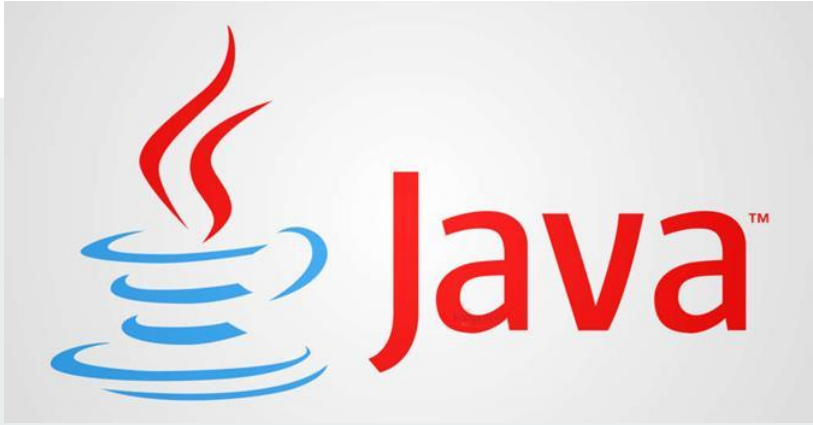


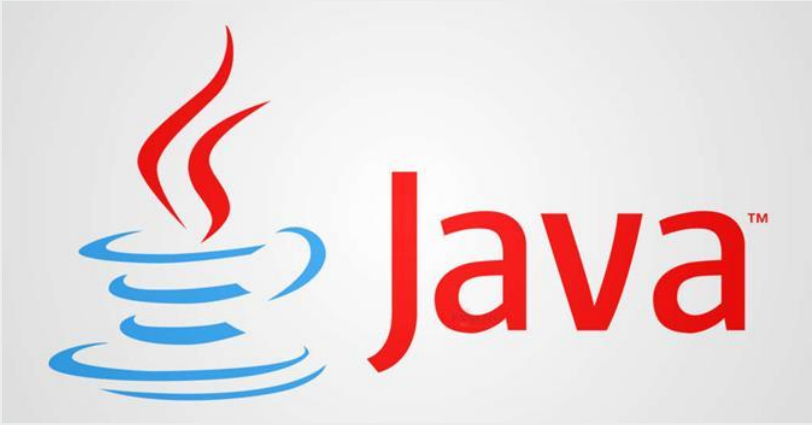
Programowanie aplikacji WWW w języku Java

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
NCBR
Narodowe Centrum Badań i Rozwoju

Materiały przygotowane w ramach projektu „PB 5.0 – dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji” (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
NCBR Narodowe Centrum Badań i Rozwoju

Wstęp Technologia serwletów



Programowanie aplikacji WWW w języku Java

Historia technologii serwletów

Servlet 1.0 (1997) - (pocz. James Gosling) Pavni Diwanji - Sun Microsystems (film 0-3:50)

Servlet 2.1 (1998) - kontener serwletów, komunikacja pomiędzy serwletami

Servlet 2.2 (1999) - rozszerzona komunikacja pomiędzy serwletami, obserwatorzy zdarzeń, pliki .war

Servlet 2.3 (2001) Java Community Process. Mechanizm filtrowania, integracja z J2EE

....

Servlet 3.1 (2013) - adnotacje, komunikacja asynchroniczna, bezpieczeństwo, ładowanie plików, nonblocking I/O, WebSocket (JSR 340)

Servlets 6.0

https://projects.eclipse.org/projects/ee4j.servlet

https://en.wikipedia.org/wiki/Jakarta_Servlet

Servlet API version	Released	Specification	Platform	Important Changes
Jakarta Servlet 6.0	Oct 15, 2021	6.0	Jakarta EE 10	remove deprecated features and implement requested enhancements
Jakarta Servlet 5.0	Oct 9, 2020	5.0	Jakarta EE 9	API moved from package `javax.servlet` to `jakarta.servlet`
Jakarta Servlet 4.0.3	Sep 10, 2019	4.0	Jakarta EE 8	Renamed from "Java" trademark

```
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
```

to

```
import jakarta.servlet.*;
import jakarta.servlet.annotation.*;
import jakarta.servlet.http.*;
```

Definicja serwletów

- klasy Javy uruchamiane po stronie serwera aplikacji
- służą do obsługi żądań GET, POST, PUT, DELETE protokołu HTTP
- serwlety mogą korzystać ze standardowych klas Javy oraz dowolnych bibliotek, klas wchodzących w skład Servlet API: javax.servlet.* i javax.servlet.http.*

```
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
```

to

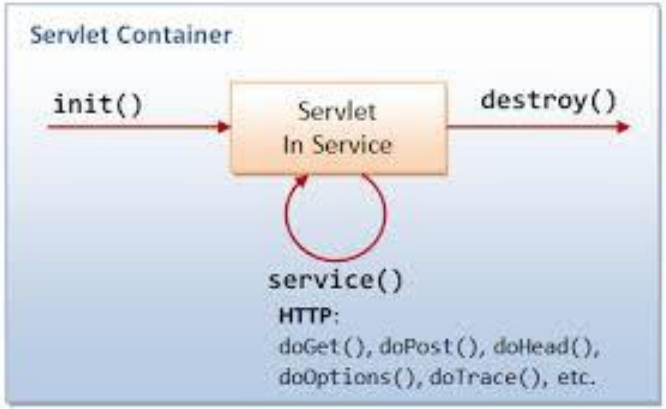
```
import jakarta.servlet.*;
import jakarta.servlet.annotation.*;
import jakarta.servlet.http.*;
```

Kod serwletu

```
public class MojSerwlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    }
```

Cykl życia serwletu

- kontrolowany przez **kontener**, w którym serwlet został osadzony (deployed)
- Obsługa żądania:
  - jeśli instancja serwletu jeszcze nie istnieje:
    - Ładuje klasę serwletu do pamięci
    - Tworzy instancję klasy serwletu
    - Wywołuje metodę `init()` serwletu
  - Wywołuje metodę `service()` do obsługi żądania
- Serwlet po zakończeniu obsługi żądania pozostaje aktywny w kontenerze
- Kolejne żądania realizowane są przez uruchomienie metody `service()` w osobnych wątkach
- Gdy kontener musi usunąć serwlet, najpierw wywołuje jego metodę `destroy()`



HttpServletRequest i HttpServletResponse

request.getHeader(String)

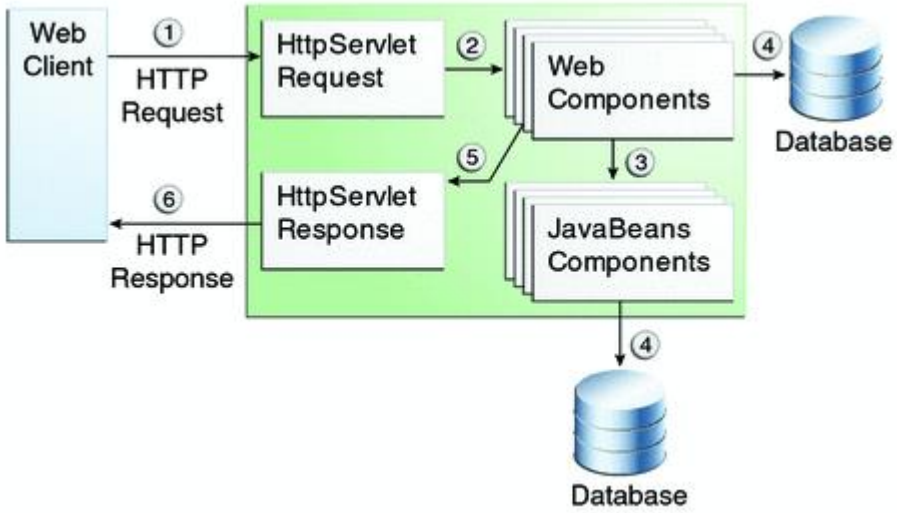
request.getReader()

request.getInputStream()

request.getMethod()

request.getParameter(String)

request.getSession()



response.addCookie(Cookie)

response.addHeader(String,String)

Parametry żądania

```
out.println("<h1>Servlet NewServlet at " + request.getContextPath() + "</h1>");
Enumeration<String> parameters=request.getParameterNames();
out.println("<ul>Parameters");
for(;parameters.hasMoreElements();){
    String par=parameters.nextElement();
    String parVal=request.getParameter(par);
    out.println("<li>"+par+"="+parVal+"</li>");
}
```

ServletContext

```
public class Counter extends HttpServlet {
    public void doGet(HttpServletRequest request,
    HttpServletResponse
    response) throws IOException, ServletException{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        ServletContext context = this.getServletContext();
        Counter counter =
        (Counter)context.getAttribute("hitCounter");
        out.println(”Jesteś ” + counter.getAndInc() +
        ” osobą na tej stronie”);
    }
}
```

Przekierowania

Jeśli chcemy przekierować zapytanie na zupełnie inną stronę, można użyć metody `ServletResponse.sendRedirect(String URL)`, która wyśle do przeglądarki prośbę o przekierowanie

Jeśli robimy przekierowanie wewnątrz własnego serwisu, zwykle lepiej jest użyć klasy `RequestDispatcher`, która przekazuje prośbę wygenerowania strony dalej

Adnotacje (@WebServlet)

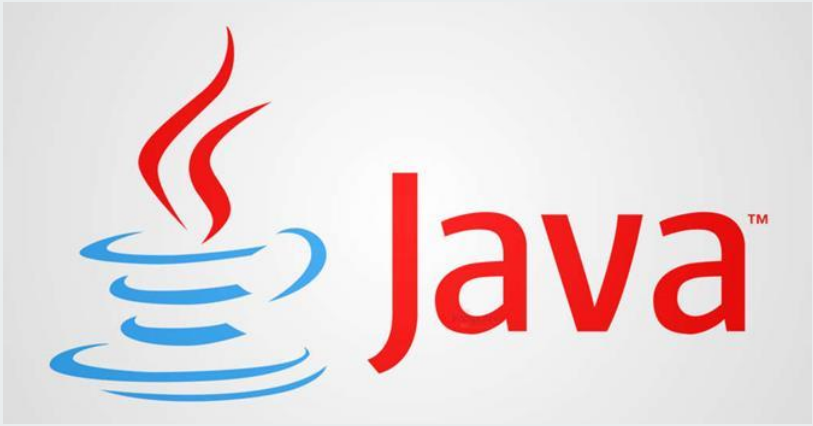
Attributes

Name	Type	Required	Description
value or **urlPatterns**	*String[]*	Required	Specify one or more URL patterns of the servlet. Either of attribute can be used, but not both.
name	*String*	Optional	Name of the servlet
displayName	*String*	Optional	Display name of the servlet
description	*String*	Optional	Description of the servlet
asyncSupported	*boolean*	Optional	Specify whether the servlet supports asynchronous operation mode. Default is false.
initParams	*WebInitParam[]*	Optional	Specify one or more initialization parameters of the servlet. Each parameter is specified by `@WebInitParam` annotation type.
loadOnStartup	*int*	Optional	Specify load-on-startup order of the servlet.
smallIcon	*String*	Optional	Specify name of the small icon of the servlet.
largeIcon	*String*	Optional	Specify name of the large icon of the servlet.

Adnotacje

```
@WebServlet("/myurl")

    public class TestServlet extends
javax.servlet.http.HttpServlet {

        ....

    }
```

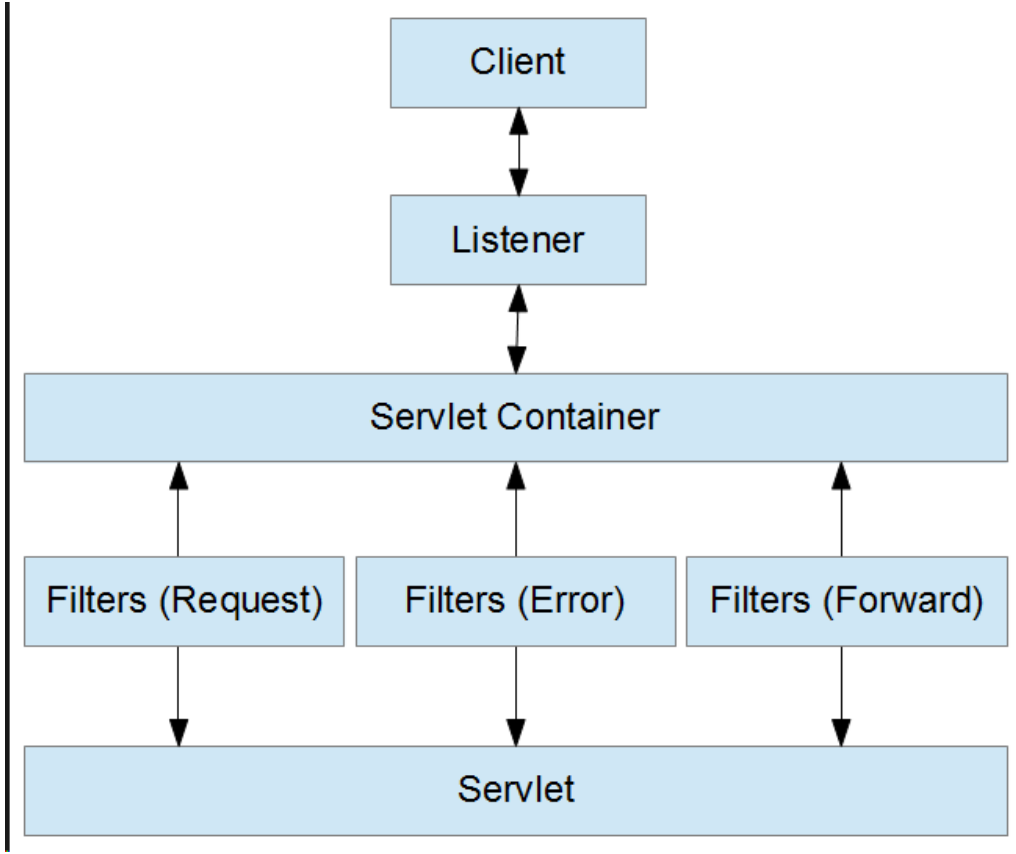
Model zdarzeń



Programowanie aplikacji WWW w języku Java

Elementy modelu zdarzeń

- Klasy nasłuchujące
- Filtry



Obiekty nasłuchujące

- Obiekty nasłuchiwania kontekstu serwletu
- Obiekty nasłuchiwania atrybutów kontekstu serwletu
- Obiekty nasłuchiwania sesji
- Obiekty nasłuchiwania atrybutów sesji
- Obiekty nasłuchiwania migracji sesji
- Obiekty nasłuchiwania wiązania obiektów sesji
- Obiekty nasłuchiwania żądań
- Obiekty nasłuchiwania atrybutów żądań

Obiekty nasłuchujące – schemat korzystania

1. Zaimplementować odpowiedni interfejs
2. Zaimplementować wymagane metody obsługujące związane z interfejsem zdarzenia
3. Uzyskać dostęp i wykorzystać istotne obiekty aplikacji
4. Zadeklarować (np. w web.xml) obiekt nasłuchujący
5. Zdefiniować wymagane parametry inicjalizacyjne

Przykład 1

Problem:

Aplikacja powinna działać w wybranych konfiguracjach bazy:

- testowej przy niewielkich zasobach systemowych,
- produkcyjnej przy wysokich zasobach systemowych

Przeniesienie aplikacji pomiędzy systemami nie powinno wymagać rekompilacji kodu.

Przykład 1

Rozwiązanie: ServletContextListener+parametry inicjalizacyjne aplikacji

```
@WebListener

public class MyContextListener implements ServletContextListener
{
public void contextInitialized(ServletContextEvent event) {
    ServletContext sc=event.getServletContext();
    String initVal=sc.getInitParameter(„db_name");
    DBSettings db=new DBSettings(initVal);
    sc.setAttribute(„db_settings", db);
}
```

Przykład 1

```
public class MyContextListener implements ServletContextListener
{
public void contextDestroyed(ServletContextEvent arg0) {
     ServletContext sc=event.getServletContext();
     DBSettings db=sc.getAttribute(„db_settings");
     db.closeConnection();
}
```

Przykład 2

Problem:

Aplikacja dla sklepu, w której jest możliwość dodawania produktów do koszyka. W przypadku, gdy klient usunie produkt z koszyka bez zakupienia go, należy zapisać ten fakt w bazie w tabeli „Porzucone”.

