greaterThanOrEqualTo(root.get("sellingPrice"),
 searchProductItemDTO.getMinPrice())))
);
 }

 if (searchProductItemDTO.getMaxPrice() != null) {
 predicate.getExpressions().add(
 cb.and(cb.lessThanOrEqualTo(root.get("sellingPrice"),
 searchProductItemDTO.getMaxPrice())))
);
 }

 return predicate;
 }
}

```

```

@Service
public class ComputerService implements InitializableProductService {
 private final AllItemsRepository allItemsRepository;
 private final StructMapper structMapper;
//Simpler option for pagination only
public Page<ComputerViewGeneralModel> getAllComputersPageable(Pageable pageable) {
 Page<ComputerViewGeneralModel> allComputers = this.allItemsRepository
 .findAllByType("computer", pageable)
 .map(comp -> this.structMapper
 .computerEntityToComputerSalesViewGeneralModel((ComputerEntity) comp));
 return allComputers;
}

//Complicated use
public Page<ComputerViewGeneralModel> getAllComputersPageableAndSearched(
 Pageable pageable, SearchProductItemDTO searchProductItemDTO, String type) {

 Page<ComputerViewGeneralModel> allComputers = this.allItemsRepository
 .findAll(new ProductItemSpecification(searchProductItemDTO, type), pageable)
 .map(comp -> this.structMapper
 .computerEntityToComputerSalesViewGeneralModel((ComputerEntity) comp));

 return allComputers;
}

@Controller
@RequestMapping("/items/all")
public class ViewItemsController {
 private final ComputerService computerService;

 @GetMapping("/computer")
 public String viewAllComputers(Model model,
 @Valid SearchProductItemDTO searchProductItemDTO,
 @PageableDefault(page = 0,
 size = 3,
 sort = "sellingPrice",
 direction = Sort.Direction.ASC) Pageable pageable,
 RedirectAttributes redirectAttributes) {

 if (!model.containsAttribute("searchProductItemDTO")) {
 model.addAttribute("searchProductItemDTO", searchProductItemDTO);
 }

 Page<ComputerViewGeneralModel> computers = this.computerService
 .getAllComputersPageableAndSearched(pageable, searchProductItemDTO, "computer");
 model.addAttribute("computers", computers);

 redirectAttributes.addFlashAttribute("searchProductItemDTO", searchProductItemDTO);

 return "/viewItems/all-computers";
 }
}

```

```
<main>
 <div class="container-fluid">
 <div class="container">
 <h2 class="text-center text-white">Search for offers</h2>
 <form
 th:method="GET"
 th:action="@{/items/all/computer}"
 th:object="${searchProductItemDTO}"
 class="form-inline"
 style="justify-content: center; margin-top: 50px;">
 <div style="position: relative">
 <input
 th:field="*{model}"
 th:errorclass="is-invalid"
 class="form-control mr-sm-2"
 style="width: 280px;"
 type="search"
 placeholder="Model name case Insensitive..."
 aria-label="Search"
 id="model"
 />
 <input
 th:field="*{minPrice}"
 th:errorclass="is-invalid"
 class="form-control mr-sm-2"
 style="width: 280px;"
 type="search"
 placeholder="Min price..."
 aria-label="Search"
 id="minPrice"
 />
 <input
 th:field="*{maxPrice}"
 th:errorclass="is-invalid"
 class="form-control mr-sm-2"
 style="width: 280px;"
 type="search"
 placeholder="Max price..."
 aria-label="Search"
 id="maxPrice"
 />
 <!--
 <!--
 <!--
 <!--
 align: center;-->
 <!--
 >-->
 Try writing something this time.-->
 </small>-->
 </div>
 <button class="btn btn-outline-info my-2 my-sm-0" type="submit">Search</button>
 </form>
 </div>
```

```

<h2 class="text-center text-white mt-5 greybg" th:text="#{view_all_computers}">.....All
Computers.....</h2>
<div class="offers row mx-auto d-flex flex-row justify-content-center .row-cols-auto">
<div
 th:each="c : ${computers}" th:object="${c}"
 class="offer card col-sm-2 col-md-3 col-lg-3 m-2 p-0">
 <div class="card-img-top-wrapper" style="height: 20rem">

 </div>
 <div class="card-body pb-1">
 <h5 class="card-title"
 th:text="' Model: ' + *{model}">
 Model name</h5>
 </div>
 <ul class="offer-details list-group list-group-flush">
 <li class="list-group-item">
 <div class="card-text">Processor</div>
 <div class="card-text">Video card</div>
 <div class="card-text">Ram</div>
 <div class="card-text">Disk</div>
 <div th:if="*{!ssd.isBlank()}" class="card-text">SSD</div>
 <div th:if="*{!moreInfo.isBlank()}" class="card-text">More
info
 </div>
 <div class="card-text">
 style="font-weight: bold">Selling price</div>

 <div class="card-body">
 <div class="row">
 <a class="btn btn-link"
 th:href="@{/items/all/computer/details/{id} (id={itemId})}">Details
 <th:block sec:authorize="hasRole('ADMIN') || hasRole('EMPLOYEE_PURCHASES')">
 <a class="btn btn-link alert-danger"
 th:href="@{/pages/purchases/computers/{id}/edit (id={itemId})}">Update
 <form th:action="@{/pages/purchases/computers/delete/{id} (id={itemId})}"
 th:method="delete">
 <input type="submit" class="btn btn-link alert-danger" value="Delete"></input>
 </form>
 </th:block>
 </div>
 </div>
</div>
<div class="container-fluid row justify-content-center">
 <nav>
 <ul class="pagination">
 <li class="page-item" th:classappend="${computers.isFirst()} ? 'disabled' : ''">
 <a th:unless="${computers.isFirst()}"
 class="page-link"
 th:href="@{/items/all/computer(size=${computers.getSize()},page=0,model=${searchProductItemDTO.getModel()},
minPrice=${searchProductItemDTO.getMinPrice()},maxPrice=${searchProductItemDTO.getMaxPrice()})}">First

 </nav>
 <nav>
 <ul class="pagination">
 <li class="page-item" th:classappend="${computers.hasPrevious() ? '' :
'disabled'}">

```

```

 <a th:if="${computers.hasPrevious()}"
 class="page-link"

th:href="@{/items/all/computer(size=${computers.getSize()},page=${computers.getNumber() - 1},model=${searchProductItemDTO.getModel()},
minPrice=${searchProductItemDTO.getMinPrice()},maxPrice=${searchProductItemDTO.getMaxPrice()}))"
>Previous

</nav>
<nav>
 <ul class="pagination">
 <li class="page-item" th:classappend="${computers.hasNext() ? '' : 'disabled'}">
 <a th:if="${computers.hasNext()}"
 class="page-link"

th:href="@{/items/all/computer(size=${computers.getSize()},page=${computers.getNumber() + 1},model=${searchProductItemDTO.getModel()},
minPrice=${searchProductItemDTO.getMinPrice()},maxPrice=${searchProductItemDTO.getMaxPrice()}))"
>Next

</nav>
<nav>
 <ul class="pagination">
 <li class="page-item" th:classappend="${computers.isLast() ? 'disabled' : ''}">
 <a th:unless="${computers.isLast()}"
 class="page-link"

th:href="@{/items/all/computer(size=${computers.getSize()},page=${computers.getTotalPages() - 1},model=${searchProductItemDTO.getModel()},minPrice=${searchProductItemDTO.getMinPrice()},maxPrice=${searchProductItemDTO.getMaxPrice()}))"
>Last

</nav>
</div>
</div>
</main>

```

## 28. OAuth 2 Simplified

<https://aaronparecki.com/oauth-2-simplified/>

```

<!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-oauth2-client -->
<dependency>
 <groupId>org.springframework.security</groupId>
 <artifactId>spring-security-oauth2-client</artifactId>
 <version>5.7.1</version>
</dependency>

```

✓ [developers.facebook.com/apps/create/](https://developers.facebook.com/apps/create/)

Meta for Developers Docs Tools Support My Apps Search developer documentation Svilen Velikov

### Create an App

Type  Details

#### Select an app type

The app type can't be changed after your app is created. [Learn more](#)

- Business**  
Create or manage business assets like Pages, Events, Groups, Ads, Messenger, WhatsApp, and Instagram Graph API using the available business permissions, features and products.
- Consumer**  
Connect consumer products and permissions, like Facebook Login and Instagram Basic Display to your app.
- Instant Games**  
Create an HTML5 game hosted on Facebook.
- Gaming**  
Connect an off-platform game to Facebook Login.
- Workplace**  
Create enterprise tools for Workplace from Meta.
- Academic research**  
Connect to Facebook data and tooling to perform research on Facebook.
- None**

Give Feedback

✓ [developers.facebook.com/apps/create/](https://developers.facebook.com/apps/create/)

Dict Online platforms SoftUni LCW other SOFT Acad... Polezno The HackerRank Int... Discord How we hire Java Web - May 2022 Gmail YouTube

Developers Docs Tools Support My Apps Search developer documentation Svilen Velikov

### Type

#### Provide basic information

**Display name**  
This is the app name associated with your app ID. You can change this later.  
 10/30

**App contact email**  
This email address is used to contact you about potential policy violations, app restrictions or steps to recover the app if it's been deleted or compromised.

**Business Account - Optional**  
To access certain permissions or features, apps need to be connected to a Business Account.

By proceeding, you agree to the [Facebook Platform Terms](#) and [Developer Policies](#). [Previous](#) [Create app](#)

The screenshot shows the Facebook Developers App Settings page for an app named 'OAuth2Demo'. The left sidebar shows navigation options like Dashboard, Settings (selected), Basic, Advanced, Roles, Alerts, App Review, and Products. The main area displays app configuration fields: App ID (1277767399626116), App secret (redacted), Display name (OAuth2Demo), Namespace (empty), App domains (empty), Contact email (svilkata\_sh@abv.bg), Privacy Policy URL (Privacy policy for Login dialog and app details), Terms of Service URL (Terms of Service for Login dialog and App Details), and Category (empty). A 'Show' link is visible next to the App secret field.