Przykład 2

Rozwiązanie: HttpSessionAttributeListener

public void attributeAdded(HttpSessionBindingEvent)

public void attributeRemoved(HttpSessionBindingEvent)

public void attributeReplaced(HttpSessionBindingEvent)

Przykład 2

```
@WebListener
public class SaveAbandonedBasket implements
HttpSessionAttributeListener{
Basket basket;

public void attributeReplaced(HttpSessionBindingEvent
event) {
     if(event.getName().equals(„basket”){
          Basket newBasket=(Basket)event.getValue();
     ...
     }
}
```

Przykład 3

Problem:

Jakie jest obciążenie serwera aplikacji pod względem liczby żądań i czasu obsługi żądania? Rejestracja zbyt długich żądań.

Przykład 3

Rozwiązanie: ServletRequestListener

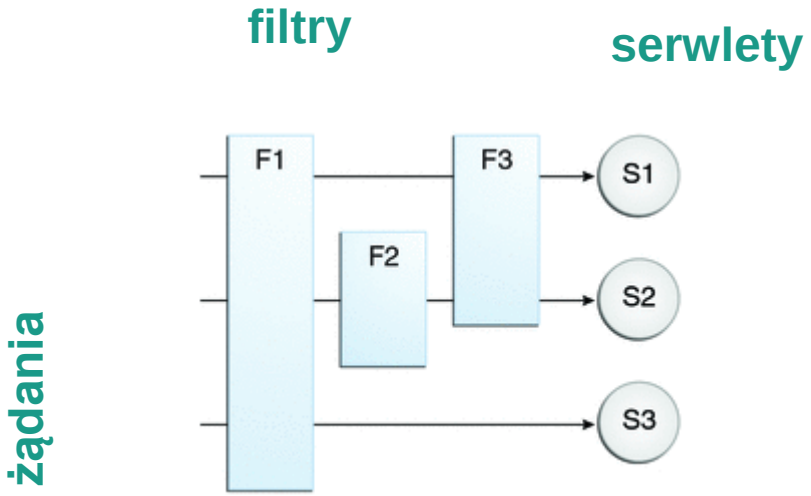
public void requestInitialized(ServletRequestEvent)

public void requestDestroyed(ServletRequestEvent)

Przykład 3

```
@WebListener
public class RequestCounter implements ServletRequestListener{
	private ArrayList<Long> requestList=new ArrayList<>();

public void requestInitialized(ServletRequestEvent event){
    recordRequest(event,System.currentTimeMilis());
}

public void requestDestroyed(ServletRequestEvent event) {
	countTime(event, System.currentTimeMilis());
}
```

Filtry



Filtry

Po odczytaniu informacji z żądania można wykonać następujące czynności:

- wywołać zasób w standardowy sposób

- wywołać zasób przekazując do niego zmienione dane żądania

- wywołać zasób i zmodyfikować dane odpowiedzi do klienta

- zapobiec wywołaniu zasobu i w zamian przekierować żądanie do innego zasobu

Zastosowanie

1) Uwierzytelnianie
2) Autoryzacja dostępu do zasobów
3) Przetwarzanie multimediów
4) Kompresja
5) Szyfrowanie
6) Przetwarzanie parametrów żądania i zawartości odpowiedzi

Filtry

Korzyści:

- wykonywanie operacji wspólnych w jednym miejscu

- separacja kodu zarządzającego dostępem od pozostałych warstw

- możliwość wprowadzania zmian jednocześnie do wielu różnych zasobów

Filtry

Czynności:

- implementacja klasy, która implementuje interfejs Filter oraz trzy metody doFilter, init, destroy

- uzupełnienie kodu metody doFilter

Metoda doFilter

```
public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain)
```

- Chain.doFilter(request,response)
- sendRedirect/RequestDispatcher
- response.setContentType("text/html");

```
PrintStream ps = new PrintStream(response.getOutputStream());
PrintWriter pw = new PrintWriter(ps);
pw.print("<html>\n<head>\n<title>Error</title>\n</head>\n<body>\n");
```

Adnotacje

```
@WebFilter(filterName = "NewFilter2", urlPatterns =
{"/test/*"})
```

```
@WebFilter(filterName = "NewFilter2", urlPatterns =
{"/test/*"}, servletNames = {"NewServlet3"})
```

```
@WebFilter(filterName = "NewFilter2", urlPatterns =
{"/test/*"}, servletNames = {"NewServlet3"}, initParams
= @WebInitParam(name="p1", value="value1"))
```

Przykład - zliczanie odwołań do poszczególnych zasobów aplikacji

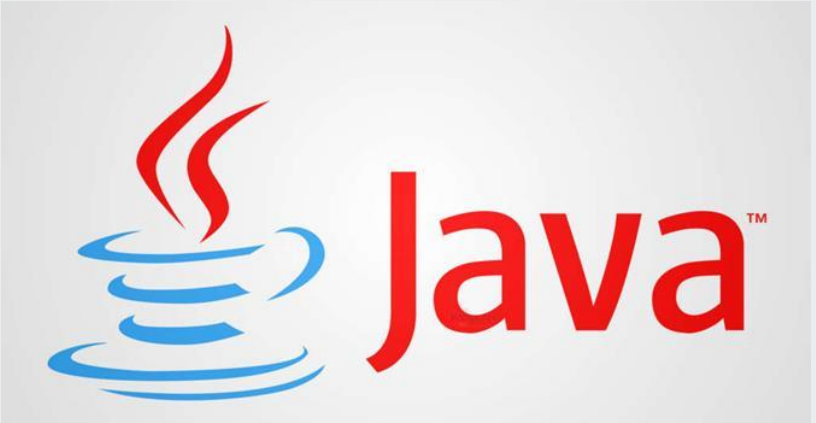
```
public void doFilter(ServletRequest request, ServletResponse
response,FilterChain chain)
throws IOException, ServletException {

try {
String link = ((HttpServletRequest)
request).getRequestURL().toString();
int count = 0;
if (links.containsKey(link)) {
        count = Integer.parseInt(links.get(link).toString());
}
```

Przykład - zliczanie odwołań do poszczególnych zasobów aplikacji

```
links.put(link, Integer.valueOf(++count));
Iterator i = links.keySet().iterator();
while (i.hasNext()) {
        link = i.next().toString();
        count = Integer.parseInt(links.get(link).toString());
        String str = link + " requested " + count + " times";
        this.getFilterConfig().getServletContext().log(str);
}
chain.doFilter(request, response);
} catch (Throwable t) {
problem = t;t.printStackTrace(); }
```

Współużytkowanie danych

Programowanie aplikacji WWW w języku Java



Metody współużytkowania danych

- cookies
- atrybuty kontekstu
- sesje

Cookies

1. Utrzymywane przez przeglądarkę
2. Do ciasteczek dostęp mamy poprzez klasę Cookie
3. HttpServletRequest.getCookies() zwraca tablicę obiektów klasy cookie
4. HttpServletResponse.addCookie(Cookie) dodaje ciasteczko

Kodowanie wartości ciasteczek

```
Cookie cookie = new Cookie("cookie21", URLEncoder.encode("short-time
living cookie", "UTF-8" ));


URLDecoder.decode(cookie.getValue(),"UTF-8")
```

Cookies

- zapis

```
//utworzenie ciasteczka o nazwie 'login' i wartości 'Ala'
Cookie cookie=new Cookie("login","Ala");


cookie.setMaxAge(1*60);//okres ważności ciasteczka w s,
//wartość "0" - usunięcie ciasteczka, wartość "-1" - czas
//ważności: do zamknięcia przeglądarki


//umieszczenie ciasteczka w obiekcie odpowiedzi
response.addCookie(cookie);
```

Cookies

- odczyt

```
//pobranie tablicy z ciasteczkami
Cookie[] cookies=request.getCookies();

//przeglądanie tablicy z ciasteczkami
for(int i=0;i<cookies.length;i++){
	Cookie cookie=cookies[i];
	if(cookie.getName().equals("login")){
		login=cookie.getValue();
		out.println("Jestes zalogowany
"+login+"<br>");
		break;
```

Sesje

1. Dane sesji przechowywane są po stronie serwera
2. Identyfikatory sesji przechowywane są w ciasteczkach o nazwie JSESSIONID po stronie klienta
3. Gdy klient ma wyłączoną obsługę ciasteczek, identyfikator sesji przekazujemy za pomocą mechanizmu GET (tzw. przepisywanie adresu URL - ang. URL rewriting).
4. Czas, po jakim sesja wygasa definiuje się w pliku `web.xml` (w minutach) lub programowo metodą `getMaxInactiveInterval()` (w sekundach)

Sesje

1. Sesja jest reprezentowana przez obiekt klasy `HttpSession`
2. Obiekt otrzymujemy wywołując metodę `getSession()` na obiekcie request
3. Atrybuty zapisujemy metodą `setAttribute(String, Object)`, i pobieramy metodą `getAttribute(String)`
4. `invalidate()` – jawne zniszczenie sesji
5. Inne informacje: `getAttributeNames()`, `getLastAccessedTime()`,

Sesje

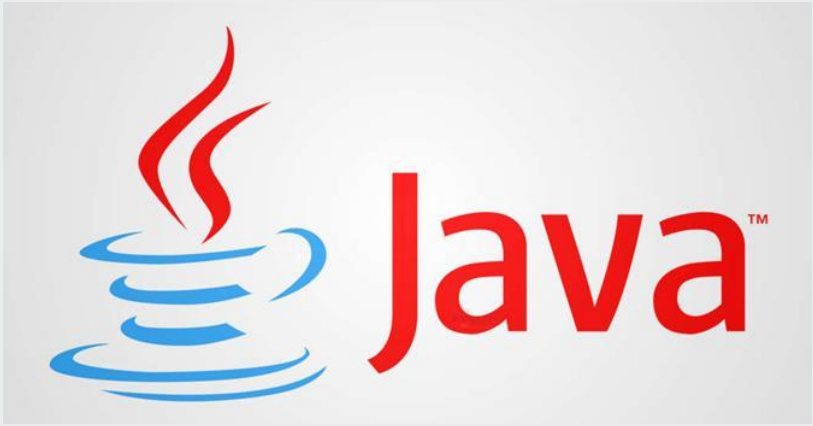
- Współużytkowane przez aplikację i poszczególne przeglądarki

```
HttpSession session=request.getSession();
if(session.isNew()){
        synchronized(session){
          session.setAttribute("a1 sesji", "16");
        }
        KlasaAtrybutu obAtrybutu=new KlasaAtrybutu(„MyAtt");
        synchronized(session){
          session.setAttribute("a4 sesji",obAtrybutu);
        }
}else session.invalidate();
```

Podsumowanie

- **Cookies:**
  - możliwość zablokowania/usunięcia w przeglądarce
  - Ograniczenie wielkości danych po stronie przeglądarki – każdy wpis do 4kB (przeważnie), 20 ciasteczek na domenę, 300 ciasteczek w sumie (szczegółowe informacje tu)
- **Kontekst**
  - Współużytkowane w obrębie aplikację
- **Sesje**
  - Współużytkowane pomiędzy poszczególnymi klientami a serwerem

Protokół HTTP



Programowanie aplikacji WWW w języku Java

Elementy protokołu

- Służy do transportu dokumentów udostępnianych przez serwery HTTP
- Tekstowy oparty na protokole TCP, domyślny port 80
- Model klient-serwer, styl żądanie-odpowiedź
- Po dostarczeniu dokumentu połączenie jest zamykane

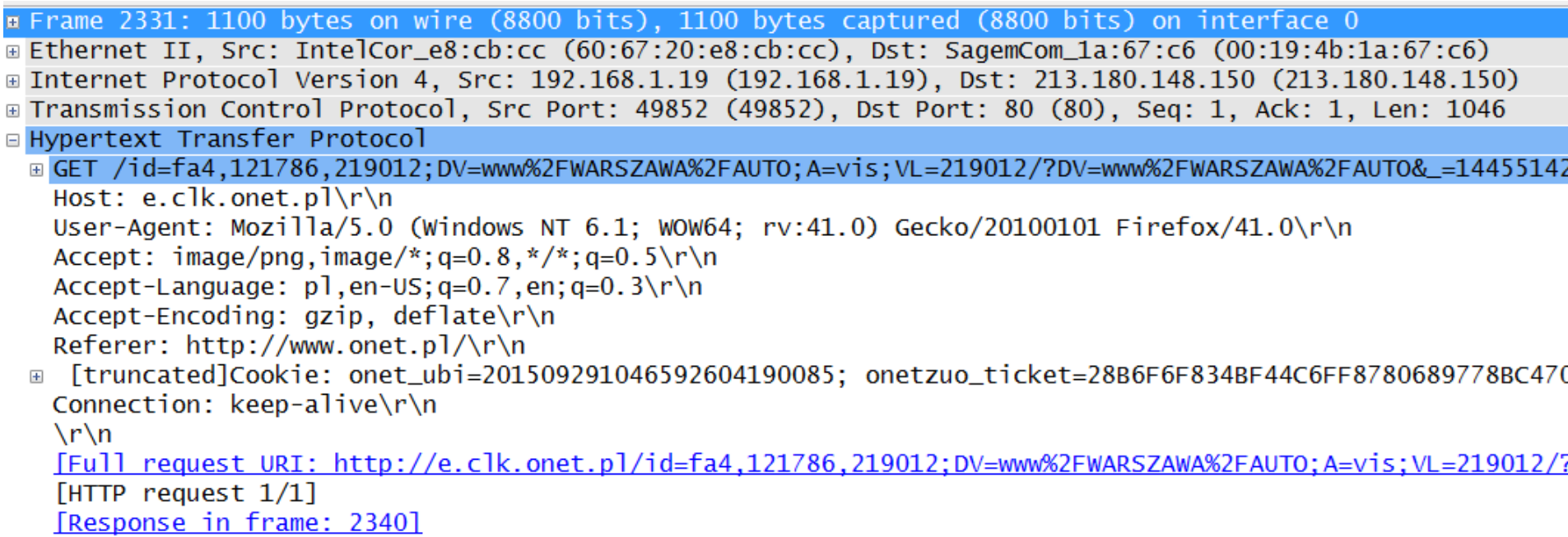
Żądanie i odpowiedź

Format żądania i odpowiedzi są podobne. Oba rodzaje komunikatów składają się z:

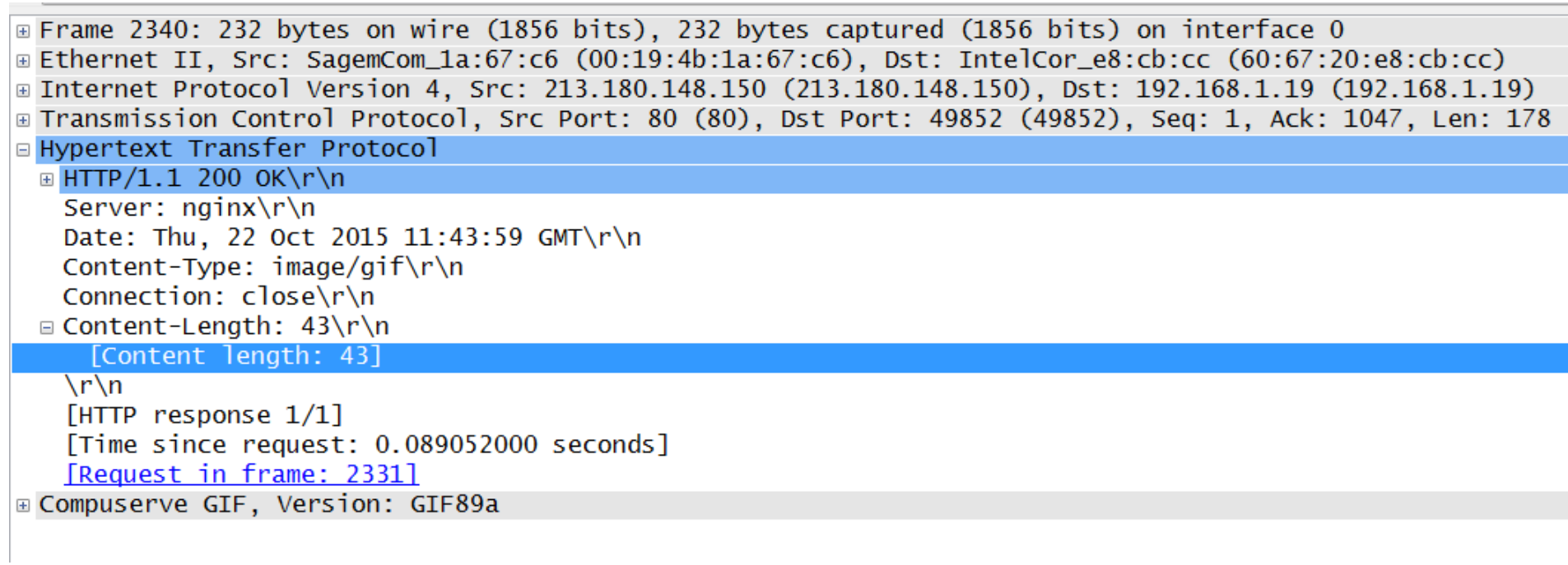
- linii początkowej,
- zera lub większej liczby linii nagłówka,
- pustej linii (tj. samo CRLF), i
- opcjonalnie treści wiadomości (np. plik lub zapytanie o dane, czy o wyjście).

Linia początkowa i nagłówki powinny się kończyć parą znaków CRLF.

Żądanie


Frame 2331: 1100 bytes on wire (8800 bits), 1100 bytes captured (8800 bits) on interface 0
Ethernet II, Src: IntelCor_e8:cb:cc (60:67:20:e8:cb:cc), Dst: SagemCom_1a:67:c6 (00:19:4b:1a:67:c6)
Internet Protocol Version 4, Src: 192.168.1.19 (192.168.1.19), Dst: 213.180.148.150 (213.180.148.150)
Transmission Control Protocol, Src Port: 49852 (49852), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 1046
Hypertext Transfer Protocol
GET /id=fa4,121786,219012;DV=www%2FWARSZAWA%2FAUTO;A=vis;VL=219012/?DV=www%2FWARSZAWA%2FAUTO&_=1445514
Host: e.clk.onet.pl\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0\r\n
Accept: image/png,image/*;q=0.8,*/*;q=0.5\r\n
Accept-Language: pl,en-US;q=0.7,en;q=0.3\r\n
Accept-Encoding: gzip, deflate\r\n
Referer: http://www.onet.pl/\r\n
[truncated]Cookie: onet_ubi=20150929104659260419008 5; onetzuo_ticket=28B6F6F834BF44C6FF8780689778BC47
Connection: keep-alive\r\n
\r\n
[Full request URI: http://e.clk.onet.pl/id=fa4,121786,219012;DV=www%2FWARSZAWA%2FAUTO;A=vis;VL=219012/?
[HTTP request 1/1]
[Response in frame: 2340]

Odpowiedź


Frame 2340: 232 bytes on wire (1856 bits), 232 bytes captured (1856 bits) on interface 0
Ethernet II, Src: SagemCom_1a:67:c6 (00:19:4b:1a:67:c6), Dst: IntelCor_e8:cb:cc (60:67:20:e8:cb:cc)
Internet Protocol Version 4, Src: 213.180.148.150 (213.180.148.150), Dst: 192.168.1.19 (192.168.1.19)
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 49852 (49852), Seq: 1, Ack: 1047, Len: 178
Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Server: nginx\r\n
Date: Thu, 22 Oct 2015 11:43:59 GMT\r\n
Content-Type: image/gif\r\n
Connection: close\r\n
Content-Length: 43\r\n
[Content length: 43]
\r\n
[HTTP response 1/1]
[Time since request: 0.089052000 seconds]
[Request in frame: 2331]
Compuserve GIF, Version: GIF89a

Komunikaty HTTP

Nagłówek

Linie nagłówka dostarczają informacji na temat żądania lub odpowiedzi, albo o obiekcie wysłanym w treści wiadomości.

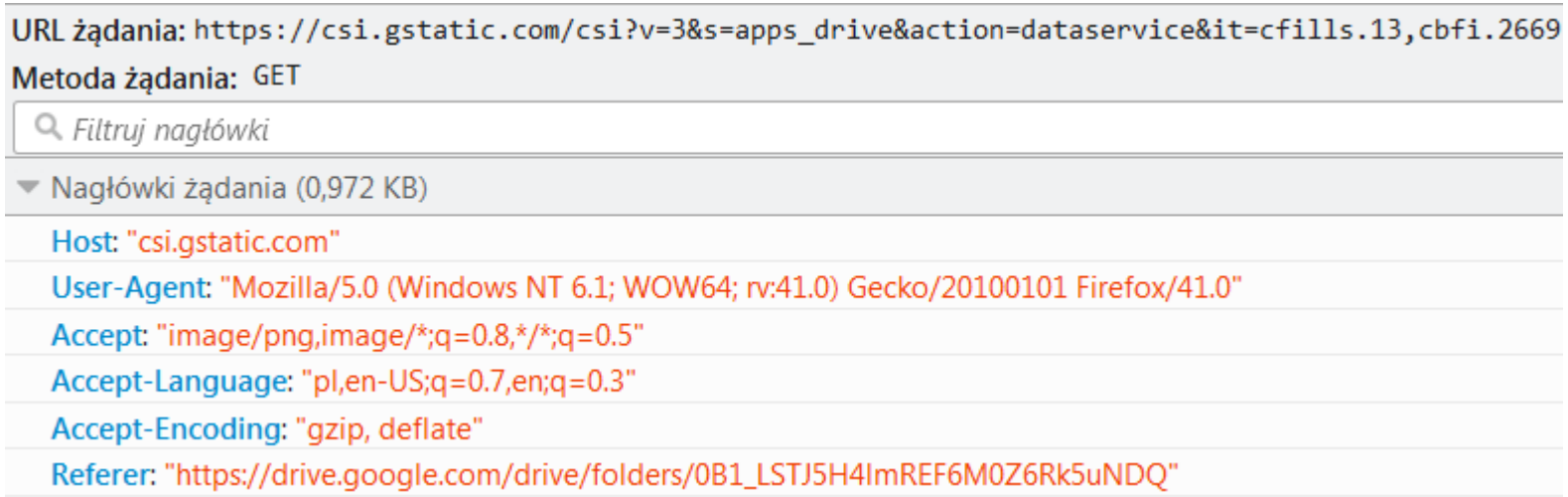
Linie nagłówka są w zwykłym formacie tekstowym w formie: „nagłówek-nazwa: wartość”, zakończone parą znaków CRLF.

W nazwie nagłówka nie jest rozróżniana wielkość liter

Pomiędzy znakiem „:” i wartością może być dowolna liczba spacji i tabulacji.

Linie nagłówka rozpoczynające się od spacji lub tabulacji są w rzeczywistości częścią poprzedniej linii nagłówka, przenoszone do następnej linii w celu łatwiejszego odczytywania.

Pola nagłówkowe



Nagłówki ze strony klienta

Host: host.com (wymagany w HTTP 1.1 nagłówek Host służący do rozpoznania hosta, jeśli serwer na jednym IP obsługuje kilka VirtualHostów)
User-Agent: nazwa aplikacji klienckiej
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8 (akceptowane (bądź nieakceptowane dla q=0) przez klienta typy plików)
Accept-Language: pl,en-us;q=0.7,en;q=0.3 (preferowany język strony – nagłówek przydatny przy Language negotiation)
Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7 (preferowane kodowanie znaków)
Keep-Alive: 300 (czas, jaki klient chce zarezerwować do następnego zapytania w przypadku połączenia Keep-Alive)
Connection: keep-alive (chęć nawiązania połączenia stałego Keep-Alive z serwerem HTTP/1.0)
HTTP/1.1 dopuszcza wysłanie kilku żądań naraz (pipelining). HTTP/1.0 zakłada jedno żądanie i jedną odpowiedź.

Priorytety na listach wartości

```
Accept: application/xhtml+xml, application/xml;q=0.9,
text/xml;q=0.7, text/html;q=0.5, text/plain;q=0.3
```

Liczby podane po ;q= powinny mieć wartości od 0 do 1, co 0.1. Jako separator miejsc dziesiętnych użyta musi być kropka. W przypadku braku zdefiniowanej wartości ;q= przyjmowana jest wartość: ;q=1 (równoznaczna z ;q=1.0).

Powyższy przykład ustala najwyższy priorytet typowi application/xhtml+xml. Jeżeli **serwer** nie posiada dokumentu w formacie **application/xhtml+xml**, to prześle plik w formacie application/xml – ogólnym dla podjęzyków **XML**, takich jak **XHTML**, **SVG**. Jeżeli na **serwerze** nie ma również dokumentu w tym formacie, dane zostaną przesłane jako text/xml – czyli ogólne dane w formacie **XML**, w ostateczności, jeżeli żadnego z tych typów nie posiada, **serwer** wyśle dane w formacie text/plain, czyli zwykłego tekstu.

Nagłówki serwera

Date: Thu, 20 Dec 2001 12:04:30 GMT (czas serwera)
Server: Apache/2.0.50 (Unix) DAV/2 (opis aplikacji serwera)
Set-Cookie: nakazanie klientowi zapisania ciasteczka
Expires: *czas wygaśnięcia zawartości zwróconego dokumentu. Data w przeszłości zabrania umieszczenie dokumentu w pamięci podręcznej.*
Cache-Control: no-store, no-cache, must-revalidate
Pragma: *no-cache (informacje dotyczące zapisywania zawartości w pamięci podręcznej. Stara, niestandardowa metoda.)*
Keep-Alive: timeout=15, max=100
Connection: *Keep-Alive (akceptacja połączenia Keep-Alive dla klientów HTTP/1.0)*
Transfer-Encoding: chunked, identity, gzip (sposób przesłania strony: w częściach, skompresowane)
Content-Type: application/xhtml+xml; charset=utf-8 (typ MIME i strona kodowa zwróconego dokumentu)
http://www.rexswain.com/httpview.html

Nagłówki | Ciasteczka | Parametry | Odpowiedź | Pomiary czasu | Podgląd

URL żądania: `http://wi.pb.edu.pl/`
Metoda żądania: `GET`
Zdalny adres: `212.33.90.174:80`
Kod stanu: ▲ `303 See other`
Wersja: `HTTP/1.1`

Edytuj i wyślij ponownie | Nieprzetworzone nagłó

Filtruj nagłówki

Nagłówki odpowiedzi (0,373 KB)

Connection: "Keep-Alive"
Content-Length: "0"
Content-Type: "text/html; charset=utf-8"
Date: "Thu, 22 Oct 2015 12:30:52 GMT"
Keep-Alive: "timeout=5, max=100"
Location: "http://wi.pb.edu.pl/index.php/pl/"
Server: "Apache/2.4.6 (CentOS) PHP/5.4.16"
Set-Cookie: "cd5e1a0dfb0534c5ac10a4aa1cc94e35=inc157g0ntdmt5k0q0ahnbupn4; path=/; HttpOnly"
X-Powered-By: "PHP/5.4.16"

Nagłówki żądania (0,480 KB)

Host: "wi.pb.edu.pl"
User-Agent: "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0"
Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
Accept-Language: "pl,en-US;q=0.7,en;q=0.3"
Accept-Encoding: "gzip, deflate"
Cookie: "cookieControl=1; __utma=99137304.959685687.1444246906.1444246906....t)|utmccn=(direct)|utmcmd=(none); _ga=GA1.3.116816906.144454
Connection: "keep-alive"

Metody protokołu HTTP

Metoda	Opis
DELETE	Żądanie usunięcia zasobu (dokumentu) z serwera
GET	Żądanie zasobu od serwera w formie nagłówka i treści
HEAD	Żądanie zasobu od serwera w formie nagłówka
LINK	Żądanie ustanowienia relacji między istniejącymi zasobami
OPTIONS	Żądanie od serwera identyfikacji obsługiwanych metod
POST	Żądanie odebrania przez serwer danych od klienta
PUT	Żądanie odebrania przez serwer od klienta pliku
TRACE	Żądanie zwrócenia przez serwer nagłówków wiadomości wysłanej od klienta (w celach testowania)
UNLINK	Żądanie usunięcia relacji między istniejącymi zasobami

- GET – pobranie zasobu wskazanego przez URI, może mieć postać warunkową jeśli w nagłówku występują pola warunkowe takie jak "If-Modified-Since"
- HEAD – pobiera informacje o zasobie, stosowane do sprawdzania dostępności zasobu
- PUT – przyjęcie danych przesyłanych od klienta do serwera, najczęściej aby zaktualizować wartość encji
- POST – przyjęcie danych przesyłanych od klienta do serwera (np. wysyłanie zawartości formularzy)
- DELETE – żądanie usunięcia zasobu, włączone dla uprawnionych użytkowników
- OPTIONS – pozwala klientowi ustalić opcje i/lub wymagania związane z danym zasobem, albo możliwości danego serwera, nie implikując działań zasobu i nie inicjując pobierania zasobu
- TRACE – wywołuje zdalną pętlę zwrotną komunikatu żądania. Ostateczny odbiorca żądania odbija otrzymany komunikat z powrotem do klienta. TRACE pozwala klientowi zobaczyć co jest odbierane na drugim końcu łańcucha żądania. Informacja ta może być wykorzystywana do testowania, lub znajdywania uszkodzeń.
- CONNECT – nazwa metody zarezerwowana do wykorzystywania wraz z proxy, który może dynamicznie przełączyć się na pełnienie funkcji tunelu, przy użyciu, na przykład, tunelowania z wykorzystaniem warstwy zabezpieczeń łączy (SSL).

- `$ curl -X OPTIONS -i http://wi.pb.edu.pl`

  ```
  HTTP/1.1 200 OK
  Server: Apache
  Allow: HEAD,GET,PUT,DELETE,OPTIONS
  Content-Length: 0
  ```

- `$ curl -i -X TRACE -H "test: abc" http://localhost/`

  ```
  HTTP/1.1 200 OK
  Date: Sat, 20 Dec 2014 02:32:12 GMT
  Server: Apache
  Transfer-Encoding: chunked
  Content-Type: message/http

  TRACE / HTTP/1.1
  User-Agent: curl/7.28.1
  Host: localhost
  Accept: */*
  test: abc
  ```

Kody stanu protokołu HTTP

Wartość	Rodzaj	Rodzaj ang.	Znaczenie
1xx	Informacje	*Informational*	Żądanie odebrane, proces w trakcie wykonywania
2xx	Sukces	*Success*	Akcja została poprawnie odebrana, zrozumiana i zaakceptowana
3xx	Przekierowanie	*Redirection*	Dalsze akcje muszą zostać podjęte, aby zakończyć żądanie
4xx	Błąd po stronie klienta	*Client Error*	Żądanie ma złą składnię lub nie może zostać spełnione
5xx	Błąd serwera	*Server Error*	Serwer najprawdopodobniej nie może wykonać poprawnego żądania

Kod stanu	Komunikat	Kod stanu	Komunikat
100	Kontynuuj	404	Nie znaleziony
101	Przełączanie protokołów	405	Metoda niedozwolona
200	OK	406	Nie do przyjęcia
201	Utworzony	407	Wymagane uwierzytelnienie proxy
202	Przyjęty	408	Przeterminowanie żądania
203	Nie autorytatywny	409	Konflikt
204	Bez treści	410	Minęło
205	Resetuj treść	411	Wymagana długość
206	Treść częściowa	412	Warunek wstępny nie powiódł się
300	Wiele opcji	413	Jednostka żądania zbyt duża
301	Przeniesiony na stałe	414	URI żądania zbyt duży
302	Znaleziony	415	Nieobsługiwany typ nośnika
303	Zobacz inny	416	Żądany zakres nie do spełnienia
304	Nie zmodyfikowany	417	Oczekiwanie nie powiodło się
305	Użyj proxy	500	Wewnętrzny błąd serwera
307	Readresowanie tymczasowe	501	Nie zaimplementowany
400	Złe żądanie	502	Zła brama
401	Nieautoryzowany	503	Usługa niedostępna
402	Wymagana opłata	504	Przeterminowanie bramy
403	Wzbroniony	505	Nieobsługiwana wersja HTTP