```
spring:
 security:
 oauth2:
 client:
 registration:
 facebook:
 client-id: 1277767399626116
 client-secret: fd.....
```

Оттук можем да не се съгласим с автоматичното логване, което еднократно вече сме задали

facebook.com/settings?tab=applications&ref=settings

**Settings**

- Stories
- Reaction preferences
- Notifications
- Mobile
- Apps and Websites**
- Business Integrations
- Ads
- Ads Payments
- Facebook Pay
- Support Inbox
- Videos

Non-public

We've made changes to Apps and Websites to include apps that are temporarily offline as well as those permanently removed by the app developer or Facebook. As a result, you may see more apps in your settings.

App	Added on	Status	Action
OAuth2Demo	Added on Sep 10, 2022	Active	<a href="#">View and Edit</a> <a href="#">Remove</a> ✓
Центрър градска мобилност	Added on Sep 1, 2022	Active	<a href="#">View and Edit</a> <a href="#">Remove</a>
Grabo.bg	Added on Dec 26, 2014	Active	<a href="#">View and Edit</a> <a href="#">Remove</a>
Eventbrite	Added on Aug 26, 2016	Active	<a href="#">View and Edit</a> <a href="#">Remove</a>
Gotin Trip	Added on Apr 17, 2021	Active	<a href="#">View and Edit</a> <a href="#">Remove</a>
Viber	Added on Jun 26, 2016	Active	<a href="#">View and Edit</a> <a href="#">Remove</a>

Ако искаме да бачка фб логин, то трябва да укажем изрично Valid OAuth Redirect URIs (обикновено подсказва и за бизнес акаунт че иска, но може и без).

OAuth2Demo

App ID: 1277767399626116

App Mode: Development  Live A

Dashboard

Required actions

App settings

App roles

Alerts

App Review

Products Add Product

Facebook Login

Settings ✓

Quickstart

App Events

Activity log

Valid OAuth

A manually specified redirect\_uri used with Login on the web must exactly match one of the URLs listed here. This list is also used by the JavaScript SDK for in-app browsers that suppress popups. [?]

https://computerstoreproject.herokuapp.com/login/oauth2/code/facebook

Copy to clipboard

No

Allowed Domains

Login and sign up

<https://computerstoreproject.herokuapp.com/login/oauth2/code/facebook>

Valid OAuth Redirect URIs

http://localhost redirects are automatically allowed while in development mode only and do not need to be added here.

http://computerstoreproject.herokuapp.com/login/oauth2/code/facebook http://localhost:8080/login/oauth2/code/facebook

Copy to clipboard

[https://www.facebook.com/v2.8/dialog/oauth?response\\_type=code&client\\_id=1277767399626116&scope=public\\_profile%20email&state=SD\\_jaUb8Uflss975VpXs1LxdYEGfIEf4DIMIKDa3qV8%3D&redirect\\_uri=https://computerstoreproject.herokuapp.com/login/oauth2/code/facebook](https://www.facebook.com/v2.8/dialog/oauth?response_type=code&client_id=1277767399626116&scope=public_profile%20email&state=SD_jaUb8Uflss975VpXs1LxdYEGfIEf4DIMIKDa3qV8%3D&redirect_uri=https://computerstoreproject.herokuapp.com/login/oauth2/code/facebook)

## 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 proxy-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 името на proxy-то.

Demo Proxies без Spring

```
public class Test {
 public static void main(String[] args) {
 StudentServiceIfc studentServiceIfc = getStudentService();

 args = {String[0]@472} []
 studentServiceIfc = {$Proxy0@617} "bg.softuni.proxies.StudentService@6d03e736"
 > f h = {CacheableInvocationHandler@640}

 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;

@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;
 }
}

@Configuration
public class WebConfig implements WebMvcConfigurer {

 private LocaleChangeInterceptor localeChangeInterceptor;

 public WebConfig(LocaleChangeInterceptor localeChangeInterceptor) {
 this.localeChangeInterceptor = localeChangeInterceptor;
 }
}

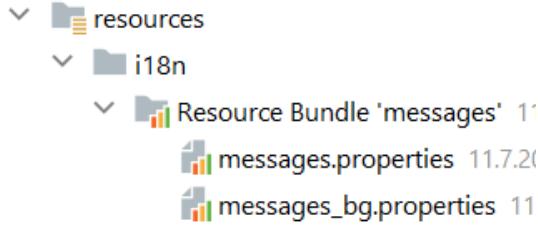
```

```

 }

 @Override
 public void addInterceptors(InterceptorRegistry registry) {
 registry.addInterceptor(localeChangeInterceptor);
 }
}

```



```

navbar_language = Language
navbar_login = Login
navbar_register = Register
navbar_all_brands= All brands
navbar_all_offers= All offers
navbar_search= Search

navbar_language = Ъ•Д•Д,Д°
navbar_login = Ъ' Н..Д%Д'
navbar_register = Д ДµД³Д, Н□Н, Н□Д°Н+Д, Н□
navbar_all_brands= Д' Н□Д, Н‡Д¤Д, Д%Д°Н□Д¤Д,
navbar_all_offers= Д' Н□Д, Н‡Д¤Д, Д%Д±Н□Д²Д,
navbar_search= Ѓ¢ÑšÑ□Ñ□ДµД%Дµ

```

```

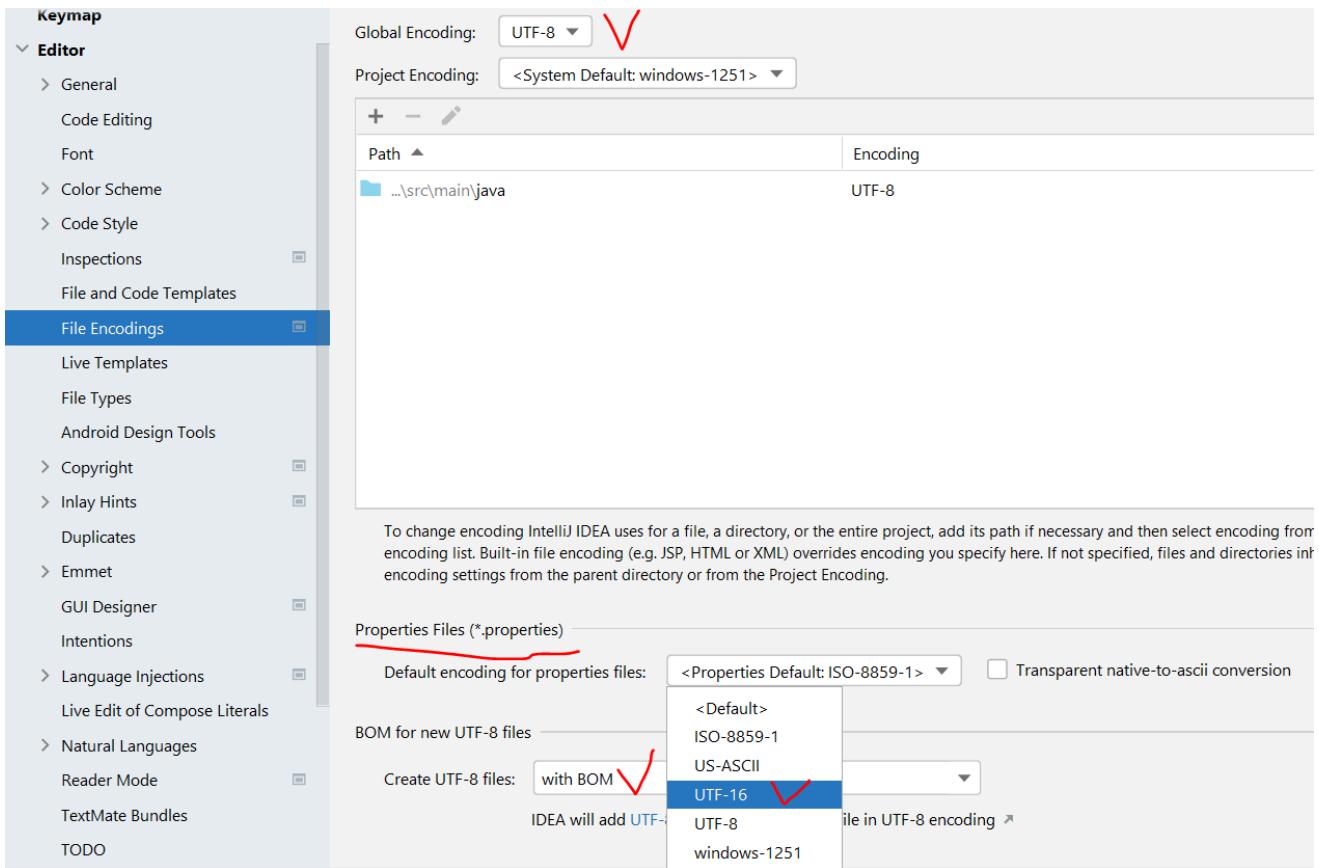
<li class="nav-item">
 <form th:method="get">
 <label class="text-white" th:text="#{navbar_language}" for="lang"></label>
 <select
 onchange="this.form.submit()"
 id="lang"
 name="lang">
 <option value="en_US" th:selected="${#locale.toString()} == 'en_US'">English</option>
 <option value="bg_BG" th:selected="${#locale.toString()} == 'bg_BG'">Български</option>
 </select>
 </form>

<li class="nav-item">
 Search

<li class="nav-item">
 All Offers


```

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
```