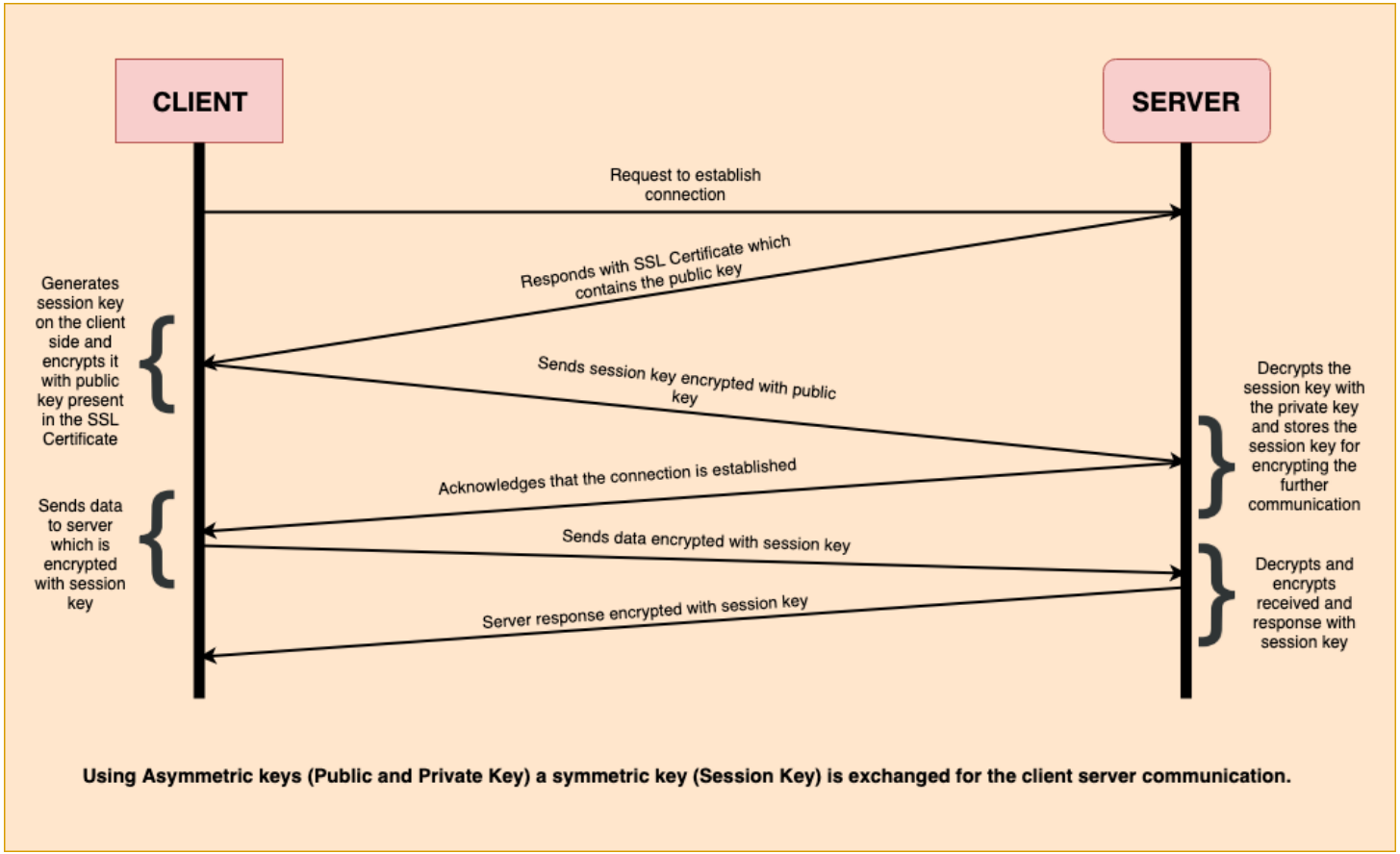
Mechanizm Persistent Connection

- Często pojedyncze żądanie użytkownika końcowego jest realizowane w postaci serii żądań HTTP
- HTTP1.0 - każde żądanie to oddzielne połączenie sieciowe
- HTTP1.1 - podtrzymanie połączenia sieciowego pomiędzy żądaniami HTTP
- Wydajność, ale też zagrożenia

HTTPS

- HTTP jest podatny na podsłuch
- HTTPS to HTTP, w którym zastosowano szyfrowanie SSL
- HTTPS zapewnia:
  - Szyfrowanie żądań i odpowiedzi
  - Kontrolę integralności żądań i odpowiedzi
  - Uwierzytelnianie serwera HTTP
  - Opcjonalne uwierzytelnianie klienta HTTP

Przebieg połączenia HTTPS



Rozwój protokołu HTTP

HTTP 0.9:

- rok 1991
- wykorzystywany protokół: TCP
- praca w trybie: żądanie-odpowiedź
- obsługiwana tylko metoda GET
- typ odpowiedzi: Hipertekst
- bez nagłówków HTTP
- bez kodów odpowiedzi

HTTP 1.0

- rok 1996
- wykorzystywany protokół: TCP
- praca w trybie: żądanie-odpowiedź
- **obsługiwana metody: GET, HEAD, POST**
- typ odpowiedzi: Hipertekst, Skrypty
- **możliwość dodawania nagłówków**

HTTP 1.1

- **rok 1997**
- wykorzystywany protokół: TCP
- praca w trybie: żądanie-odpowiedź
- **obsługiwana metody:** GET, HEAD, POST**+OPTIONS, PUT, DELETE, TRACE, CONNECT**
- typ odpowiedzi: Hipertekst, Skrypty
- możliwość dodawania nagłówków
- **Mechanizm Persistent Connection**

Rozwój protokołu HTTP

HTTP 2.0:

- **rok 2015**
- **wykorzystywany protokół: SPDY** – protokół, opracowany przez Google, który bazuje na TCP i jest ukierunkowany na prędkość działania komunizacji przeglądarki z serwerem wykorzystując kompresje i manipulacje pakietami.
- **praca w trybie: trwałego połączenia, które jest aktywne do zamknięcia strony**
- **umożliwia na wykonywanie wielu zapytań do serwera na raz**

HTTP 3.0 (które przeglądarki)

- **rok 2018**
- **wykorzystywany protokół: QUIC** – nowy protokół, również opracowany przez Google, który bazuje na protokole UDP. Motywacja do zmiany protokołu był problem związany z blokadą żądania HTTP, aż do czasu uzyskania odpowiedzi. UDP jest szybszy i ma mniejsze opóźnienie od TCP.

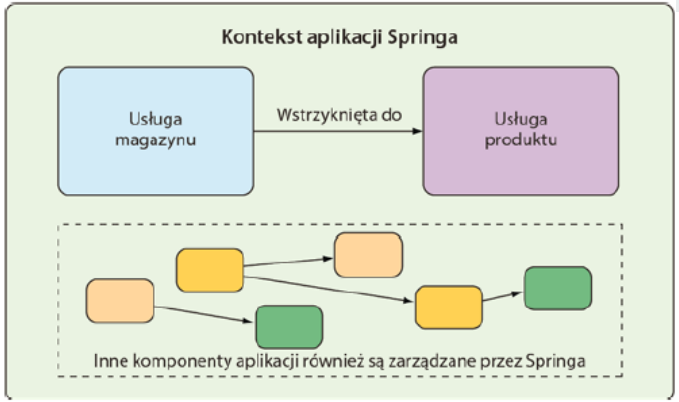
Technologia SpringMVC i Spring Boot

Programowanie aplikacji WWW w języku Java

Spring

Spring jest frameworkiem open source, zaproponowanym przez Roda Johnsona i opisanym w jego książce Expert One-on-One: J2EE Design and Development (2002)

Spring

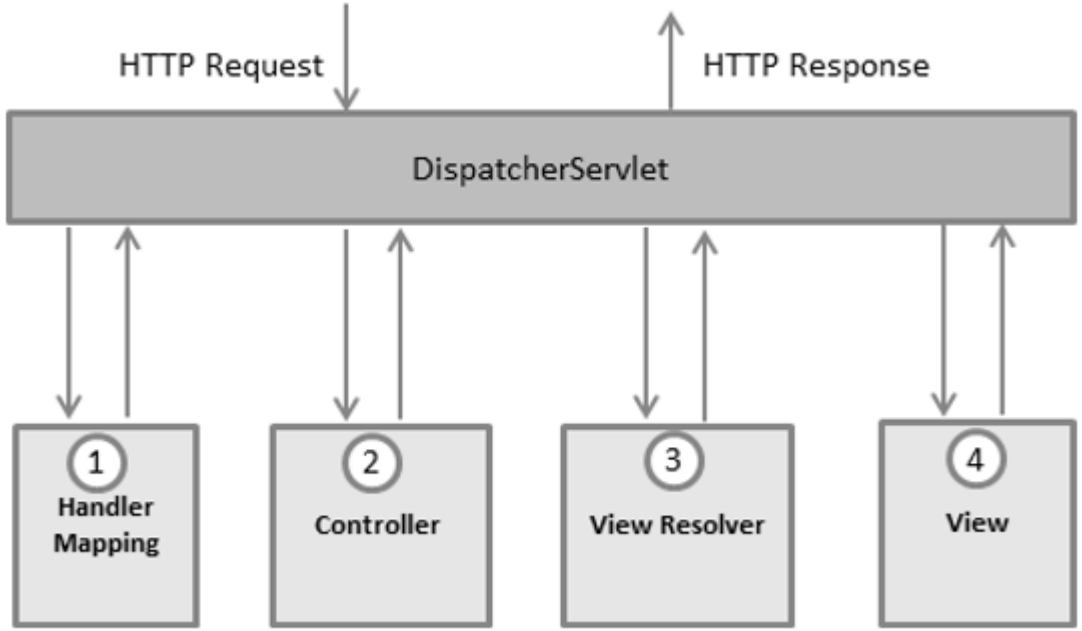


Spring oferuje tzw. kontener, często określany mianem **kontekstu aplikacji Springa**, odpowiedzialny za utworzenie komponentów aplikacji i zarządzanie nimi.

Operacja powiązania ze sobą komponentów jest oparta na wzorcu znanym pod nazwą **wstrzykiwanie zależności** (ang. dependency injection — DI).

Architektura - wzorzec Front-controller

Serwlet = DispatcherServlet + HandlerMapping + Controller + ViewResolver



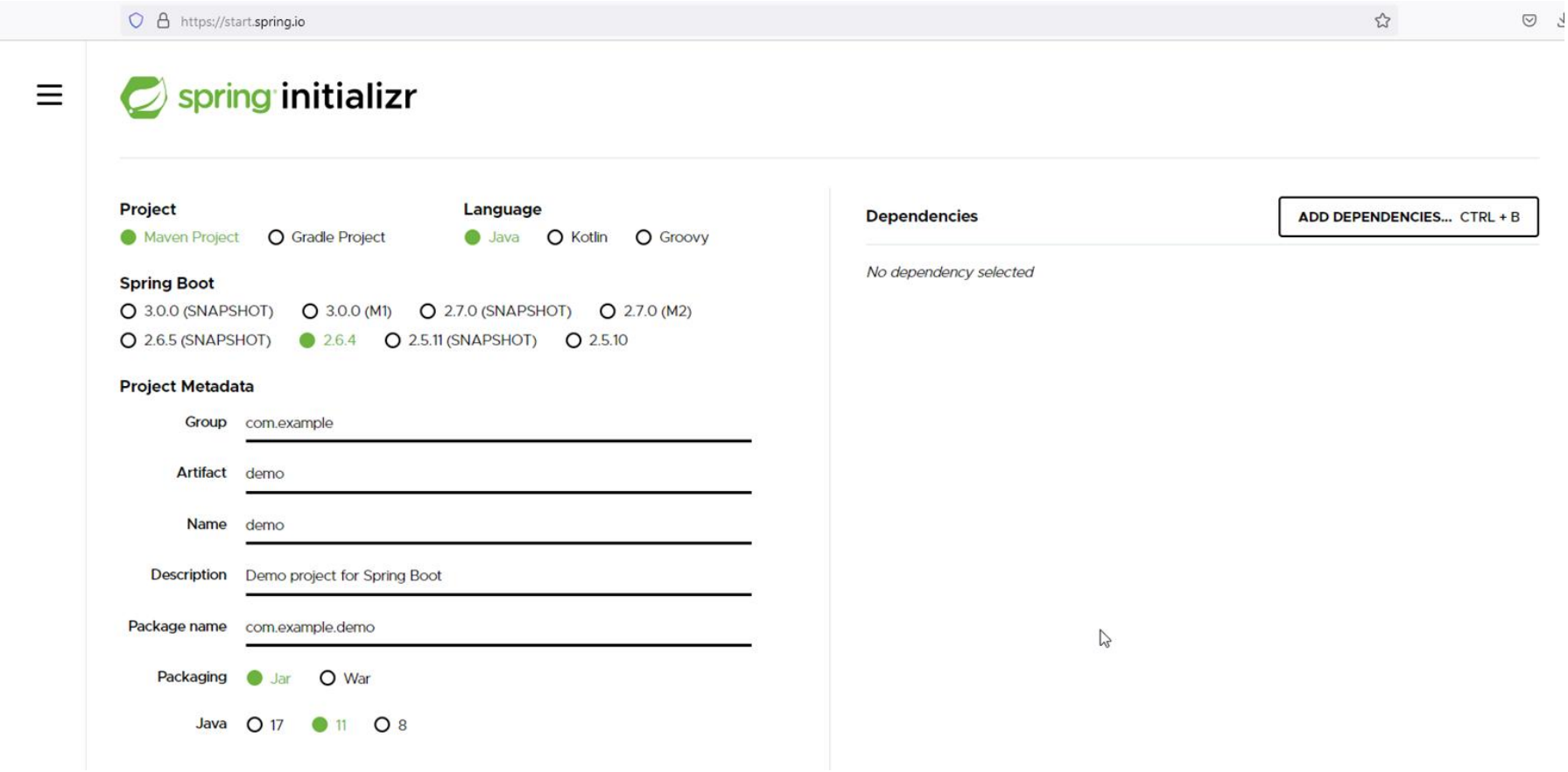
- request→ *DispatcherServlet* → *HandlerMapping* → *Controller*.
- *Controller* → metody GET/POST → service → nazwa widoku
- *DispatcherServlet* → *ViewResolver* → który widok?
- *DispatcherServlet* → dane modelu (kontekstu aplikacji) → view

Kontekst aplikacji Springa a kontekst aplikacji (dwa konteksty :))

Gdy serwlet dystrybutora (DispatcherServlet) rozpoczyna działanie, tworzy **kontekst aplikacji Springa** i **wypełnia go komponentami** (elementami technologii np. kontrolery, producenci widoków) zadeklarowanymi we wskazanych klasach lub plikach konfiguracyjnych.

Drugi **kontekst aplikacji** jest tworzony za pomocą klasy nasłuchującej ContextLoaderListener. Przechowuje **pozostałe komponenty**: komponenty warstwy pośredniej oraz warstwy danych, odpowiadające za funkcjonowanie warstwy logicznej aplikacji.

SpringBoot


https://start.spring.io
spring initializr
Project
Maven Project
Gradle Project
Language
Java
Kotlin
Groovy
Spring Boot
3.0.0 (SNAPSHOT)
3.0.0 (M1)
2.7.0 (SNAPSHOT)
2.7.0 (M2)
2.6.5 (SNAPSHOT)
2.6.4
2.5.11 (SNAPSHOT)
2.5.10
Project Metadata
Group
com.example
Artifact
demo
Name
demo
Description
Demo project for Spring Boot
Package name
com.example.demo
Packaging
Jar
War
Java
17
11
8
Dependencies
ADD DEPENDENCIES... CTRL + B
No dependency selected

Zalety Spring Boot

- Zależności pomiędzy komponentami (klasami) są realizowane automatycznie za pomocą mechanizmu wstrzykiwania zależności (z ang. dependency injection)
- Tworzony kod jest bardzo zwięzły i czytelny. Aplikacje tworzenie z wykorzystaniem Spring są zatem łatwe do zrozumienia nawet kiedy ich obszerność jest duża
- Spring dostarcza możliwość łatwego wykonywania testów jednostkowych oraz integracyjnych.
- Startery Spring Boot — łączą często stosowane grupy zależności w pojedyncze zależności, które można dodawać do procesu budowania aplikacji przy użyciu takich narzędzi jak Maven lub Gradle

Zalety Spring Boot

- Automatyczna konfiguracja — mechanizm automatycznej konfiguracji Spring Boot rozszerza dostępne w Springu 4 wsparcie dla konfiguracji warunkowej
- Plik wykonywalny aplikacji jest pakowany do jar lub war. Wdrożenie aplikacji nie wymaga obszernego serwera aplikacyjnego, lecz lekkiego kontenera aplikacji. Programista nie musi męczyć się z uruchomieniem serwera i jego żmudna konfiguracją.

Spring Boot

List of dependencies for **Spring Boot 2.1.5.RELEASE**

Web

- [ ] **Web**: Servlet web application with Spring MVC and Tomcat
- [ ] **Reactive Web**: Reactive web applications with Spring WebFlux and Netty
- [ ] **Rest Repositories**: Exposing Spring Data repositories over REST via Spring Data REST
- [ ] **Rest Repositories HAL Browser**: Browsing Spring Data REST repositories in your browser
- [ ] **HATEOAS**: HATEOAS-based RESTful services
- [ ] **Web Services**: Contract-first SOAP services with Spring Web Services
- [ ] **Jersey**: RESTful Web Services with JAX-RS
- [ ] **REST Docs**: Document RESTful services by combining hand-written and auto-generated documentation
- [ ] **Vaadin**: Vaadin java web application framework
- [ ] **Apache CXF**: RESTful Web Services with JAX-RS
  Requires Spring Boot >=1.5.0.RELEASE and <2.1.0.M1.
- [ ] **Ratpack**: Spring Boot integration for the Ratpack framework
  Requires Spring Boot >=1.5.0.RELEASE and <2.0.0.M1.
- [ ] **Mobile**: Simplify the development of mobile web applications with Spring Mobile
  Requires Spring Boot >=1.5.0.RELEASE and <2.0.0.M1.

Core

- [ ] **DevTools**: Spring Boot Development Tools
- [ ] **Lombok**: Java annotation library which helps to reduce boilerplate code and code faster
- [ ] **Configuration Processor**: Generate metadata for your custom configuration keys
- [ ] **Session**: API and implementations for managing a user's session information
- [ ] **Cache**: Spring's Cache abstraction
- [ ] **Validation**: JSR-303 validation infrastructure (already included with web)
- [ ] **Retry**: Provide declarative retry support via spring-retry
- [ ] **Aspects**: Create your own Aspects using Spring AOP and AspectJ

Template Engines

- [ ] **Thymeleaf**: Thymeleaf templating engine
- [ ] **Freemarker**: FreeMarker templating engine

Update dependencies | Cancel

Spring Boot

List of dependencies for **Spring Boot 2.1.5.RELEASE**

Web

- [ ] **Web**: Servlet web application with Spring MVC and Tomcat
- [ ] **Reactive Web**: Reactive web applications with Spring WebFlux and Netty
- [ ] **Rest Repositories**: Exposing Spring Data repositories over REST via Spring Data REST
- [ ] **Rest Repositories**
- [ ] **HATEOAS**: HATEOAS
- [ ] **Web Services**: Con
- [ ] **Jersey**: RESTful Web Services with JAX-RS
- [ ] **REST Docs**: Document RESTful services by combining hand-written and auto-generated documentation
- [ ] **Vaadin**: Vaadin java web application framework
- [ ] **Apache CXF**: RESTful Web Services with JAX-RS
  Requires Spring Boot >=1.5.0.RELEASE and <2.1.0.M1.
- [ ] **Ratpack**: Spring Boot integration for the Ratpack framework
  Requires Spring Boot >=1.5.0.RELEASE and <2.0.0.M1.
- [ ] **Mobile**: Simplify the development of mobile web applications with Spring Mobile
  Requires Spring Boot >=1.5.0.RELEASE and <2.0.0.M1.

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

Core

- [ ] **DevTools**: Spring Boot Development Tools
- [ ] **Lombok**: Java annotation library which helps to reduce boilerplate code and code faster
- [ ]
- [ ]
- [ ]
- [ ]
- [ ] **Retry**: Provide declarative retry support via spring-retry
- [ ] **Aspects**: Create your own Aspects using Spring AOP and AspectJ

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

Template Engines

- [ ] **Thymeleaf**: Thymeleaf templating engine
- [ ] **Freemarker**: FreeMarker templating engine

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

Update dependencies | Cancel

Lista starterów link

Table 13.1. Spring Boot application starters

Name	Description	Pom
`spring-boot-starter`	Core starter, including auto-configuration support, logging and YAML	Pom
`spring-boot-starter-activemq`	Starter for JMS messaging using Apache ActiveMQ	Pom
`spring-boot-starter-amqp`	Starter for using Spring AMQP and Rabbit MQ	Pom
`spring-boot-starter-aop`	Starter for aspect-oriented programming with Spring AOP and AspectJ	Pom
`spring-boot-starter-artemis`	Starter for JMS messaging using Apache Artemis	Pom
`spring-boot-starter-batch`	Starter for using Spring Batch	Pom
`spring-boot-starter-cache`	Starter for using Spring Framework's caching support	Pom
`spring-boot-starter-cloud-connectors`	Starter for using Spring Cloud Connectors which simplifies connecting to services in cloud platforms like Cloud Foundry and Heroku	Pom
`spring-boot-starter-data-cassandra`	Starter for using Cassandra distributed database and Spring Data Cassandra	Pom
`spring-boot-starter-data-cassandra-reactive`	Starter for using Cassandra distributed database and Spring Data Cassandra Reactive	Pom
`spring-boot-starter-data-couchbase`	Starter for using Couchbase document-oriented database and Spring Data Couchbase	Pom
`spring-boot-starter-data-couchbase-reactive`	Starter for using Couchbase document-oriented database and Spring Data Couchbase Reactive	Pom
`spring-boot-starter-data-elasticsearch`	Starter for using Elasticsearch search and analytics engine and Spring Data Elasticsearch	Pom
`spring-boot-starter-data-jdbc`	Starter for using Spring Data JDBC	Pom
`spring-boot-starter-data-jpa`	Starter for using Spring Data JPA with Hibernate	Pom

Autokonfiguracja

@SpringBootApplication lub @EnableAutoConfiguration

Elementy konfiguracji są uruchamiane, jeśli odpowiednia biblioteka znajduje się w ścieżce classpath

@SpringBootApplication = @SpringBootConfiguration + @EnableAutoConfiguration + @ComponentScan

Elementy Thymeleaf link


```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<link rel="stylesheet" th:href="@{/styl.css}" />
```

```
<p th:text="#{welcome}">Welcome!</p>
```

```
<p th:text="${welcome}">Welcome!</p>
```

```
<input type="text" th:field="*{name}"></input>
```

- Odwołanie url tu: do zasobów statycznych na dysku: “static” (inne: /META-INF/resources, /resources, /public)
- Odwołanie do komunikatu w pliku messages.properties w katalogu “resources”
- Zmienne
- Pola zmiennych

Uruchamianie aplikacji poza IDE

```
$ mvn package
```

- Wersja WAR - wdrożenie w kat. webapps Tomcata
- Wersja Jar:

```
$ java -jar target/myproject-0.0.1-SNAPSHOT.jar
```

- Wdrażanie w chmurze (narzędzie Cloud Foundry)

```
$ cf push myproject -p target/myproject-0.0.1-SNAPSHOT.jar
```

Stereotypy

@Component — bazowy stereotyp, oznacza, że na podstawie tej klasy będzie utworzony bean Springa. Tego stereotypu używamy najczęściej do klas, które są pomocnicze i nie oferują elementów logiki biznesowej, a jedynie pomocnicze funkcje (np. konwersja między typami, jakieś wspólne elementy)

@Service — stereotyp, który wskazuje, że ta klasa jest serwisem, tzn. oferuje pewną logikę biznesową, którą będziemy wykorzystywać w innych miejscach

@Repository — wskazuje, że klasa pozwala na dostęp do danych, np. wspiera obsługę bazy danych.

Application.properties link

```
server.servlet.context-path=/myapplication
server.port=9090
spring.thymeleaf.cache=true
spring.thymeleaf.mode=HTML
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.session.timeout=30m
spring.messages.basename=messages
spring.messages.encoding=UTF-8
```

Komponenty @Bean

```
<bean id="inventoryService" class="com.example.InventoryService" />
<bean id="productService" class="com.example.ProductService" />
<constructor-arg ref="inventoryService" />
</bean>
```

```
@Configuration
public class ServiceConfiguration {
@Bean
public InventoryService inventoryService() {
    return new InventoryService();
}
@Bean
public ProductService productService() {
    return new ProductService(inventoryService());
}
}
```

skanowanie komponentów+@ComponentScan

Podstawowe elementy technologii SpringMVC

Programowanie aplikacji WWW w języku Java

Przykład

Sklep internetowy, funkcjonalności:

a) przeglądanie produktów (nazwa, cena, kategoria)
b) wkładanie, wyjmowanie produktów do koszyka (sesja)
c) kupowanie
d) przeglądanie historii zamówień
e) wyszukiwanie
f) filtrowanie
g) admin - zarządzanie produktami

Przykład

Sklep internetowy:

a) Klasy dziedziny (domain-driven design)
b) Komponenty, Repozytorium
c) Kontroler
d) Widok
e) Zakres widoczności obiektów dziedziny

Przeglądarka → żadanie → kontroler → dziedzina → kontroler → widok → przeglądarka

Klasy dziedziny

```
public class SomeData {
  private String id;
  private String name;
  private int age;

 public SomeData(String id, String name, int age){
        this.id=id;this.name=name;this.age=age;
 }
 public String getName(){...}
 public void setName(String name){...}
...
}
```

Klasy dziedziny - Lombok

```
@Data
@RequiredArgsConstructor
public class SomeData {
  private String id;
  private String name;
  private int age;

...
}
```

```
<dependency>
     <groupId>org.projectlombok</groupId>
     <artifactId>lombok</artifactId>
 </dependency>
```

Kontroler - @Controller

@RequestMapping ogólnego przeznaczenia obsługa żądań

@GetMapping Obsługa żądań HTTP GET

@PostMapping Obsługa żądań HTTP POST

@PutMapping Obsługa żądań HTTP PUT

@DeleteMapping Obsługa żądań HTTP DELETE

@PatchMapping Obsługa żądań HTTP PATCH

Klasa kontrolera

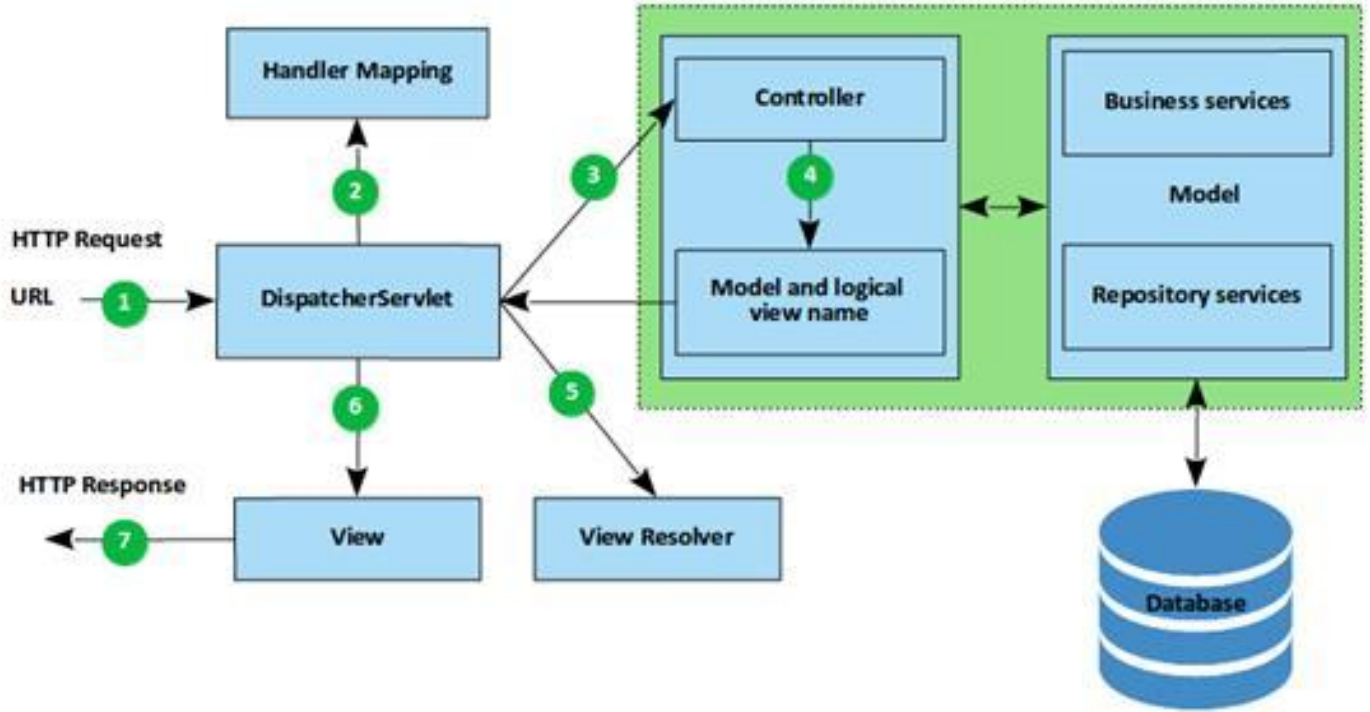
```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseStatus;

@Controller
public class ExampleController {

          @GetMapping("/home")
          public String home() {
          System.out.println("HOME");
                    return "home";
          }

     @RequestMapping("/login", method=GET)
          public String login() {
                    System.out.println("LOGIN");
          return "login";
          }
```

Widok

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org"> ← przestrzeń nazw Thymeleaf
<head>
<title>Tytuł aplikacji</title>
<link rel="stylesheet" type="text/css"
th:href="@{/resources/style.css}"></link> ← Łącze th:href do arkusza stylów
</head>
<body>
<h1>Witamy w aplikacji </h1>
<h1>Wybierz menu: </h1>
<a th:href="@{/abc}">Pierwsza strona</a>
<a th:href="@{/dfgh}">Druga strona</a>
</body>
</html>
```

Model - dane kontekstu

Interfejs org.springframework.ui.Model



Kontroler+Model

Model to obiekt przenoszący dane między kontrolerem i widokiem odpowiedzialnym za ich wyświetlenie.