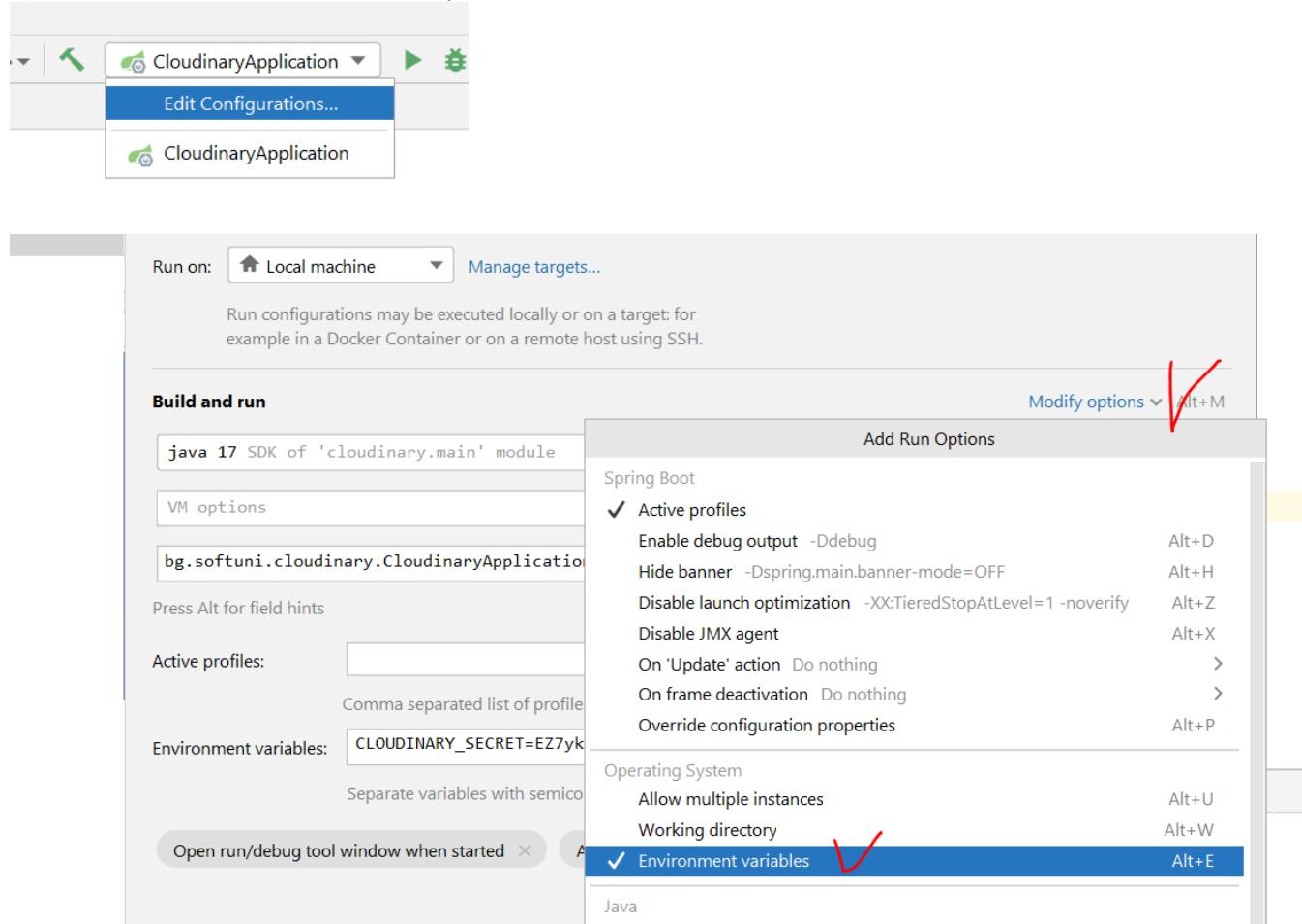
```
application.yml
spring:
 h2:
 console:
 enabled: true
 servlet:
 multipart:
 max-file-size: 10MB
 max-request-size: 10MB
 mvc:
 hiddenmethod:
 filter:
 enabled: true
 jpa:
 # defer-datasource-initialization: true
 database-platform: org.hibernate.dialect.H2Dialect
 hibernate:
 ddl-auto: create

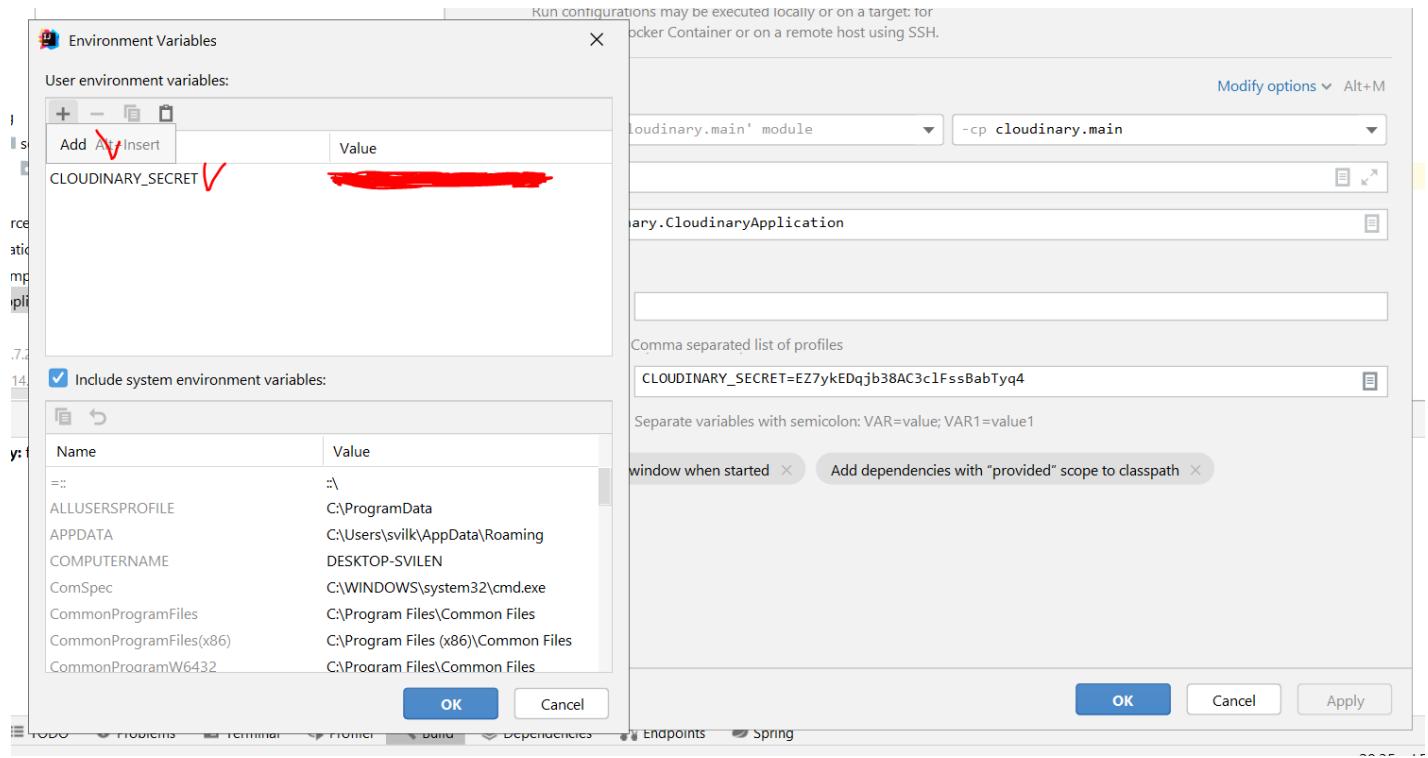
#Cloudinary properties
cloudinary:
 cloud_name: "dtfd8gw16"
 api_key: "218356847735493"
 api_secret: ${CLOUDINARY_SECRET}
```

```
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`

```

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 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:

The Cloudinary `upload` method performs an authenticated upload API call over HTTPS:

```
cloudinary.uploader().upload(Object file, Map options);
```

**Tip:** For a full listing of the possible optional parameters (`options = {}`) available for the upload method, see the [Upload API reference](#).

For example, uploading a local image file named `sample.jpg`:

```
cloudinary.uploader().upload("sample.jpg", ObjectUtils.emptyMap());
```

Uploading is performed synchronously, and once finished, the uploaded asset

```
@SuppressWarnings({ "rawtypes", "unchecked" })
public class Cloudinary {
```

## 32. Pageabale and Sorted

## Demo1 – слаба ракия

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
```

@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

localhost:8080/books/pageablesort X +

localhost:8080/books/pageablesorted?pageNo=3

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
```

```

localhost:8080/books/pageablesorted?pageSize=4&pageNo=0&sortBy=title
+ JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
content:
 0:
 author:
 name: "Николай Хайтов"
 id: 2
 title: "Диви разкази"
 isbn: "7е363430-a7db-4ba2-b6bc-8ad426f4bc7d"
 1:
 author:
 name: "Елин Пелин"
 id: 4
 title: "Пижо и Пенда"
 isbn: "f6731d63-c915-4ae5-9a1f-c42b248ce93a"
 2:
 author:
 name: "Иван Вазов"
 id: 8
 title: "Под Игото"
 isbn: "a6e781a9-e744-4192-9e65-254980af431f"
 3:
 author:
 name: "Елин Пелин"
 id: 6
 title: "Под манастирската лоза"
 isbn: "a38a79f8-48d5-4a72-8c5f-33e33163cb44"
pageable:
 sort:
 empty: false
 unsorted: false

```

## Demo 2 - Да

By having it extend [PagingAndSortingRepository](#), we get `findAll(Pageable pageable)` and `findAll(Sort sort)` methods for paging and sorting.

Conversely, we could have chosen to extend [JpaRepository](#) instead, as it extends [PagingAndSortingRepository](#) too.

Once we extend [PagingAndSortingRepository](#), we can add our own methods that take `Pageable` and `Sort` as parameters, like we did here with `findAllByPrice`.

```
public interface ProductRepository extends PagingAndSortingRepository<Product, Integer> {
 List<Product> findAllByPrice(double price, Pageable pageable); }
```

ДА СИ РАЗЦЪКАМ КОЙ МЕТОД КАКВО ТОЧНО ПРАВИ АКО ЩЕ ПРАВЯ НАНОВО

В репозиторито, SQL заявките вървят реално!

```
<div class="container-fluid row justify-content-center">
 <nav>
 <ul class="pagination">
 <li class="page-item" th:classappend="${computers.isFirst()} ? 'disabled' : ''">
 <a th:unless="${computers.isFirst()}" class="page-link" href="@{/items/all/computer(size=${computers.getSize()},page=0)}>First
```

```


</nav>

<nav>
 <ul class="pagination">
 <li class="page-item" th:classappend="${computers.hasPrevious() ? '' : 'disabled'}">
 <a th:if="${computers.hasPrevious()}"
 class="page-link"

th:href="@{/items/all/computer(size=${computers.getSize()},page=${computers.getNumber() - 1})}">Previous

</nav>

<nav>
 <ul class="pagination">
 <li class="page-item" th:classappend="${computers.hasNext() ? '' : 'disabled'}">
 <a th:if="${computers.hasNext()}"
 class="page-link"

th:href="@{/items/all/computer(size=${computers.getSize()},page=${computers.getNumber() + 1})}">Next

</nav>

<nav>
 <ul class="pagination">
 <li class="page-item" th:classappend="${computers.isLast() ? 'disabled' : ''}">
 <a th:unless="${computers.isLast()}"
 class="page-link"

th:href="@{/items/all/computer(size=${computers.getSize()},page=${computers.getTotalPages() - 1})}">Last

</nav>
</div>

```

```

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

```

```

@Controller
@RequestMapping("/items/all")
public class ViewItemsController {

 @GetMapping("/computer")
 public String viewAllComputers(Model model,
 @PageableDefault(page = 0,
 size = 3,
 sort = "sellingPrice",
 direction = Sort.Direction.ASC) Pageable pageable) {
 if (!model.containsAttribute("computers")) {
 List<ComputerViewGeneralModel> computers = this.computerService.findAllComputers();
 }
 }
}

```

```

Page<ComputerViewGeneralModel> computers=this.computerService.getAllComputersPageable(pageable);
 model.addAttribute("computers", computers);
}

return "/viewItems/all-computers";
}

@Service
public class ComputerService implements InitializableProductService {

public Page<ComputerViewGeneralModel> getAllComputersPageable(Pageable pageable) {
 Page<ComputerViewGeneralModel> allComputers = this.allItemsRepository
 .findAllByType("computer", pageable)
 .map(comp -> this.structMapper
 .computerEntityToComputerSalesViewGeneralModel((ComputerEntity) comp));

 return allComputers;
}

Repository
public interface AllItemsRepository extends PagingAndSortingRepository<ItemEntity, Long> {
 Page<ItemEntity> findAllByType(String type, Pageable pageable);
}

private void basketsInit() {
 List<ItemEntity> allItemsInTheCurrentBasket = (List<ItemEntity>)
this.allItemsRepository.findAll(); //връща Iterable<ItemEntity>, и затова го cast-ваме
}

```

## More Pagination and Sorting

Similarly, to just have our query results sorted, we can simply pass an instance of `Sort` to the method:

```
Page<Product> allProductsSortedByName = productRepository.findAll(Sort.by("name"));
However, what if we want to both sort and page our data?
```

We can do that by passing the sorting details into our `PageRequest` object itself:

```

Pageable sortedByName =
 PageRequest.of(0, 3, Sort.by("name"));

Pageable sortedByPriceDesc =
 PageRequest.of(0, 3, Sort.by("price").descending());

Pageable sortedByPriceDescNameAsc =
 PageRequest.of(0, 5, Sort.by("price").descending().and(Sort.by("name")));
Based on our sorting requirements, we can specify the sort fields and the sort direction while creating
our PageRequest instance.
```

## 33. MailHog

SMTP server for testing receiving e-mails locally.

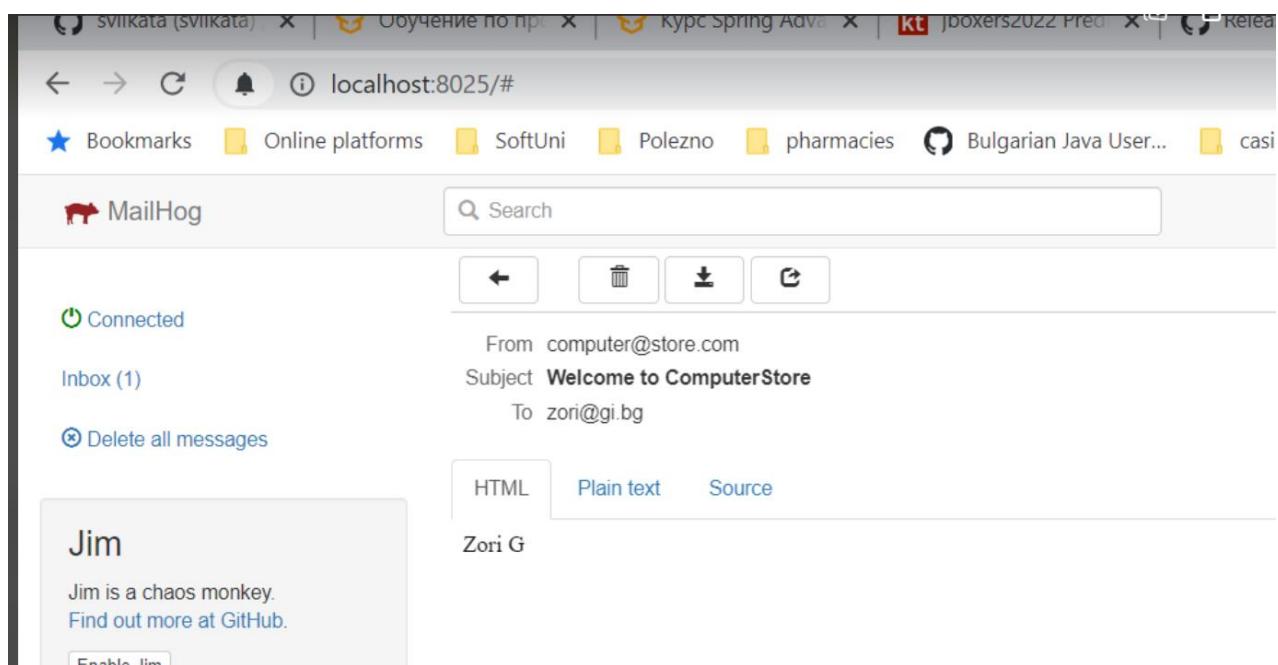
Може да се инсталира на Windows или на Linux.

Може и през Докер с docker-compose.yml файл.

<https://www.youtube.com/watch?v=yozMCBg2sBk&t=307s>

Като се подкара, заема порт localhost:8025

```
2022/11/22 00:26:50 Using in-memory storage
2022/11/22 00:26:50 [SMTP] Binding to address: 0.0.0.0:1025
[HTTP] Binding to address: 0.0.0.0:8025
2022/11/22 00:26:50 Serving under http://0.0.0.0:8025/
Creating API v1 with WebPath:
Creating API v2 with WebPath:
```



## 34. JavaMailSender from Spring

### 1. Demo with MailHog

Какво правим в Java Spring, за да го подкараме – ами много неща в Spring реално:

```
<!DOCTYPE html>
<html lang="en">
 xmlns:th="http://www.thymeleaf.org"
 xmlns:sec='http://www.thymeleaf.org/extras/spring-security'>
```

```
<head th:replace="fragments/commons::head"></head>

<body>

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

<main>
 <div class="container">
 <h2 class="text-center text-white greybg" th:text="#{register}">.....Register
Customer.....</h2>
 <form th:action="@{/users/register}"
 th:method="post"
 th:object="${userRegistrationModel}"
 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="firstName" class="text-white font-weight-bold">First
Name</label>
 <input
 id="firstName"
 th:field="*{firstName}"
 th:errorclass="is-invalid"
 type="text"
 class="form-control"
 placeholder="First name"
 required minlength="2" maxlength="20"/>
 <div class="invalid-feedback errors alert alert-danger">
 <div th:each="err : ${#fields.errors('firstName')}" th:text="${err}">
 </div>
 </div>
 </div>
 <div class="form-group col-md-6 mb-3">
 <label for="lastName" class="text-white font-weight-bold">Last Name</label>
 <input id="lastName"
 th:field="*{lastName}"
 th:errorclass="is-invalid"
 type="text"
 class="form-control"
 placeholder="Last name"
 required minlength="2" maxlength="20"/>
 <div class="invalid-feedback errors alert alert-danger">
 <div th:each="err : ${#fields.errors('lastName')}" th:text="${err}">
 </div>
 </div>
 </div>
 </div>
 <div class="row">
 <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="email"
 class="form-control"
 placeholder="email"
 required/>
 <div class="invalid-feedback errors alert alert-danger">
 <div th:each="err : ${#fields.errors('email')}" th:text="${err}">
 </div>
 </div>
 </div>
 </div>
 </form>
 </div>
</main>
```

```

 </div>
 </div>
 <div class="form-group col-md-6 mb-3">
 <label for="username" class="text-white font-weight-bold">Username</label>
 <input id="username"
 th:field="*{username}"
 th:errorclass="is-invalid"
 type="text"
 class="form-control"
 placeholder="username"
 required minlength="3" maxlength="20"/>
 <div class="invalid-feedback errors alert alert-danger">
 <div th:each="err : ${#fields.errors('username')}" th:text="${err}">
 </div>
 </div>
 </div>

 <div class="row">
 <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"
 required minlength="3" maxlength="20"/>
 <div class="invalid-feedback errors alert alert-danger">
 <div th:each="err : ${#fields.errors('password')}" th:text="${err}">
 </div>
 </div>

 <div class="form-group col-md-6 mb-3">
 <label for="confirmPassword" class="text-white font-weight-bold">Confirm
 Password</label>
 <input id="confirmPassword"
 name="confirmPassword"
 th:field="*{confirmPassword}"
 th:errorclass="is-invalid"
 type="password"
 class="form-control"
 placeholder="confirmPassword"
 required minlength="3" maxlength="20"/>
 <div class="invalid-feedback errors alert alert-danger">
 <div th:each="err : ${#fields.errors('confirmPassword')}" th:text="${err}">
 </div>
 <!--
 <p class="invalid-feedback exception alert alert-danger">-->
 <!--
 Passwords should match.-->
 </p>-->
 </div>
 </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="Register"/>
 </div>
</div>
</form>
</div>
</main>

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

</body>
</html>

@Controller
public class RegistrationController {
 private final UserService userService;

 public RegistrationController(UserService userService) {
 this.userService = userService;
 }

 @ModelAttribute("userRegistrationModel")
 public void initUserRegistrationModel(Model model){ //изпълнява се в рамките на текущия
 контролер само!!!
 model.addAttribute("userRegistrationModel", new UserRegisterBindingDTO());
 }

 @GetMapping("/users/register")
 public String register() {
 // когато зареждаме за първи път страницата, то автоматично ще влезе към модела атрибут
 userRegistrationModel == празен new UserRegisterBindingDto()
 return "/user/auth-registerUser";
 }

 @PostMapping("/users/register")
 public String registerConfirm(@Valid UserRegisterBindingDTO userRegisterBindingDto,
 BindingResult bindingResult,
 RedirectAttributes redirectAttributes) {

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

 redirectAttributes.addFlashAttribute("org.springframework.validation.BindingResult.userRegistrationModel",
 bindingResult);
 return "redirect:/users/register";
 }

 userService.registerUserAndAutoLogin(userRegisterBindingDto);
 return "redirect:/";
 }
}

pom.xml

```

```

<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-mail</artifactId>
 <version>2.7.5</version>
</dependency>

application.yml
mail:
 host: "localhost"
 port: 1025
 username: "computer@store.com"
 password: ""

import bg.softuni.computerStore.initSeed.InitializableUserService;
import bg.softuni.computerStore.model.binding.user.ChangeUserPasswordDTO;
import bg.softuni.computerStore.model.binding.user.EmployeeRegisterBindingDTO;
import bg.softuni.computerStore.model.binding.user.UserRegisterBindingDTO;
import bg.softuni.computerStore.model.binding.user.UserRolesBindingDTO;
import bg.softuni.computerStore.model.entity.users.UserEntity;
import bg.softuni.computerStore.model.entity.users.UserRoleEntity;
import bg.softuni.computerStore.model.enums.UserRoleEnum;
import bg.softuni.computerStore.model.view.user.UserViewModel;
import bg.softuni.computerStore.repository.users.UserRepository;
import bg.softuni.computerStore.repository.users.UserRoleRepository;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Value;
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.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.stream.Collectors;

@Service
public class UserService implements InitializableUserService {
 private final UserRepository userRepository;
 private final UserRoleRepository userRoleRepository;
 private final PasswordEncoder passwordEncoder;
 private final UserDetailsService appUserDetailsService;
 private final String adminPass;
 private final ModelMapper modelMapper;
 private final BasketService basketService;
 private final DemoEmailService demoEmailService;

 public UserService(
 UserRepository userRepository, UserRoleRepository userRoleRepository,
 PasswordEncoder passwordEncoder,
 UserDetailsService appUserDetailsService,
 @Value("${app.default.admin.password}") String adminPass, ModelMapper modelMapper,
 BasketService basketService, DemoEmailService demoEmailService) {
 this.userRepository = userRepository;
 }
}

```

```

 this.userRoleRepository = userRoleRepository;
 this.passwordEncoder = passwordEncoder;
 this.appUserDetailsService = appUserDetailsService;
 this.adminPass = adminPass;
 this.modelMapper = modelMapper;
 this.basketService = basketService;
 this.demoEmailService = demoEmailService;
 }

//customers
public Long registerUserAndAutoLogin(UserRegisterBindingDTO userRegisterBindingDTO) {
 //The customer user role
 UserRoleEntity userRole = userRoleRepository.findById(4L).get();

 UserEntity newCustomer =
 new UserEntity().
 setUserRoles(Set.of(userRole)).
 setUsername(userRegisterBindingDTO.getUsername()).
 setEmail(userRegisterBindingDTO.getEmail()).
 setFirstName(userRegisterBindingDTO.getFirstName()).
 setLastName(userRegisterBindingDTO.getLastName());

 setPassword(passwordEncoder.encode(userRegisterBindingDTO.getPassword()));

 UserEntity savedUser = userRepository.save(newCustomer);

 //Creating here the relevant basket
 this.basketService.addBasketForRegisteredUser(savedUser);

 //this is the Spring representation of a User - after register, we AUTO Log-in the users directly = THE LOGIN PROCESS
 login(newCustomer.getUsername());

 demoEmailService.sendRegistrationEmail(savedUser.getEmail(), savedUser.getUsername());

 return savedUser.getId();
}

import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;
import org.thymeleaf.TemplateEngine;
import org.thymeleaf.context.Context;

import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;

@Service
public class DemoEmailService {
 private final TemplateEngine templateEngine;
 private final JavaMailSender javaMailSender;

 public DemoEmailService(TemplateEngine templateEngine, JavaMailSender javaMailSender) {
 this.templateEngine = templateEngine;
 this.javaMailSender = javaMailSender;
 }
}

```

```

public void sendRegistrationEmail(String userEmail, String userName) {
 MimeMessage mimeMessage = javaMailSender.createMimeMessage();

 try {
 MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(mimeMessage);
 mimeMessageHelper.setFrom("computer@store.com");
 mimeMessageHelper.setTo(userEmail);
 mimeMessageHelper.setSubject("Welcome to ComputerStore");
 mimeMessageHelper.setText(userName, true);

 javaMailSender.send(mimeMessageHelper.getMimeMessage());
 } catch (MessagingException e) {
 throw new RuntimeException(e);
 }
}

private String generateMessageContent(String userName){
 Context ctx = new Context(); //org.thymeleaf.context;
 ctx.setVariable("userRegistrationModel.username", userName);

 return templateEngine.process("/user/auth-registerUser", ctx);
}
}

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.JavaMailSenderImpl;

import java.util.Properties;

@Configuration
public class MailConfig {

 @Bean
 public JavaMailSender javaMailSender(
 @Value("${mail.host}") String mailHost,
 @Value("${mail.port}") Integer mailPort,
 @Value("${mail.username}") String userName,
 @Value("${mail.password}") String password) {

 JavaMailSenderImpl javaMailSender = new JavaMailSenderImpl();
 javaMailSender.setHost(mailHost);
 javaMailSender.setPort(mailPort);
 javaMailSender.setUsername(userName);
 javaMailSender.setPassword(password);
 javaMailSender.setJavaMailProperties(mailProperties());

 return javaMailSender;
 }

 private Properties mailProperties() {
 Properties properties = new Properties();