```
@GetMapping("/printData")
      public String getAllData(Model model) {
      model.addAttribute("name", “Ala Makota”);
      model.addAttribute("age", 23);
      model.addAttribute("login", “abc12we”);
      return "printData";
      }
```

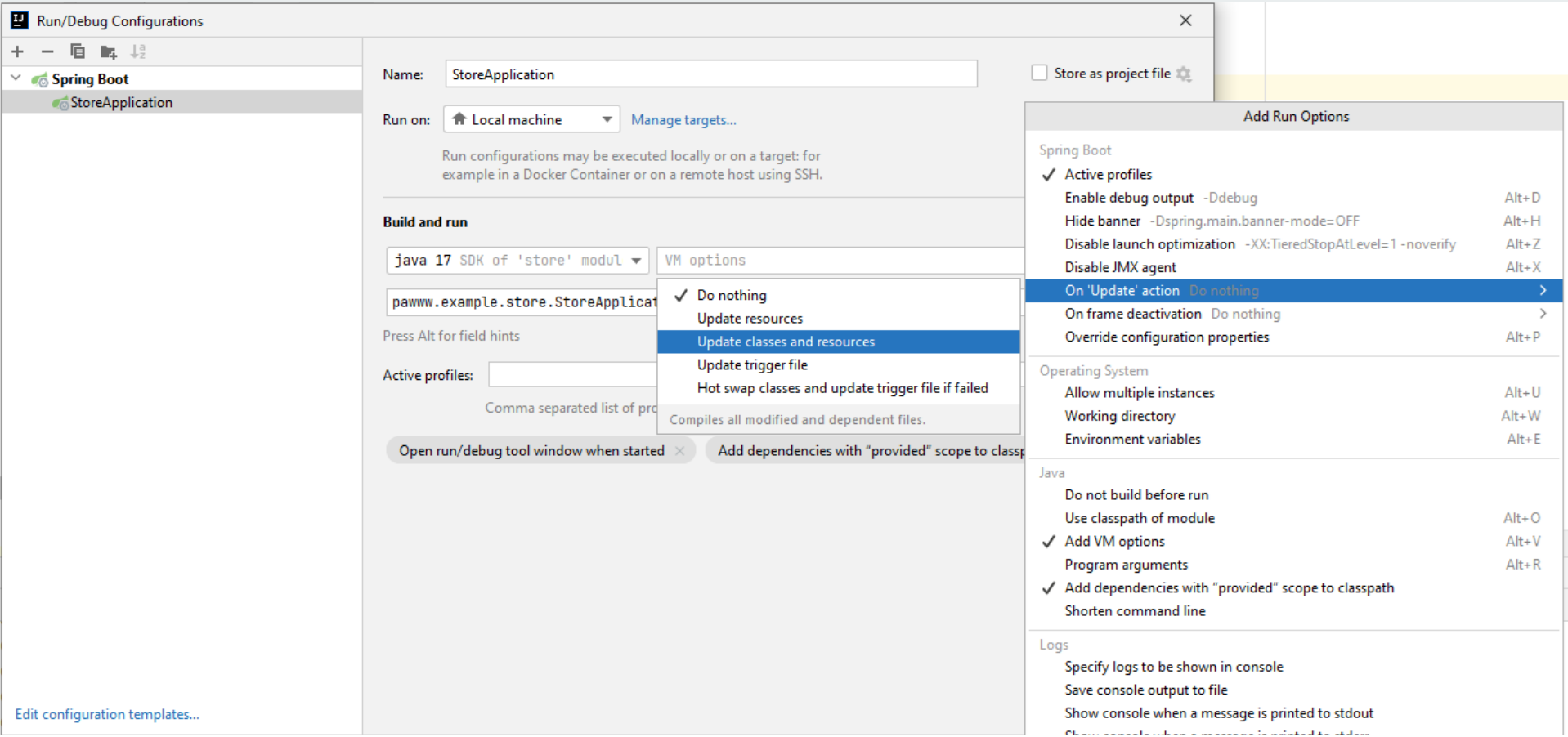
Kontroler+Model

Automatyczne wyznaczanie nazwy atrybutu i widoku

```
@GetMapping("/printData")
	public SomeData getAllData() {
		return new SomeData(1,“Ala
Makota”,23);
	}
```

Widok

```
<h3>Wartości atrybutów:</h3>
<span th:text="${name}">Imię</span><br/>
<span th:text="${age}">Wiek</span><br/>
<span
th:text="${login}">Login</span><br/>
```


Run/Debug Configurations
Spring Boot
StoreApplication
Edit configuration templates...
Name:
StoreApplication
Store as project file
Run on:
Local machine
Manage targets...
Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.
Build and run
java 17 SDK of 'store' modul
VM options
pawww.example.store.StoreApplicat
Press Alt for field hints
Active profiles:
Comma separated list of pro
Open run/debug tool window when started
Add dependencies with "provided" scope to classp
Do nothing
Update resources
Update classes and resources
Update trigger file
Hot swap classes and update trigger file if failed
Compiles all modified and dependent files.
Add Run Options
Spring Boot
Active profiles
Enable debug output -Ddebug Alt+D
Hide banner -Dspring.main.banner-mode=OFF Alt+H
Disable launch optimization -XX:TieredStopAtLevel=1 -noverify Alt+Z
Disable JMX agent Alt+X
On 'Update' action Do nothing
On frame deactivation Do nothing
Override configuration properties Alt+P
Operating System
Allow multiple instances Alt+U
Working directory Alt+W
Environment variables Alt+E
Java
Do not build before run
Use classpath of module Alt+O
Add VM options Alt+V
Program arguments Alt+R
Add dependencies with "provided" scope to classpath
Shorten command line
Logs
Specify logs to be shown in console
Save console output to file
Show console when a message is printed to stdout

Repozytorium danych - @Repository

```
import org.springframework.stereotype.Repository;

@Repository
public class DataRepository {
	List<SomeData> users;

	public DataRepository() {
		this.users=new ArrayList<>();
		this.users.add(new SomeData(1,”Ala
Makota”,23));
	…
	}
```

Repozytorium danych - @Repository Field injection

```
@Controller
public class ExampleController {
     @Autowired
          DataRepository repository;

          @GetMapping("/getUsers")
          public String getAll(Model model) {
          System.out.println("GET ALL");
          model.addAttribute("users", repository.getData());
                    return "printData";
          }
```

Widok

```
<ol th:each="user : ${users}">
<li th:text="${user.id+” “+user.name+”
“+user.age}" />
</ol>
```

Parametry zapytania i zmienne ścieżki

```
@Controller
public class ExampleController {
     @Autowired
          DataRepository repository;

          @GetMapping("/getUsers")
          public String getAll(Model model) {
          System.out.println("GET ALL");
          model.addAttribute("users", repository.getData());
                    return "printData";
          }
//@RequestMapping(value="/getUsersWithRange", params={"min"})
@GetMapping("/getUsersWithRange",params={"min",”max”})
          public String getRange(Model model, final HttpServletRequest req) {
          System.out.println("GET RANGE");
          model.addAttribute("users",
     repository.getRangeData(req.getParameter("min"),req.getParameter("max")));
                    return "printData";
          }
```

Parametry zapytania i zmienne ścieżki

```
@Controller
public class ExampleController {
     @Autowired
          DataRepository repository;

          @GetMapping("/getUsers")
          public String getAll(Model model) {
          System.out.println("GET ALL");
          model.addAttribute("users", repository.getData());
                    return "printData";
          }
          @GetMapping("/getUsersWithRange")
          public String getRange(Model model, @RequestParam("min") String min,
@RequestParam("max") String max)) {
          System.out.println("GET RANGE");
          model.addAttribute("users", repository.getRangeData(min,max));
                    return "printData";
          }
```

Parametry zapytania i zmienne ścieżki

```
@Controller
public class ExampleController {
     @Autowired
          DataRepository repository;

          @GetMapping("/getUsers")
          public String getAll(Model model) {
          System.out.println("GET ALL");
          model.addAttribute("users", repository.getData());
                    return "printData";
          }
          @GetMapping("/getUsersWithRange")
          public String getRange(Model model,
@RequestParam(value="min",defaultValue=1) String min,
@RequestParam(value="max",defaultValue=MAX_LONG_AS_STRING) String max)) {
          System.out.println("GET RANGE");
          model.addAttribute("users", repository.getRangeData(min,max));
                    return "printData";
          }
```

Parametry zapytania i zmienne ścieżki

```
@Controller
public class ExampleController {
     @Autowired
          DataRepository repository;

          @GetMapping("/getUsers")
          public String getAll(Model model) {
          System.out.println("GET ALL");
          model.addAttribute("users", repository.getData());
                    return "printData";
          }
          @GetMapping("/getUser/{id}")
          public String getRange(Model model, @PathVariable("id") String id) {
          System.out.println("GET ONE");
          model.addAttribute("user", repository.getOneData(id));
                    return "printOne";
          }
```

Parametry zapytania i zmienne ścieżki

```
@Controller
public class ExampleController {
     @Autowired
          DataRepository repository;

          @GetMapping("/getUsers")
          public String getAll(Model model) {
          System.out.println("GET ALL");
          model.addAttribute("users", repository.getData());
                    return "printData";
          }
          @GetMapping("/getUser/{id}")
          public String getRange(Model model, @PathVariable String id) {
          System.out.println("GET ONE");
          model.addAttribute("user", repository.getOneData(id));
                    return "printOne";
          }
```

Zasięg komponentów

@Session

Prototype, singleton, session, request

Wyjątki

Mapowanie wyjątków na kody odpowiedzi HTTP

Wyjątek Springa	Kod odpowiedzi http
HttpMessageNotReadableException	400 – Nieprawidłowe zapytanie
ConversionNotSupportedException	500 – Wewnętrzny błąd serwera
HttpMediaTypeNotSupportedException	415 – Nieznany sposób żądania
HttpRequestMethodNotSupportedException	405 – Niedozwolona metoda

Klasa kontrolera

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseStatus;

@Controller
public class ExampleController {

          @GetMapping("/home")
          public String home() {
          System.out.println("HOME");
                    return "home";
          }

     @RequestMapping("/login", method=GET)
          public String login() {
                    System.out.println("LOGIN");
          return "login";
          }
```

```
@ExceptionHandler(Exception.class)
          @ResponseStatus(value =
HttpStatus.INTERNAL_SERVER_ERROR, reason = "Error
message")
          public void handleError() {
                    System.out.println("ERROR");
          }
```

Klasa kontrolera

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseStatus;

@Controller
public class ExampleController {

        @GetMapping("/home")
        public String home() {
        System.out.println("HOME");
                return "home";
        }

    @RequestMapping("/login", method=GET)
        public String login() {
                System.out.println("LOGIN");
        return "login";
        }
```

```
@ExceptionHandler(Exception.class)
        @ResponseStatus(value =
HttpStatus.NOT_FOUND, reason = "No requested
data")
        public void handleDataError() {
                System.out.println("ERROR");
        }
```

Wyjątki w klasie wyjątku

```
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value=HttpStatus.NOT_FOUND, reason="No data
found")
public class MyDataNotFoundException extends RuntimeException
{
            public MyDataNotFoundException(String msg){
            super(msg);
      }
 }
```

Wyjątki w klasie wyjątku - kontroler

```
@GetMapping("/add/{id}")
          public String addItemToCart(@PathVariable int id, Model model) {
        try{
           Item item2add=service.getItem(id);
           if(item2add==null) throw new DataNotFoundException(String.valueOf(id));
           this.service.addToCart(item2add);
           model.addAttribute("cart", this.service.getCart().getItems());
           return "redirect:/cart/";
     }catch (Exception e){
        System.out.println("error "+e.getMessage());
        model.addAttribute("errorMsg",e.getMessage());
        return "error";
     }
}
```

LUB Wyjątki w klasie wyjątku - kontroler

```
@GetMapping("/add/{id}")
          public String addItemToCart(@PathVariable int id, Model model) {
          Item item2add=service.getItem(id);
          if(item2add==null) throw new DataNotFoundException(String.valueOf(id));
          this.service.addToCart(item2add);
          model.addAttribute("cart", this.service.getCart().getItems());
          return "redirect:/cart/";
}

@ExceptionHandler(DataNotFoundException.class)
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "No requested data")
public String handleDataError(DataNotFoundException e) {
   System.out.println("ERROR "+e.getMessage() );
   return "error";
}
```

Generowanie widoków - Thymeleaf

Programowanie aplikacji WWW w języku Java

Przykład

Sklep internetowy, funkcjonalności:

a) **przeglądanie produktów (nazwa, cena, kategoria)**
b) wkładanie, wyjmowanie produktów do koszyka (sesja)
c) **kupowanie**
d) **przeglądanie historii zamówień**
e) wyszukiwanie
f) filtrowanie
g) admin - zarządzanie produktami

Notacja wyrażeń

- `${...}` : Variable expressions.
- `*{...}` : Selection expressions.
- `#{...}` : Message (i18n) expressions.
- `@{...}` : Link (URL) expressions.
- `~{...}` : Fragment expressions.

```
${session.user.name}
<span th:text="${book.author.name}">

<div th:object="${book}">
  <span th:text="*{title}">...</span>
</div>
```

```
{
  final Book selection = (Book)

context.getVariable("book");
  output(selection.getTitle());
}
```

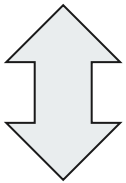
?

?

```
#{main.title}
#{message.entrycreated(${entryId})}
#{${config.adminWelcomeKey}(${session.user.name})}
```

Znaczniki Thymeleaf

```
<span th:text="|Welcome to our application,
${user.name}!|">
```



```
<span th:text="'Welcome to our application, ' +
${user.name} + '!'">
```

```
<div th:with="isEven=(${prodStat.count} % 2 == 0)">
```

Zmienne lokalne

```
<div th:with="firstPer=${persons[0]}">
  <p>
    The name of the first person is <span
th:text="${firstPer.name}">Julius Caesar</span>.
  </p>
</div>
```

Odniesienia

- `${...}` : Variable expressions.
- `*{...}` : Selection expressions.
- `#{...}` : Message (i18n) expressions.
- `@{...}` : Link (URL) expressions.
- `~{...}` : Fragment expressions.

```
<a th:href="@{/order/list}">...</a>
<a
th:href="@{/order/details(id=${orderId},type=${orderTy
pe})}">...</a>
<a th:href="@{../documents/report}">...</a>
<a th:href="@{~/contents/main}">...</a>
<a th:href="@{http://www.mycompany.com/main}">...</a>
```

Widok błędu - plik error.html

- Automatyczne przekierowanie do widoku ‘error’ w momencie wystąpienia wyjątku
- Zmienna ‘message’ z komunikatem błędu - komunikat z ‘reason’ lub zwracany przez metodę getMessage()

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Error</h1>
<p th:text="${message}">Not found</p>
</body>
</html>
```

```
@ResponseStatus(value= HttpStatus.NOT_FOUND, reason="No data with this id ")
public class DataNotFoundException extends RuntimeException {
    String message;
    public DataNotFoundException(String msg){
        System.out.println("err");
        message="ERROR with data id "+msg;
    }

    @Override
    public String getMessage() {
        return message;
    }
}
```

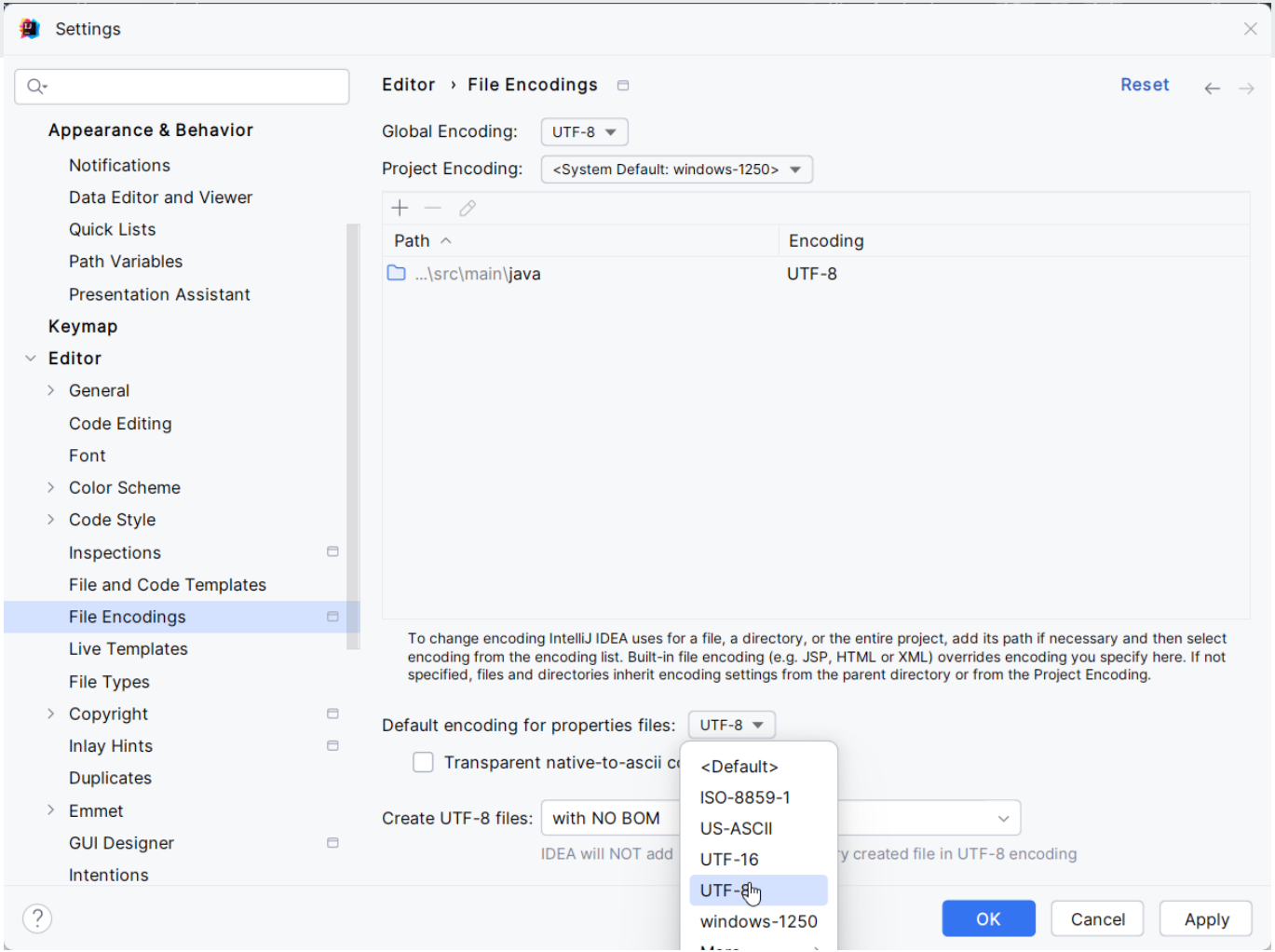
Html in messages

home.welcome=Welcome to our fantastic grocery store!

```
<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>
```