```

```

 properties.setProperty("mail.smtp.auth", "true");
 properties.setProperty("mail.transport.protocol", "smtp");

 return properties;
 }
}

```

application.yml

```

mail:
 host: "localhost"
 port: 1025
 username: "computer@store.com"
 password: ""

```

## 2. Demo with real gmail e-mail

```

application.yml
mail:
 default-encoding: UTF-8
 host: smtp.gmail.com
 username: store@gmail.com the real e-mail here
 password: 123
 port: 587
 properties:
 mail:
 smtp:
 auth: true
 starttls:
 enable: true
 protocol: smtp
 test-connection: false

import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.thymeleaf.TemplateEngine;
import org.thymeleaf.context.Context;

import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;
import java.time.LocalDate;
import java.util.Locale;

@Service
public class EmailServiceImpl implements EmailService {

 private final JavaMailSender javaMailSender;
 private final TemplateEngine templateEngine;

 public EmailServiceImpl(JavaMailSender javaMailSender, TemplateEngine
templateEngine) {

```

```

 this.javaMailSender = javaMailSender;
 this.templateEngine = templateEngine;
 }

 public void sendEmail(String toEmail, String subject, String body) {
 SimpleMailMessage message = new SimpleMailMessage();
 message.setFrom("store@gmail.com");
 message.setTo(toEmail);
 message.setText(body);
 message.setSubject(subject);
 javaMailSender.send(message);
 System.out.println("Message sent successfully");
 }

 public void sendEmailToMultipleRecipients(String [] toEmails, String subject, String body) {
 MimeMessage mimeMessage = javaMailSender.createMimeMessage();
 MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(mimeMessage);
 try {
 mimeMessageHelper.setFrom("store@gmail.com");
 mimeMessageHelper.setTo(toEmails);
 mimeMessageHelper.setSubject(subject + LocalDate.now());
 mimeMessageHelper.setText(generateTaskEmailText(body), true);

 javaMailSender.send(mimeMessageHelper.getMimeMessage());
 } catch (MessagingException e) {
 throw new RuntimeException(e);
 }
 System.out.println("Message sent successfully");
 }

 private String generateTaskEmailText(String body) {
 Context ctx = new Context();
 ctx.setLocale(Locale.getDefault());

 ctx.setVariable("body", body);

 return templateEngine.process("task-email", ctx);
 }

 public void sendRegistrationEmail(UserViewModel userEmailViewModel) {

 MimeMessage mimeMessage = javaMailSender.createMimeMessage();
 MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(mimeMessage);

 try {
 mimeMessageHelper.setFrom("store@gmail.com");
 mimeMessageHelper.setTo(userEmailViewModel.getEmail());
 mimeMessageHelper.setSubject("Welcome to the Store Family!");
 mimeMessageHelper.setText(generateEmailText(userEmailViewModel), true);

 javaMailSender.send(mimeMessageHelper.getMimeMessage());
 } catch (MessagingException e) {
 throw new RuntimeException(e);
 }
 }
}

```

```

 }

}

private String generateEmailText(UserViewModel userEmailViewModel) {
 Context ctx = new Context();
 ctx.setLocale(Locale.getDefault());
 String userFullName = userEmailViewModel.getFirstName() + " " +
userEmailViewModel.getLastName();
 ctx.setVariable("user", userFullName);

 return templateEngine.process("email-template", ctx);
}

public void sendForgottenPasswordEmail(String email, String resetPasswordLink)
throws MessagingException {
 MimeMessage message = javaMailSender.createMimeMessage();
 MimeMessageHelper helper = new MimeMessageHelper(message);

 helper.setFrom("store@gmail.com");
 helper.setTo(email);

 String subject = "Here's the link to reset your password";

 String content = "<p>Hello,</p>" +
 "<p>You have requested to reset your password.</p>" +
 "<p>Click the link below to change your password:</p>" +
 "<p>Change my password</p>" +
 "
" +
 "<p>Ignore this email if you do remember your password, " +
 "or you have not made the request.</p>";

 helper.setSubject(subject);
 helper.setText(content, true);
 javaMailSender.send(message);
}
}

```

## 35. JWT (JSON Web Token)

What is JSON Web Token?

Не се прави при системи с логин на потребители. А когато например един микросървис комуникира с друг микросървис., или една система с друга система.

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>

You can see demo for the autorepairs project (GitLab)

Demo from stuckata:

<https://github.com/stuckata/spring-boot-security-jwt-demo>

`package com.demo.springbootsecurityjwtdemo.config;`

```
import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.info.Contact;
import io.swagger.v3.oas.annotations.info.Info;
import io.swagger.v3.oas.annotations.info.License;
import io.swagger.v3.oas.annotations.servers.Server;
import io.swagger.v3.oas.models.Components;
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.security.SecurityRequirement;
import io.swagger.v3.oas.models.security.SecurityScheme;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
@OpenAPIDefinition(
 info = @Info(
 title = "${api.title}",
 version = "${api.version}",
 contact = @Contact(
 name = "${api.contact.name}",
 email = "${api.contact.email}",
 url = "${api.contact.url}"
),
 license = @License(
 name = "${api.license.name}",
 url = "${api.license.url}"
),
 termsOfService = "${api.tos.uri}",
 description = "${api.description}"
),
 servers = {
 @Server(
 url = "http://localhost:9090",
 description = "Development"
),
 @Server(
 url = "${api.server.url}",
 description = "${api.stage}"
)
 }
)
public class OpenAPI30Configuration {

 public static final String SECURITY_SCHEME_BEARER_TOKEN = "bearerAuth";

 /**
 * Configure the OpenAPI components.
 *
 * @return Returns fully configure OpenAPI object

```

```

* @see OpenAPI
*/
@Bean
public OpenAPI customizeOpenAPI() {

 return new OpenAPI()
 .addSecurityItem(new SecurityRequirement().addList(SECURITY_SCHEME_BEARER_TOKEN))
 .components(new Components()
 .addSecuritySchemes(SECURITY_SCHEME_BEARER_TOKEN, new SecurityScheme()
 .name(SECURITY_SCHEME_BEARER_TOKEN)
 .type(SecurityScheme.Type.HTTP)
 .scheme("bearer")
 .description(
 "Provide the JWT token. JWT token can be obtained from the Authentication API.")
 .bearerFormat("JWT")));
}
}

```

```

package com.demo.springbootsecurityjwtdemo.config;

import com.nimbusds.jose.jwk.JWK;
import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.ImmutableJWKSet;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.security.oauth2.jwt.JwtEncoder;
import org.springframework.security.oauth2.jwt.NimbusJwtDecoder;
import org.springframework.security.oauth2.jwt.NimbusJwtEncoder;
import
org.springframework.security.oauth2.server.resource.web.BearerTokenAuthenticationEntryPoint;
import
org.springframework.security.oauth2.server.resource.web.access.BearerTokenAccessDeniedHandler;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.util.Arrays;
import java.util.List;

@EnableWebSecurity

@Configuration
public class SecurityConfiguration {

 @Value("${cors.allowedOrigins}")
 private String[] corsAllowedOrigins;

 @Value("${jwt.public.key}")
 private RSAPublicKey publicKey;

 @Value("${jwt.private.key}")
 private RSAPrivateKey privateKey;
}

```

```

/**
 * This bean is used to configure the JWT token. Configure the URLs that should not be
protected by the JWT token.
 */
@Param http the HttpSecurity object
@return the HttpSecurity object
@throws Exception if an error occurs
*/
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
 HttpSecurity securityConfig = http
 .authorizeHttpRequests(authorizeRequests -> authorizeRequests
 .requestMatchers(
 "/api/auth/login",
 "/api/auth/account/create",
 "/swagger-ui/**",
 "/v3/api-docs/**")
 .permitAll()
 .anyRequest()
 .authenticated())
 .cors(cors -> cors.configurationSource(corsConfigurationSource()))
 .csrf(AbstractHttpConfigurer::disable)
 .formLogin(AbstractHttpConfigurer::disable) //removing default Spring configs
 .httpBasic(AbstractHttpConfigurer::disable) //removing default Spring configs
 .sessionManagement(session ->
 session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
 .oauth2ResourceServer(oauth2 -> oauth2.jwt(Customizer.withDefaults()))
 .exceptionHandling(exceptions -> exceptions
 .authenticationEntryPoint(new BearerTokenAuthenticationEntryPoint())
 .accessDeniedHandler(new BearerTokenAccessDeniedHandler()));
 return securityConfig.build();
}

/**
 * Used by spring security if CORS is enabled.
 */
@Bean
public CorsConfigurationSource corsConfigurationSource() {
 CorsConfiguration config = new CorsConfiguration();
 config.setAllowedOrigins(Arrays.asList(this.corsAllowedOrigins));
 config.setAllowCredentials(true);
 config.setAllowedHeaders(List.of("*"));
 config.setAllowedMethods(List.of("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"));
 UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
 source.registerCorsConfiguration("/**", config);
 return source;
}

/**
 * This bean is used to decode the JWT token.
 *
 * @return Returns the JwtDecoder bean to decode JWT tokens.
 */
@Bean
JwtDecoder jwtDecoder() {
 return NimbusJwtDecoder.withPublicKey(this.publicKey).build();
}

/**
 * This bean is used to encode the JWT token.
 *
 * @return Returns the JwtEncoder bean to encode JWT tokens.
 */
@Bean

```

```

JwtEncoder jwtEncoder() {
 JWK jwk = new RSAKey
 .Builder(this.publicKey)
 .privateKey(this.privateKey)
 .build();
 return new NimbusJwtEncoder(new ImmutableJWKSet<>(new JWKSet(jwk)));
}
}

package com.demo.springbootsecurityjwtdemo.service;

import com.demo.springbootsecurityjwtdemo.api.dto.LoginRequestDto;
import com.demo.springbootsecurityjwtdemo.entity.UserEntity;
import com.demo.springbootsecurityjwtdemo.entity.UserStatus;
import com.demo.springbootsecurityjwtdemo.exception.ApplicationException;
import com.demo.springbootsecurityjwtdemo.service.jwt.JwtService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.util.Set;

import static com.demo.springbootsecurityjwtdemo.api.dto.ErrorCode.USER_ACCOUNT_NOT_ACTIVE;

@Service
public class AuthenticationServiceImpl implements AuthenticationService {

 private final UserService userService;
 private final JwtService jwtService;

 @Override
 public String authenticate(LoginRequestDto dto) throws ApplicationException {
 UserEntity userEntity = this.userService.findUserByUsernameAndPassword(dto);
 return generateToken(userEntity, dto.getEmail());
 }

 @Override
 public String refreshToken() throws ApplicationException {
 String email = SecurityContextHolder.getContext().getAuthentication().getName();
 UserEntity userEntity = this.userService.findUserByEmail(email);
 return generateToken(userEntity, email);
 }

 private String generateToken(UserEntity userEntity, String email) throws ApplicationException {
 boolean isActive = UserStatus.ACTIVE.equals(userEntity.getStatus());
 if (!isActive) {
 throw new ApplicationException(USER_ACCOUNT_NOT_ACTIVE);
 }
 UserDetails user = createUserDetails(userEntity, email);
 return this.jwtService.generateToken(user);
 }

 private static UserDetails createUserDetails(UserEntity userEntity, String email) {
 return new User(
 email,
 email,
 true,
 true,

```

```

 true,
 true,
 Set.of(new SimpleGrantedAuthority(userEntity.getRole().name())))
);
}
}

package com.demo.springbootsecurityjwtdemo.service.jwt;

import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.oauth2.jwt.JwtClaimsSet;
import org.springframework.security.oauth2.jwt.JwtEncoder;
import org.springframework.security.oauth2.jwt.JwtEncoderParameters;
import org.springframework.stereotype.Service;

import java.time.Instant;
import java.util.Collections;
import java.util.Map;
import java.util.Objects;
import java.util.stream.Collectors;

@RequiredArgsConstructor
@Service
public class JwtServiceImpl implements JwtService {

 @Value("${jwt.expirationTimeInSeconds}")
 private int jwtExpirationTimeInSeconds;

 @Value("${jwt.issuer}")
 private String jwtIssuer;

 private final JwtEncoder encoder;

 @Override
 public String generateToken(UserDetails userDetails) {
 return generateToken(Collections.emptyMap(), userDetails);
 }

 @Override
 public String generateToken(Map<String, Object> additionalClaims, UserDetails userDetails) {
 String roles = userDetails.getAuthorities().stream()
 .map(GrantedAuthority::getAuthority)
 .collect(Collectors.joining(" "));

 Instant now = Instant.now();

 JwtClaimsSet.Builder claimsBuilder = JwtClaimsSet.builder()
 .issuer(this.jwtIssuer)
 .issuedAt(now)
 .expiresAt(now.plusSeconds(this.jwtExpirationTimeInSeconds))
 .subject(userDetails.getUsername())
 .claim("roles", roles); // You can add whatever fields(claims) you need
 additionalClaims.forEach(claimsBuilder::claim);
 JwtClaimsSet claims = claimsBuilder.build();

 return this.encoder.encode(JwtEncoderParameters.from(claims)).getTokenValue();
 }
}

```

```

package com.demo.springbootsecurityjwtdemo.controller;

import com.demo.springbootsecurityjwtdemo.api.AuthenticationApi;
import com.demo.springbootsecurityjwtdemo.api.dto.CreateAccountRequestDto;
import com.demo.springbootsecurityjwtdemo.api.dto.LoginRequestDto;
import com.demo.springbootsecurityjwtdemo.api.dto.TokenResponseDto;
import com.demo.springbootsecurityjwtdemo.exception.ApplicationException;
import com.demo.springbootsecurityjwtdemo.service.AuthenticationService;
import com.demo.springbootsecurityjwtdemo.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.security.oauth2.jwt.Jwt;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;

@RequiredArgsConstructor
@RestController
@RequestMapping("/api/auth")
public class AuthenticationController implements AuthenticationApi {

 private static final String X_AUTH_TOKEN = "X-AUTH-TOKEN";

 @Value("${jwt.expirationTimeInSeconds}")
 private int jwtExpirationTimeInSeconds;

 private final UserService userService;
 private final AuthenticationService authenticationService;
 private final JwtDecoder jwtDecoder;

 @Override
 public ResponseEntity<Void> createAccount(String clientId, CreateAccountRequestDto request)
 throws ApplicationException {
 this.userService.createUser(request);
 return ResponseEntity.noContent().build();
 }

 @Override
 public ResponseEntity<TokenResponseDto> login(String clientId, LoginRequestDto request) throws
 ApplicationException {
 String token = this.authenticationService.authenticate(request);
 return buildTokenResponseDto(token);
 }

 @Override
 public ResponseEntity<TokenResponseDto> refreshToken(String clientId) throws
 ApplicationException {
 String token = this.authenticationService.refreshToken();
 return buildTokenResponseDto(token);
 }

 private ResponseEntity<TokenResponseDto> buildTokenResponseDto(String token) {
 HttpHeaders httpHeaders = new HttpHeaders();
 httpHeaders.set(X_AUTH_TOKEN, token); // optional (you can return it as header OR in the
 body)

 TokenResponseDto tokenDto = TokenResponseDto.builder()
 .token(token)
 .expiresInSec(this.jwtExpirationTimeInSeconds)
 .expirationTimeUtc(getTokenExpirationTimeUtc(token))
 }
}

```

```
 .build();

 return ResponseEntity.ok()
 .headers(httpHeaders)
 .body(tokenDto);
}

private LocalDateTime getTokenExpirationTimeUtc(String token) {
 LocalDateTime expirationTimeUtc = null;
 Jwt jwt = this.jwtDecoder.decode(token);
 Instant expiresAt = jwt.getExpiresAt();
 if (expiresAt != null) {
 expirationTimeUtc = LocalDateTime.ofInstant(expiresAt, ZoneOffset.UTC);
 }
 return expirationTimeUtc;
}
```

36. JWT between 2 backend microservices demo

## Main microservice – at port 8080

```
import bg.softuni.travelNest.service.JWTService;
import bg.softuni.travelNest.service.UserService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.MediaType;
import org.springframework.http.client.ClientHttpRequestInterceptor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.web.client.RestClient;

import java.util.Map;

@Configuration
public class RestConfig {

 @Bean("imageRestClient")
 public RestClient imageRestClient(ImgurConfig imgurConfig){
 return RestClient.builder()
 .defaultHeader("Authorization", "Client-ID " + imgurConfig.getKey())
 .build();
 }

 @Bean("attractionRestClient")
 public RestClient attractionRestClient(AttractionApiConfig attractionApiConfig,
 ClientHttpRequestInterceptor requestInterceptor){
 return RestClient.builder()
 .baseUrl(attractionApiConfig.getBaseUrl())
 .defaultHeader("Content-Type", MediaType.APPLICATION_JSON_VALUE)
 .requestInterceptor(requestInterceptor)
 .build();
 }

 @Bean
 public ClientHttpRequestInterceptor requestInterceptor(UserService userService, JWTService
jwtService){
 return (r, b, e) -> {

```

```

 userService.getCurrentUser()
 .ifPresent(tnud -> {
 String token = jwtService.generateToken(tnud.getUuid().toString(),
 Map.of(
 "username", tnud.getUsername(),
 "roles", tnud.getAuthorities()
 .stream()
 .map(GrantedAuthority::getAuthority)
 .toList()
));
 r.getHeaders().setBearerAuth(token);
 });
 return e.execute(r, b);
 };
}
}

```

---

```

import bg.softuni.travelNest.config.JWTConfig;
import bg.softuni.travelNest.service.JWTService;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.nio.charset.StandardCharsets;
import java.security.Key;
import java.util.Date;
import java.util.Map;

@Service
@RequiredArgsConstructor
public class JWTServiceImpl implements JWTService {

 private final JWTConfig jwtConfig;

 @Override
 public String generateToken(String userId, Map<String, Object> claims) {
 Date now = new Date();

 return Jwts.builder()
 .setClaims(claims)
 .setSubject(userId)
 .setIssuedAt(now)
 .setNotBefore(now)
 .setExpiration(new Date(now.getTime() + jwtConfig.getExpiration()))
 .signWith(getSigningKey(), SignatureAlgorithm.HS256)
 .compact();
 }

 private Key getSigningKey(){
 return Keys
 .hmacShaKeyFor(jwtConfig
 .getSecret())

```

```
 .getBytes(StandardCharsets.UTF_8));
 }
}
```

---

```
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
```

```
@Getter
@Setter
@NoArgsConstructor
@Configuration
@ConfigurationProperties(prefix = "jwt")
public class JWTConfig {

 private String secret;

 private Long expiration;
}
```

---

JWT (JWT secret) from <https://jwt.io/>

```
application.properties
jwt.secret=${JWT_SECRET:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvG4gRG9lIiwiawF0IjoxNTE2MjM5MDIyfQ.SfIKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c}
jwt.expiration=${JWT_EXPIRATION:60000}
```

---

```
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Getter
@Setter
@NoArgsConstructor
@Configuration
@ConfigurationProperties(prefix = "attraction.api")
public class AttractionApiConfig {

 private String baseUrl;
}
```

```
application.properties
attraction.api.baseUrl=http://localhost:8081
```

---

```

import bg.softuni.travelNest.config.Messages;
import bg.softuni.travelNest.exception.ObjectNotFoundException;
import bg.softuni.travelNest.model.dto.AddAttractionDTO;
import bg.softuni.travelNest.model.dto.AttractionDetailsDTO;
import bg.softuni.travelNest.model.dto.TicketDTO;
import bg.softuni.travelNest.service.AttractionService;
import bg.softuni.travelNest.service.PictureService;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestClient;

import java.io.IOException;
import java.math.BigDecimal;
import java.util.List;
import java.util.Objects;
import java.util.UUID;

@Service
public class AttractionServiceImpl implements AttractionService {

 private final @Qualifier("attractionRestClient") RestClient restClient;
 private final ModelMapper modelMapper;
 private final PictureService pictureService;
 private final Messages messages;

 public AttractionServiceImpl(@Qualifier("attractionRestClient") RestClient restClient,
 ModelMapper modelMapper, PictureService pictureService, Messages messages) {
 this.restClient = restClient;
 this.modelMapper = modelMapper;
 this.pictureService = pictureService;
 this.messages = messages;
 }

 @Override
 public List<String> getAttractionCities() {
 return restClient.get()
 .uri("/attraction/cities")
 .accept(MediaType.APPLICATION_JSON)
 .retrieve()
 .body(new ParameterizedTypeReference<>() {});
 }

 @Override
 public List<AttractionDetailsDTO> getAllAttractions(String attractionType) {
 return restClient.get()
 .uri("/{attraction-type}/list", attractionType)
 .accept(MediaType.APPLICATION_JSON)
 .retrieve()
 .body(new ParameterizedTypeReference<>() {});
 }

 @Override
 public AttractionDetailsDTO getAttractionById(UUID attractionId) {

```

```

 return restClient.get()
 .uri("/attraction/details/{uuid}", attractionId)
 .accept(MediaType.APPLICATION_JSON)
 .retrieve()
 .onStatus(HttpStatusCode::is4xxClientError,
 ((request, response) -> {
 throw new
 ObjectNotFoundException(messages.get("message.error.attraction"));
 }))
 .body(AttractionDetailsDTO.class);
}

@Override
public TicketDTO getTickets(UUID uuid) {
 return restClient.get()
 .uri("/attraction/tickets/{uuid}", uuid)
 .accept(MediaType.APPLICATION_JSON)
 .retrieve()
 .body(TicketDTO.class);
}

@Override
public void buyTickets(TicketDTO tickets, UUID attractionId) {
 restClient.post()
 .uri("/attraction/details/{uuid}", attractionId)
 .body(tickets)
 .retrieve();
}

@Override
public UUID add(AddAttractionDTO addAttractionDTO, String attractionType) throws IOException
{
 ;
 return Objects.requireNonNull(restClient.post()
 .uri("/{attraction-type}/add", attractionType)
 .accept(MediaType.APPLICATION_JSON)
 .body(modelMapper.map(addAttractionDTO, AttractionDetailsDTO.class)
 .setCity(addAttractionDTO.getCity().replaceAll("\\.", " "))
 .setPaid(!addAttractionDTO.getPrice().equals(BigDecimal.ZERO)))
 .setPictureUrl(pictureService.uploadImage(addAttractionDTO.getImage())))
 .retrieve()
 .body(AttractionDetailsDTO.class))
 .getId();
}

@Override
public void deleteById(UUID attractionId) {
 restClient.delete()
 .uri("/attraction/delete/{uuid}", attractionId)
 .retrieve()
 .onStatus(HttpStatusCode::is4xxClientError,
 ((request, response) -> {
 throw new
 ObjectNotFoundException(messages.get("message.error.attraction"));
 }));
}

```

```
 }
}
```

---

```
@Configuration
public class SecurityConfig {
 Nothing here about the JWT
}
```