This will output our message just like we wanted it:

```
<p>Welcome to our <b>fantastic</b> grocery store!</p>
```


Settings
Editor › File Encodings
Reset
Appearance & Behavior
Notifications
Data Editor and Viewer
Quick Lists
Path Variables
Presentation Assistant
Keymap
Editor
General
Code Editing
Font
Color Scheme
Code Style
Inspections
File and Code Templates
File Encodings
Live Templates
File Types
Copyright
Inlay Hints
Duplicates
Emmet
GUI Designer
Intentions
Global Encoding: UTF-8
Project Encoding: <System Default: windows-1250>
Path
Encoding
...\src\main\java
UTF-8
To change encoding IntelliJ IDEA uses for a file, a directory, or the entire project, add its path if necessary and then select encoding from the encoding list. Built-in file encoding (e.g. JSP, HTML or XML) overrides encoding you specify here. If not specified, files and directories inherit encoding settings from the parent directory or from the Project Encoding.
Default encoding for properties files: UTF-8
Transparent native-to-ascii c
Create UTF-8 files: with NO BOM
IDEA will NOT add
y created file in UTF-8 encoding
<Default>
ISO-8859-1
US-ASCII
UTF-16
UTF-8
windows-1250
OK
Cancel
Apply

Znaczniki Thymeleaf

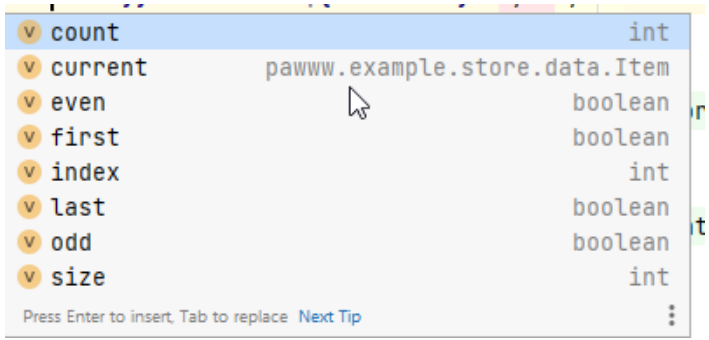
```
 <div th:object="${session.user}">

    <p>Name: <span
th:text="*{firstName}">Sebastian</span>.</p>

    <p>Surname: <span
th:text="*{lastName}">Pepper</span>.</p>

    <p>Nationality: <span
th:text="*{nationality}">Saturn</span>.</p>

  </div>
```

Warunki logiczne

<h1 th:if=”${items.size()>0} ” th:text="’Items in store'"></h1>

<h1 th:text="${items.size()>0} ? 'Items in store':'No items in store'"></h1>

<tr th:each="o ,oStat: ${orders}" th:class="${oStat.odd}? 'odd'">



Warunki logiczne

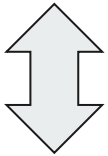
```
<div th:switch="${user.role}">
  <p th:case="'admin'">User is an administrator</p>
  <p th:case="#{roles.manager}">User is a manager</p>
  <p th:case="*">User is some other thing</p>
</div>
```

Wartości logiczne

- f value is not null:
  - If value is a boolean and is true.
  - If value is a number and is non-zero
  - If value is a character and is non-zero
  - If value is a String and is not “false”, “off” or “no”
  - If value is not a boolean, a number, a character or a String.
- (If value is null, th:if will evaluate to false).

Wyrażenia poza atrybutami

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



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

<p>The message is "[(${msg})]"</p> → utext
```

Kolejność wykonywania atrybutów

Order	Feature	Attributes
1	Fragment inclusion	`th:insert` `th:replace`
2	Fragment iteration	`th:each`

3	Conditional evaluation	`th:if`
		`th:unless`
		`th:switch`
		`th:case`

4	Local variable definition	`th:object` `th:with`

5	General attribute modification	`th:attr`
		`th:attrprepend`
		`th:attrappend`

6	Specific attribute modification	`th:value`
		`th:href`
		`th:src`
		`...`

7	Text (tag body modification)	`th:text` `th:utext`
8	Fragment specification	`th:fragment`
9	Fragment removal	`th:remove`

Fragmenty

- ${...} : Variable expressions.
- *{...} : Selection expressions.
- #{...} : Message (i18n) expressions.
- @{...} : Link (URL) expressions.
- ~{...} : Fragment expressions.

~~ ... ~~

```
<div th:insert="~{commons :: main}">...</div>

<div th:insert="~{commons :: some_markup}">...</div>
<div th:insert="~{commons :: div.some_class}">...</div>
```

commons.html:

```
<div th:fragment="main">
        The main part of all templates
</div>
<some_markup>The footer</some_markup>
<div class="some_class">Additional content</div>
```

Fragmenty

Difference between th:insert and th:replace (and ~~th:include~~)

```
<footer th:fragment="copy">
  &copy; 2011 The Good Thymes Virtual
Grocery
</footer>
```

```
<body>
  ...
  <div th:insert="footer::copy"></div>

  <div th:replace="footer::copy"></div>

  <div th:include="footer::copy"></div>

</body>
```

```
<body>

  ...

  <div>
    <footer>
      &copy; 2011 The Good Thymes
Virtual Grocery
    </footer>
  </div>

  <footer>
    &copy; 2011 The Good Thymes
Virtual Grocery
  </footer>

  <div>
    &copy; 2011 The Good Thymes
Virtual Grocery
  </div>

</body>
```

Fragmenty z parametrami

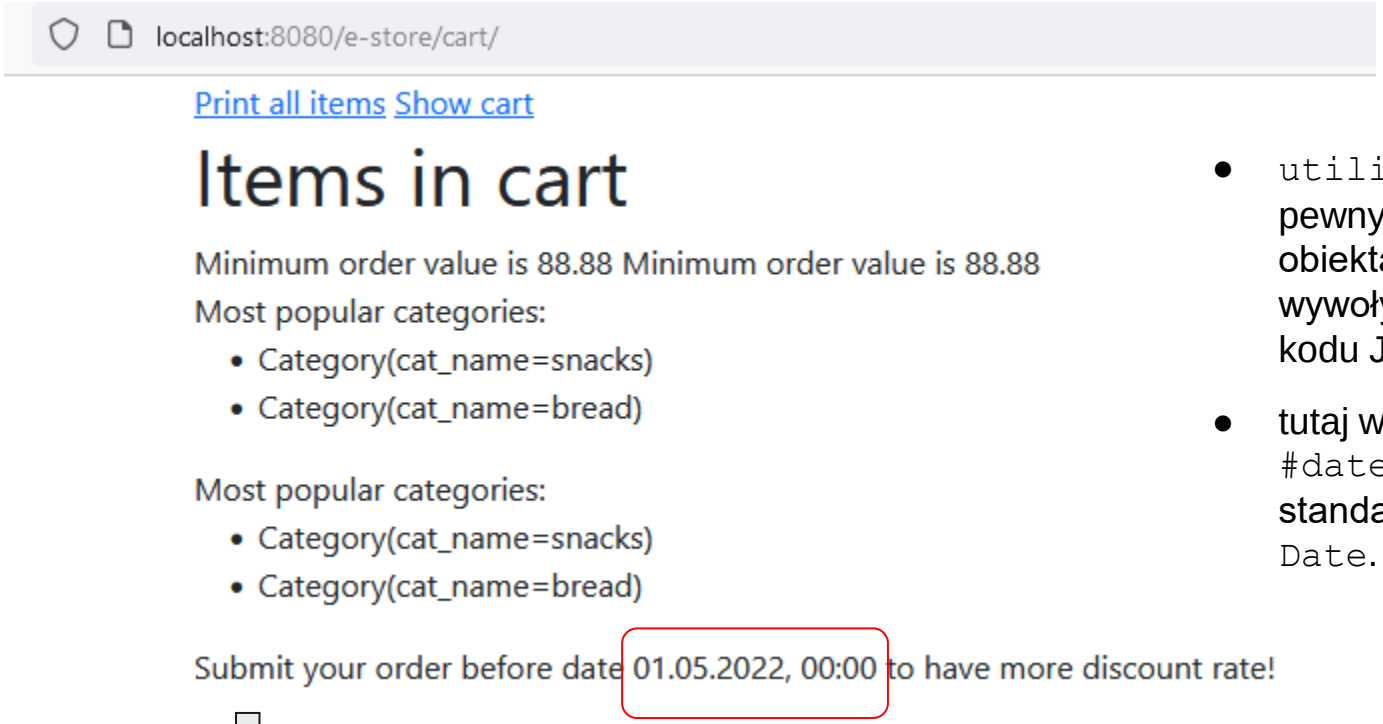
- `${...}` : Variable expressions.
- `*{...}` : Selection expressions.
- `#{...}` : Message (i18n) expressions.
- `@{...}` : Link (URL) expressions.
- `~{...}` : Fragment expressions.

```
<div th:replace="~{commons ::
formField(field=${user.name}, value='John Doe',
size='40')}">
</div>
```

commons.html:

```
<div th:fragment="formField (field, value, size)">
   <span th:text="${field}">Field</span>
   <input type="text" th:name="${field}" th:value="${value}" th:size="${size}">
   <span th:if=”${size>0}”>Text is limited</span>
</div>
```

Utility objects



- `utility objects` są pewnymi predefiniowanymi obiektami, które umożliwiają wywoływanie standardowego kodu Javy w widokach
- tutaj wykorzystano obiekt `#dates`, na którym wywołano standardową metodę z klasy `Date`.

```
<p th:text="${#dates.format(date, 'dd-MM-yyyy HH:mm')}"></p>
```

Utility objects

- ***#dates***: utility methods for *java.util.Date* objects
- ***#calendars***: similar to *#dates*, used for *java.util.Calendar* objects
- ***#numbers***: utility methods for formatting numeric objects
- ***#strings***: utility methods for *String* objects
- ***#objects***: utility methods for Java *Object* class in general
- ***#bools***: utility methods for *boolean* evaluation
- ***#arrays***: utility methods for arrays
- ***#lists***: utility methods for lists
- ***#sets***: utility methods for sets
- ***#maps***: utility methods for maps
- ***#aggregates***: utility methods for creating aggregates on arrays or collections
- ***#messages***: utility methods for obtaining externalized messages inside variables expressions

Utility objects

- ***#dates***: utility methods for *java.util.Date* objects
- ***#calendars***: similar to *#dates*, used for *java.util.Calendar* objects

```
<p th:text="${#dates.format(date, 'dd-MM-yyyy HH:mm')}"></p>
<p th:text="${#dates.dayOfWeekName(date)}"></p>
<p th:text="${#dates.createNow()}"></p>
<p th:text="${#dates.createToday()}"></p>

<p th:text="${#calendars.formatISO(calendar)}"></p>
<p th:text="${#calendars.format(calendar, 'dd-MM-yyyy HH:mm')}"></p>
<p th:text="${#calendars.dayOfWeekName(calendar)}"></p>
```

Utility objects

```
<p th:text="${#numbers.formatDecimal(num,2,3)}"></p>
<p th:text="${#numbers.formatDecimal(num,2,3,'COMMA')}"></p>
```

The options are *POINT*, *COMMA*, *WHITESPACE*, *NONE* or *DEFAULT* (by locale).

```
<p th:each="number: ${#numbers.sequence(0,2)}">
    <span th:text="${number}"></span>
</p>
<p th:each="number: ${#numbers.sequence(0,4,2)}">
    <span th:text="${number}"></span>
</p>
```

Utility objects

```
<p th:text="${#strings.isEmpty(string)}"></p>
<p th:text="${#strings.isEmpty(nullString)}"></p>
<p th:text="${#strings.defaultString(emptyString,'Empty String')}"></p>

<p th:text="${#strings.indexOf(name,frag)}"></p>
<p th:text="${#strings.substring(name,3,5)}"></p>
<p th:text="${#strings.substringAfter(name,prefix)}"></p>
<p th:text="${#strings.substringBefore(name,suffix)}"></p>
<p th:text="${#strings.replace(name,'las','ler')}"></p>

<p th:text="${#strings.equals(first, second)}"></p>
<p th:text="${#strings.equalsIgnoreCase(first, second)}"></p>
<p th:text="${#strings.concat(values...)}"></p>
<p th:text="${#strings.concatReplaceNulls(nullValue, values...)}"></p>

<p th:text="${#strings.abbreviate(string,5)} "></p>
<p th:text="${#strings.capitalizeWords(string)}"></p>
```

Utility objects

```
<p th:text="${#aggregates.sum(array)}"></p>
<p th:text="${#aggregates.avg(array)}"></p>
<p th:text="${#aggregates.sum(set)}"></p>
<p th:text="${#aggregates.avg(set)}"></p>
```

Wprowadzanie i walidacja danych

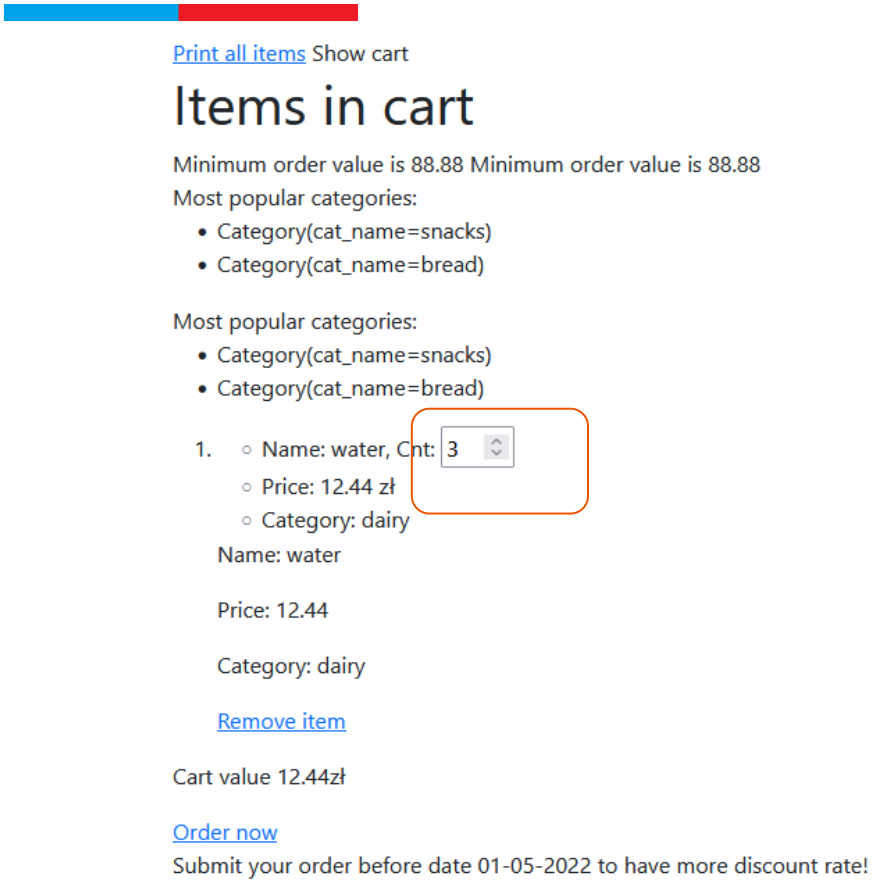
Programowanie aplikacji WWW w języku Java

Przykład

Sklep internetowy, funkcjonalności:

a) przeglądanie produktów (nazwa, cena, kategoria)
b) wkładanie, wyjmowanie produktów do koszyka (sesja)
c) kupowanie
d) przeglądanie historii zamówień
e) wyszukiwanie
f) filtrowanie
g) admin - zarządzanie produktami

Jak to zrealizować?



localhost:8080/e-store/items/

Print all items Show cart

Items in store

Search items:

select category

select category

dairy

snacks

Price: 12.44

Category: dairy

Add to cart

2. ◦ Index:1
   ◦ Name: protein bar
   ◦ Price: 2.58 zł
   ◦ Category: snacks

Name: protein bar

Price: 2.58

Category: snacks

Add to cart

3. ◦ Index:2
   ◦ Name: milk
   ◦ Price: 8.14 zł
   ◦ Category: dairy

Walidacja danych

Praca z formularzem składa się z dwóch etapów:

1) wyświetlania formularza

2) i przetwarzania danych przesłanych przez użytkownika za ich pomocą

Dodawanie produktu

Praca z formularzem składa się z dwóch etapów:

1) wyświetlania formu

2) i przetwarzania dan