Second microservice to be called by the main microservice – at port 8081

```
import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.boot.actuate.autoconfigure.security.servlet.EndpointRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
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 SecurityConfig {

 private static final String[] AUTH_WHITELIST = {
 "/attraction/cities",
 "/attraction/list",
 "/attraction/details/**",
 "/attraction/tickets/**",
 "/event/cities",
 "/event/list",
 "/event/details/**",
 "/event/tickets/**"
 };

 //injected JWTAuthorizationFilter
 @Bean
 public SecurityFilterChain filterChain(HttpSecurity httpSecurity, JWTAuthorizationFilter
jwtAuthorizationFilter) throws Exception {
 return httpSecurity
 .csrf(AbstractHttpConfigurer::disable)
 .authorizeHttpRequests(authorize ->

 authorize.requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
 .requestMatchers(AUTH_WHITELIST).permitAll()
 .requestMatchers(EndpointRequest.toAnyEndpoint()).permitAll()
 .anyRequest().authenticated()
)
 .addFilterBefore(jwtAuthorizationFilter, UsernamePasswordAuthenticationFilter.class)
 .build();
 }

 @Bean
 public AuthenticationProvider noopAuthenticationProvider(){
 return new AuthenticationProvider() {
```

```

 @Override
 public Authentication authenticate(Authentication authentication) throws
AuthenticationException {
 return null;
 }

 @Override
 public boolean supports(Class<?> authentication) {
 return false;
 }
};

}

//Opinionated, but get out of the way quickly - custom defining encoder for PasswordEncoder bean
@Bean
public PasswordEncoder passwordEncoder() {
 return Pbkdf2PasswordEncoder.defaultsForSpringSecurity_v5_8();
}
}

```

-----

```

import bg.softuni.travelNestAccount.service.JWTService;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JWTAuthorizationFilter extends OncePerRequestFilter {

 private final JWTService jwtService;

 @Override
 protected void doFilterInternal(HttpServletRequest request,
 HttpServletResponse response,
 FilterChain filterChain) throws ServletException, IOException {

 final String authHeader = request.getHeader("Authorization");

 if (authHeader == null || authHeader.isBlank() || !authHeader.startsWith("Bearer ")) {
 filterChain.doFilter(request, response);
 return;
 }

 String jwtToken = authHeader.substring(7);
 UserDetails userDetails = jwtService.extractUserDetails(jwtToken);

 if (SecurityContextHolder.getContext().getAuthentication() == null) {
 SecurityContext context = SecurityContextHolder.getContext();

 UsernamePasswordAuthenticationToken authenticationToken = new

```

```

UsernamePasswordAuthenticationToken(
 userDetails, "", userDetails.getAuthorities());

 context.setAuthentication(authenticationToken);
 SecurityContextHolder.setContext(context);
 }

 filterChain.doFilter(request, response);
}
}

import bg.softuni.travelNestAccount.service.JWTService;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.nio.charset.StandardCharsets;
import java.security.Key;
import java.util.List;

@Service
public class JWTServiceImpl implements JWTService {

 private final String jwtSecret;

 public JWTServiceImpl(@Value("${jwt.secret}") String jwtSecret) {
 this.jwtSecret = jwtSecret;
 }

 @Override
 public UserDetails extractUserDetails(String jwtToken) {

 Claims claims = Jwts.parserBuilder()
 .setSigningKey(getSigningKey())
 .build()
 .parseClaimsJws(jwtToken)
 .getBody();

 List<String> roles = getRoles(claims);
 String username = getUsername(claims);

 return new User(username, "",
 roles.stream().map(SimpleGrantedAuthority::new).toList());
 }

 private String getUsername(Claims claims) {
 return claims.get("username", String.class);
 }

 @SuppressWarnings("unchecked")
 private List<String> getRoles(Claims claims) {
 return claims.get("roles", List.class);
 }

 private Key getSigningKey(){
 return Keys

```

```

 .hmacShaKeyFor(jwtSecret
 .getBytes(StandardCharsets.UTF_8));
 }
}

application.properties
server.port=8081

jwt.secret=${JWT_SECRET:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG
9IiwiWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c}

import bg.softuni.travelNestAccount.model.dto.AttractionDTO;
import bg.softuni.travelNestAccount.model.dto.TicketDTO;
import bg.softuni.travelNestAccount.model.enums.City;
import bg.softuni.travelNestAccount.service.AttractionService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;

import java.util.Arrays;
import java.util.List;
import java.util.UUID;

@RestController
@RequiredArgsConstructor
public class AttractionController {

 private final AttractionService attractionService;

 @GetMapping("/attraction/cities")
 public ResponseEntity<List<String>> getCities(){
 return ResponseEntity.ok(Arrays.stream(City.values())
 .map(City::toString)
 .toList());
 }

 @GetMapping("/{attraction-type}/list")
 public ResponseEntity<List<AttractionDTO>> getAttractions(@PathVariable("attraction-type") String attractionType){
 return ResponseEntity.ok(attractionService.getAllAttractions(attractionType));
 }

 @GetMapping("/attraction/details/{uuid}")
 public ResponseEntity<AttractionDTO> getAttractionDetails(@PathVariable("uuid") UUID attractionId){

 return ResponseEntity.ok(attractionService.getById(attractionId));
 }

 @GetMapping("/attraction/tickets/{uuid}")
 public ResponseEntity<TicketDTO> getTickets(@PathVariable("uuid") UUID attractionId,
 @AuthenticationPrincipal UserDetails userDetails){
 return ResponseEntity.ok(attractionService.getTickets(attractionId, userDetails));
 }
}

```

```

 @PostMapping("/attraction/details/{uuid}")
 public void buyTickets(@PathVariable("uuid") UUID attractionId,
 @RequestBody TicketDTO ticketDTO){

 attractionService.buyTickets(attractionId, ticketDTO);

 }

 @PostMapping("/{attraction-type}/add")
 public ResponseEntity<AttractionDTO> addAttraction(
 @PathVariable("attraction-type") String attractionType,
 @RequestBody AttractionDTO attractionDTO
){
 return ResponseEntity.ok(attractionService.createAttraction(attractionDTO, attractionType));
 }

 @DeleteMapping("/attraction/delete/{uuid}")
 public ResponseEntity<AttractionDTO> deleteAttraction(@PathVariable("uuid") UUID attractionId,
 @AuthenticationPrincipal UserDetails
userDetails){
 attractionService.deleteById(attractionId, userDetails);
 return ResponseEntity
 .noContent()
 .build();
 }
}

```

## XX. Other

```

 @PostMapping(value = "/users/register", consumes = MediaType.APPLICATION_FORM_URLENCODED_VALUE)
 public String doRegister(@Valid @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(@Valid @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);

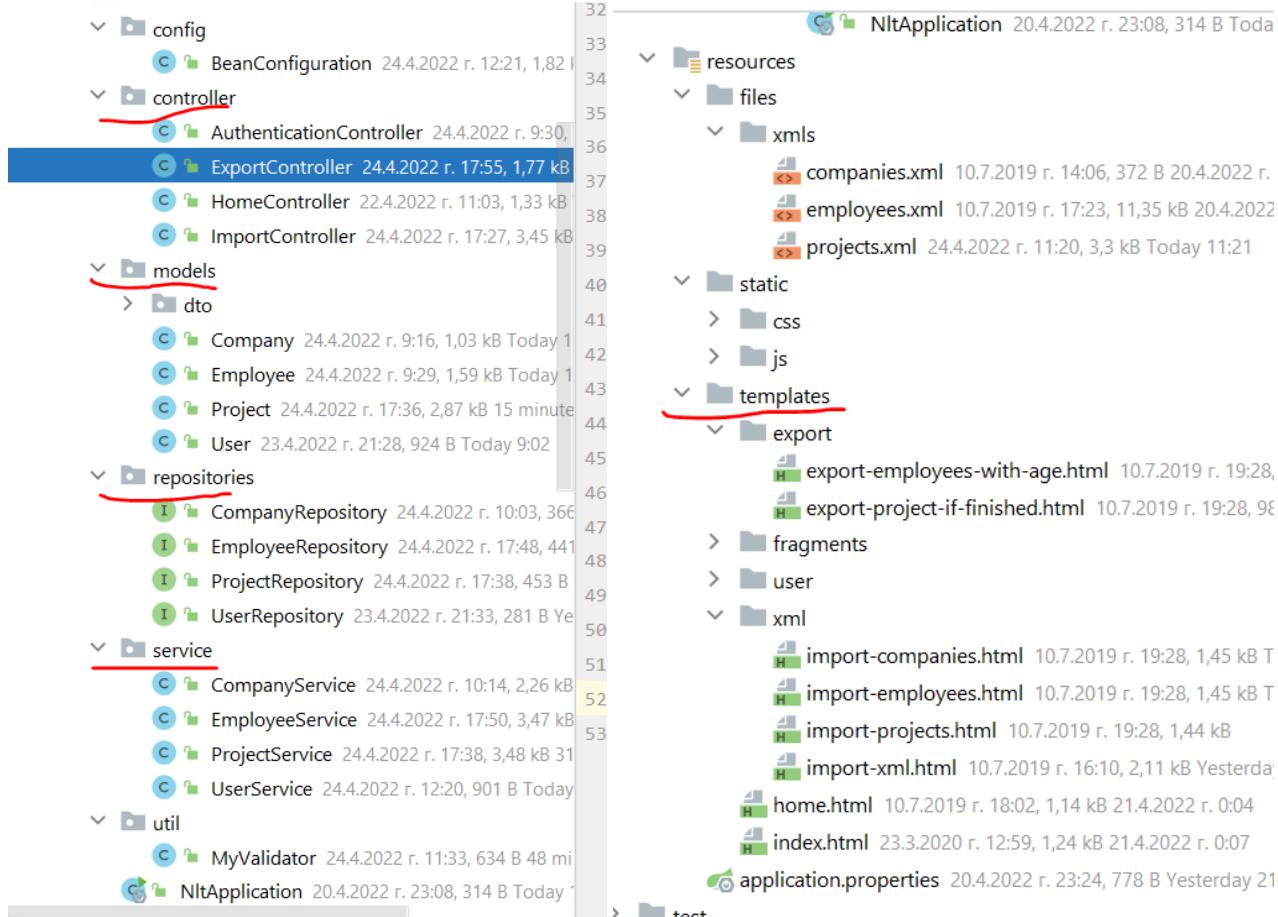
 return "user/login";
 }

```

## Logout с Rest

Трябва да си вземем и csrf 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;
}
}

```

▼ templates

- ▼ export
  - export-employees-with-age.html 10
  - export-project-if-finished.html 10.7.

```

<!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>

```

```
</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

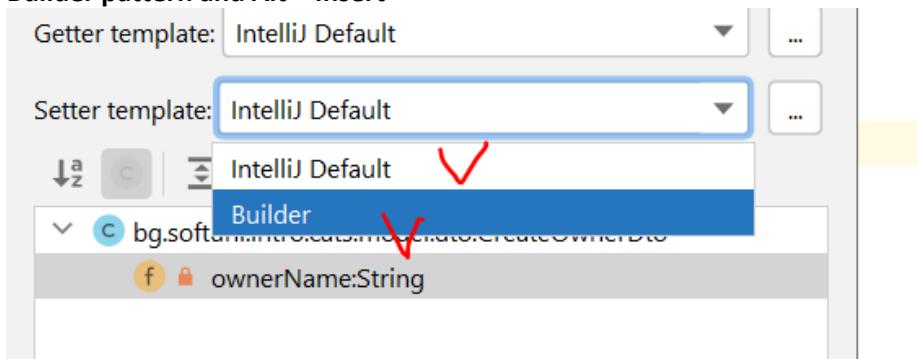
```
@PostMapping("/")
public String storeContact(Contact contact, @RequestParam("picture") MultipartFile picture) {
 this.contacts.add(contact);

 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 А-В 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)") //но 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

! + Tab

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 автоматично 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.)

<b>LocalStorage</b>	5MB/10MB storage It's not session based, need to be deleted via JS or manually Client side reading only Less older browsers support
<b>SessionStorage</b>	5MB storage It's session based and working per window or tab Client side reading only Less older browsers support
<b>Cookie</b>	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

	<b>Cookies</b>	<b>Local Storage</b>	<b>Session Storage</b>
<b>Capacity</b>	4kb	10mb	5mb
<b>Browsers</b>	HTML4 / HTML 5	HTML 5	HTML 5
<b>Accessible from</b>	Any window	Any window	Same tab
<b>Expires</b>	Manually set	Never	On tab close
<b>Storage Location</b>	Browser and server	Browser only	Browser only
<b>Sent with requests</b>	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 CLEARING/DELETING LIFETIME	SERVER & CLIENT SIDE PHP, JS, AUTOMATIC AS SPECIFIED	CLIENT SIDE ONLY JS ONLY TILL DELETED	CLIENT SIDE ONLY JS & AUTOMATIC 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.

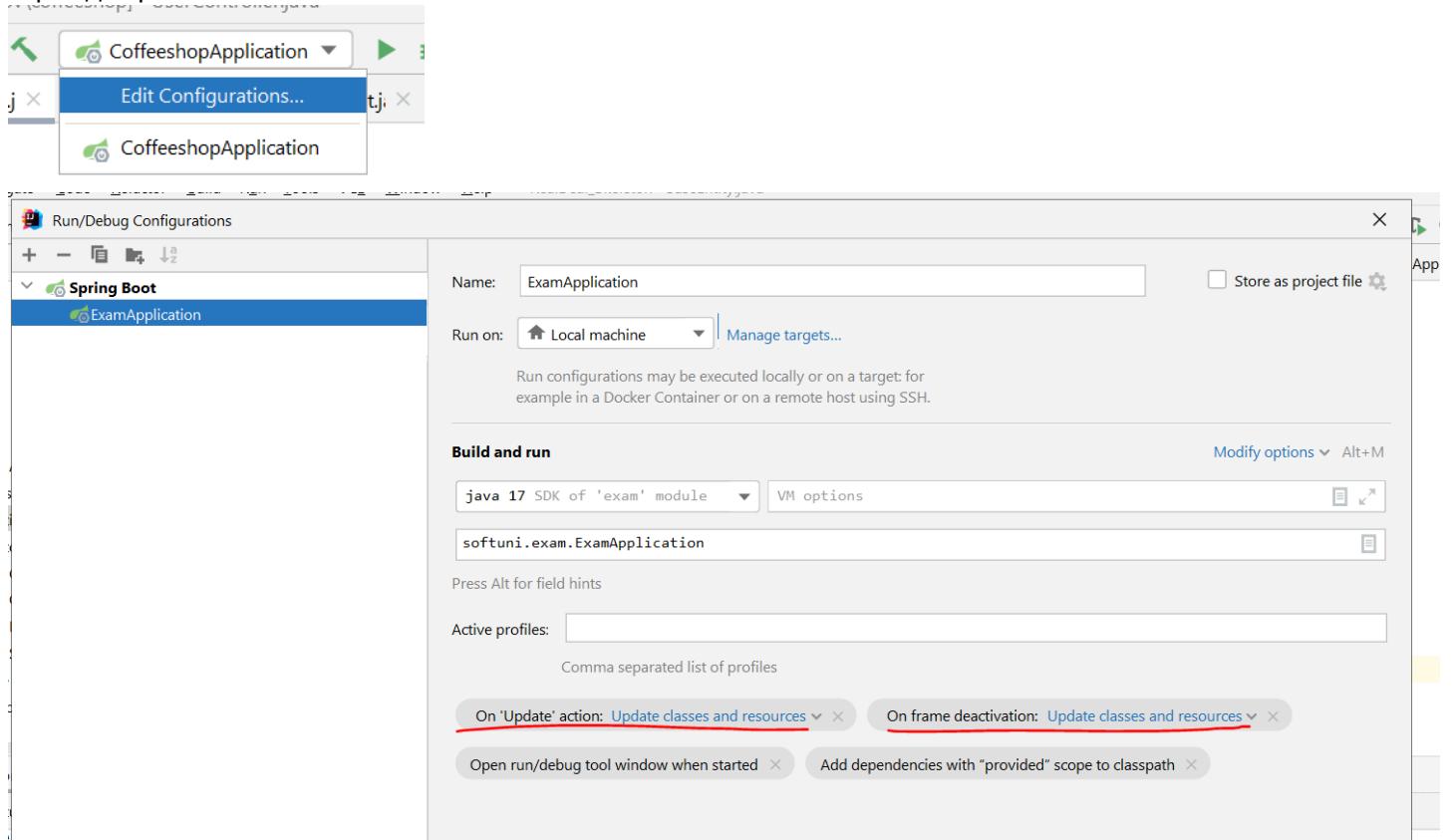
JS е нещото, което може да вмъкне json данни (дошли от http rest протокола) в html кода, без значение дали използваме thymeleaf или не.

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 front-енд кода особено) да зарежда проекта наново



## Настройки Thymeleaf

When copying manually templates in resources 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

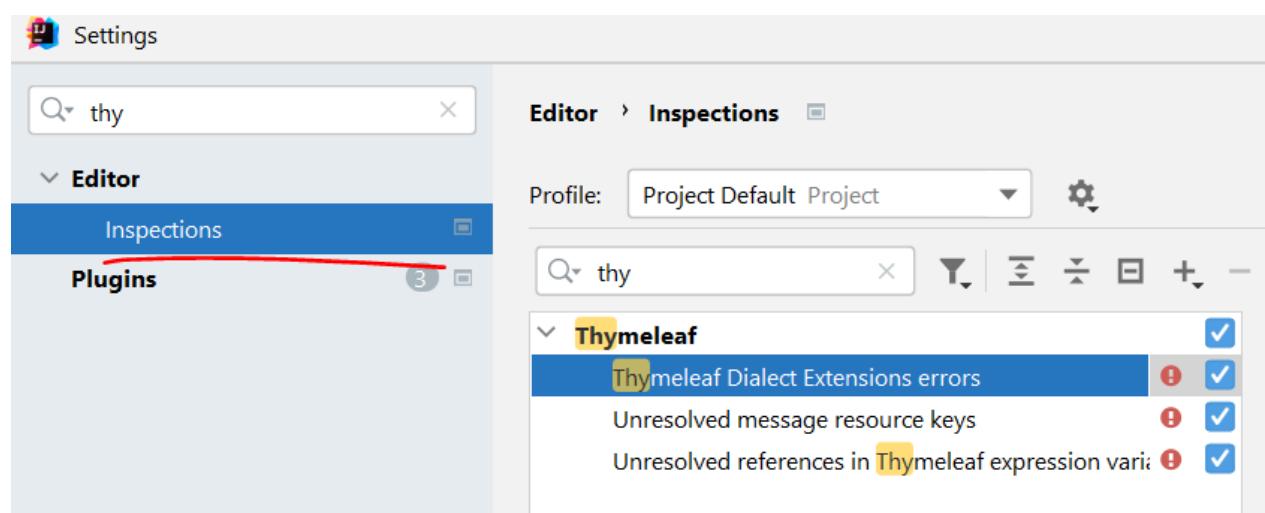
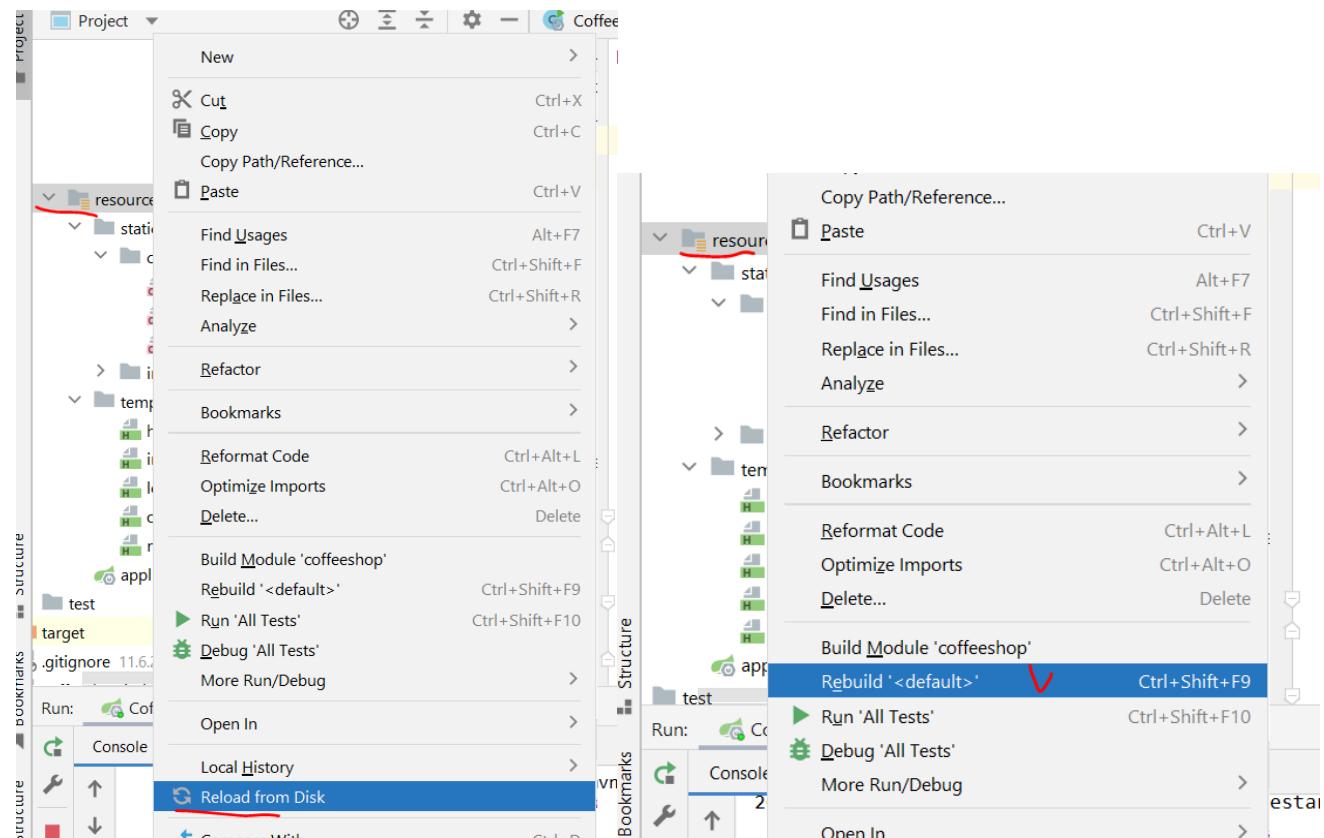
```
<parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>2.5.3</version>
 <relativePath/> <!-- Lookup parent from repository -->
</parent>
```

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'
}

```



## 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.

Client-Side vs. Server-Side Rendering - client-side първоначално зарежда някаква страница при браузъра на клиента и от нея се извикват rest заявки за допълнително зареждане и визуализиране на HTML код. Server-side изплюва крайната версия на страницата в браузъра на клиента - **при промяна трябва да се презареди цялата страница!**

### 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.