```
@GetMapping("/manage/add")
public String manage(Model model){
    model.addAttribute( attributeName: "categories",this.items.getCategories());
    model.addAttribute( attributeName: "newItem", new Item());
    return "add_item";
}
```

```
@PostMapping("/manage/add")
public String managePost(@ModelAttribute("newItem") Item newItem) {
    this.items.addItem(newItem);
    return "redirect:/items/";

}
```

Formularz

```
<body>
<div class="container">
    <span th:insert="~{menu :: menu}"></span>
    <h1>Add new item</h1>
    <form action="#" th:action="@{/items/manage/add}" th:object="${newItem}" method="post">
        <p>Name: <input type="text" th:field="*{name}" /></p>
        <p>Price: <input type="number" th:field="*{price}" /></p>
        <span>
            <select class="form-select form-select-lg mb-md-3" th:field="*{category}">
            <option value="*">select category</option>
            <option th:each="option : ${categories}" th:value="${option.cat_name}" th:text="${option.cat_name}"></option>
           </select>
        </span>
        <p><input type="submit" value="Add" /> </p>
    </form>
</div>
<span th:insert="~{menu :: footer}"></span>
</body>
```

Walidacja

1. Zadeklarować reguły weryfikacji w klasie przeznaczonej do sprawdzenia.
2. Zadeklarować, że weryfikacja powinna się odbywać w metodach kontrolera wymagających sprawdzania danych.
3. Zmodyfikować widoki formularzy w taki sposób, aby wyświetlały informacje o błędach odkrytych podczas weryfikacji danych.

Walidacja pełna lista link

1. Zadeklarować reguły weryfikacji w klasie przeznaczonej do sprawdzenia.
2. …ać w metodach
3. …by wyświetlały …cji danych.

Adnotacja	Opis
@AssertFalse	Oznaczony element musi być typu Boolean i przyjąć wartość false.
@AssertTrue	Oznaczony element musi być typu Boolean i przyjąć wartość true.
@DecimalMax	Oznaczony element musi być liczbą, której wartość jest mniejsza od danej wartości typu BigDecimalString lub jej równa.
@DecimalMin	Oznaczony element musi być liczbą, której wartość jest większa od danej wartości typu BigDecimalString lub jej równa.
@Digits	Oznaczony element musi być liczbą posiadającą określoną liczbę cyfr.
@Future	Wartością oznaczonego element musi być data z przyszłości.
@Max	Oznaczony element musi być liczbą, której wartość jest mniejsza od danej wartości lub jej równa.
@Min	Oznaczony element musi być liczbą, której wartość jest większa od danej wartości lub jej równa.
@NotNull	Wartością oznaczonego elementu nie może być null.
@Null	Wartością oznaczonego elementu musi być null.
@Past	Wartością oznaczonego elementu musi być data z przeszłości.
@Pattern	Wartość oznaczonego elementu musi spełniać warunek określony wyrażeniem regularnym.
@Size	Wartością oznaczonego elementu musi być ciąg znaków typu String, kolekcja lub tablica, której długość mieści się w podanym zakresie.

Walidacja

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

```java
public class Item {
    @NonNull
    @NotNull
    @Size(min=3, max=10)
    private String name;
    @NotNull
    @DecimalMin("0.1")
    private float price;
    @NonNull
    private Category category;
}
```

```java
@PostMapping("/manage/add")
public String managePost(@Valid @ModelAttribute("newItem") Item newItem, /*Errors*/ BindingResult result) {
    System.out.println("resuult "+result.hasErrors());
        if (result.hasErrors()) {
            System.out.println(result.getErrorCount());
            result.getAllErrors().forEach(el-> System.out.println(el));
            return "add_item";
        }

        this.items.addItem(newItem);
    return "redirect:/items/";

}
```

Walidacja

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

```java
public class Item {
    @NonNull
    @NotNull
    @Size(min=3, max=10)
    private String name;
    @NotNull
    @DecimalMin("0.1")
    private float price;
    @NonNull
    private Category category;
}
```

```java
@PostMapping("/manage/add")
public String managePost(@Valid @ModelAttribute("newItem") Item newItem, /*Errors*/ BindingResult result) {
    System.out.println("resuult "+result.hasErrors());
        if (result.hasErrors()) {
            System.out.println(result.getErrorCount());
            result.getAllErrors().forEach(el-> System.out.println(el));
            return "add_item";
        }

        this.items.addItem(newItem);
    return "redirect:/items/";

}
```

???

Walidacja

```java
public class Item {
    @NonNull
    @NotNull
    @Size(min=3, max=10)
    private String name;
    @NotNull
    @DecimalMin("0.1")
    private float price;
    @NonNull
    private Category category;
}
```

???

```java
@Data
@RequiredArgsConstructor
@NoArgsConstructor
public class Category {
    @NonNull
    @NotNull
    @Size(min=3, max=10)
    private String cat_name;
}
```

```java
public class Item {
    @NonNull
    @NotNull
    @Size(min=3, max=10)
    private String name;
    @NotNull
    @DecimalMin("0.1")
    private float price;
    @NonNull
    @Valid
    private Category category;
}
```

Komunikaty błędów

```
<h1>Add new item</h1>
<form action="#" th:action="@{/items/manage/add}" th:object="${newItem}" method="post">
    <p>Name: <input type="text" th:field="*{name}" /></p><span th:if="${#fields.hasErrors('name')}" th:errors="*{name}">Name Error</span>
    <p>Price: <input type="number" th:field="*{price}" step="0.1" /></p><span th:if="${#fields.hasErrors('price')}" th:errors="*{price}">Price Error</span>
    <span>
        <select class="form-select form-select-lg mb-md-3" th:field="*{category}">
        <option value="">select category</option>
        <option th:each="option : ${categories}" th:value="${option.cat_name}" th:text="${option.cat_name}"></option>
       </select>
    </span><span th:if="${#fields.hasErrors('category.cat_name')}" th:errors="*{category.cat_name}">Category Error</span>
    <p><input type="submit" value="Add" /> </p>
</form>
```

Komunikaty błędów

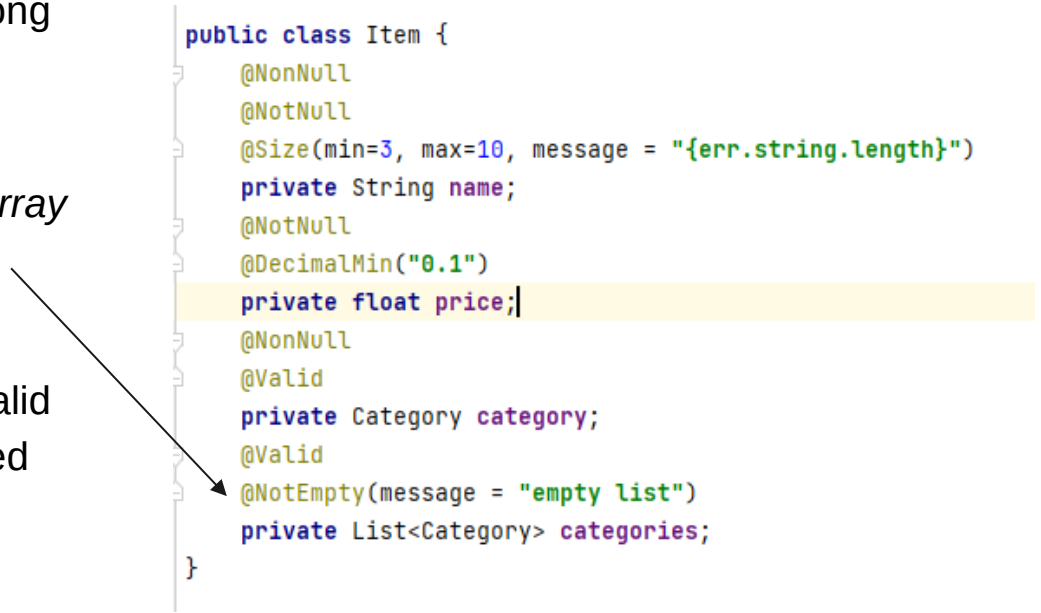
```java
public class Item {
    @NonNull
    @NotNull
    @Size(min=3, max=10, message = "{err.string.length}")
    private String name;
    @NotNull
    @DecimalMin("0.1")
    private float price;
    @NonNull
    @Valid
    private Category category;
}
```

messages.properties | add_item.html | items.html | ItemsController.java | DataNotFoundExceptio

```
1  welcome=Hello everybody!
2  thankyou=Thank you for your order! Pay {0} {1}!
3  home.welcome=Welcome to our <b>fantastic</b> grocery store!
4  menu.print_all.msg=Print all items
5  menu.print_one.msg=Print the first item
6  items.welcome.msg=Items in store
7  menu.show_cart.msg=Show cart
8  menu.search_items.msg=Search and filter items
9  item.welcome.msg=Details of the item "{0}"
10 cart.promo.msg=Submit your order before date {0} to have more discount rate!
11 currency.short=zł
12 err.string.length=Pole {0} musi mieć długość od {2} do {1}
```

Walidacja

- *@NotNull:* a constrained *CharSequence*, *Collection*, *Map,* or *Array* is valid as long as it's not null, but it can be empty.
- *@NotEmpty:* a constrained *CharSequence*, *Collection*, *Map,* or *Array* is valid as long as it's not null, and its size/length is greater than zero.
- *@NotBlank:* a constrained *String* is valid as long as it's not null, and the trimmed length is greater than zero.



```
public class Item {
    @NonNull
    @NotNull
    @Size(min=3, max=10, message = "{err.string.length}")
    private String name;
    @NotNull
    @DecimalMin("0.1")
    private float price;
    @NonNull
    @Valid
    private Category category;
    @Valid
    @NotEmpty(message = "empty list")
    private List<Category> categories;
}
```

Własny walidator

```java
import javax.validation.Constraint;
import javax.validation.Payload;
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.*;

@Target( { FIELD, METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Constraint(validatedBy = CategoryValidator.class)
public @interface CategoryValidation {
    //error message
    public String message() default "{err.category}";
    //represents group of constraints
    public Class<?>[] groups() default {};
    //represents additional information about annotation
    public Class<? extends Payload>[] payload() default {};
}
```

Własny walidator

```java
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;


public class CategoryValidator implements
        ConstraintValidator<CategoryValidation, String> {

    public void initialize(CategoryValidator categoryValidator) {
    }

    @Override
    public boolean isValid(String category,
                           ConstraintValidatorContext cxt) {
        return category != null && !category.contains(" ")
                && (category.length() > 2);
    }

}
```

```java
@Data
@RequiredArgsConstructor
@NoArgsConstructor
public class Category {
    @NonNull
    @NotBlank(message = "{err.required}")
    @CategoryValidation
    @Size(min=2, max=13, message = "{err.string.length}")
    private String cat_name;
}
```

Utrwalanie danych Spring Data JPA

Programowanie aplikacji WWW w języku Java

Przykład

Sklep internetowy, funkcjonalności:

a) przeglądanie produktów (nazwa, cena, kategoria)
b) wkładanie, wyjmowanie produktów do koszyka (sesja)
c) kupowanie
d) przeglądanie historii zamówień
e) wyszukiwanie
f) filtrowanie
g) admin - zarządzanie produktami

Java Persistence API

SQL

- [ ] **JPA**: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate
- [ ] **MySQL**: MySQL JDBC driver
- [ ] **H2**: H2 database (with embedded support)
- [ ] **JDBC**: Persistence support using JDBC
- [ ] **SQL Server**: Microsoft SQL Server JDE
- [ ] **HSQLDB**: HSQLDB database (with em
- [ ] **Apache Derby**: Apache Derby database (with embedded support)

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

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

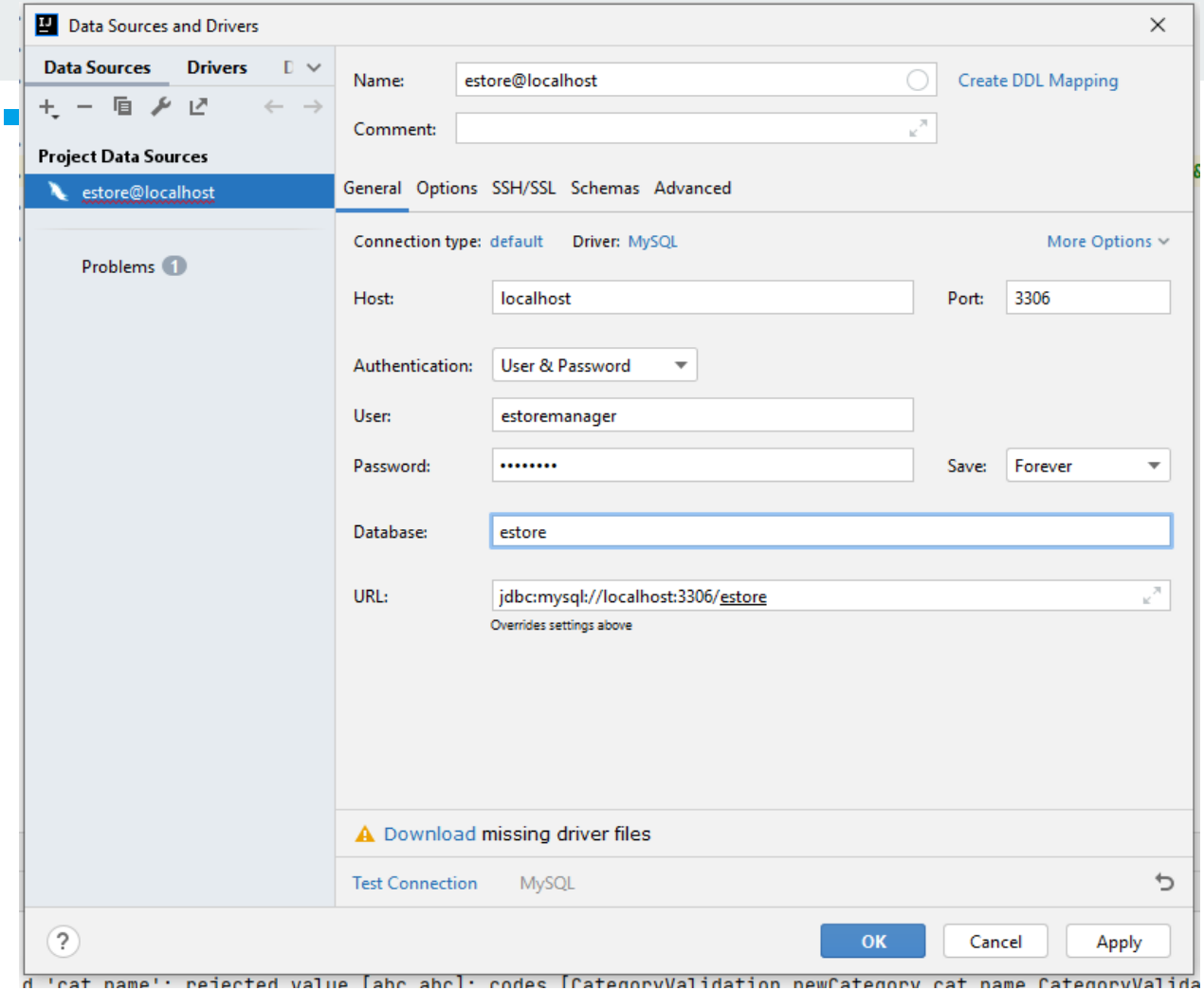
```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
</dependency>
```

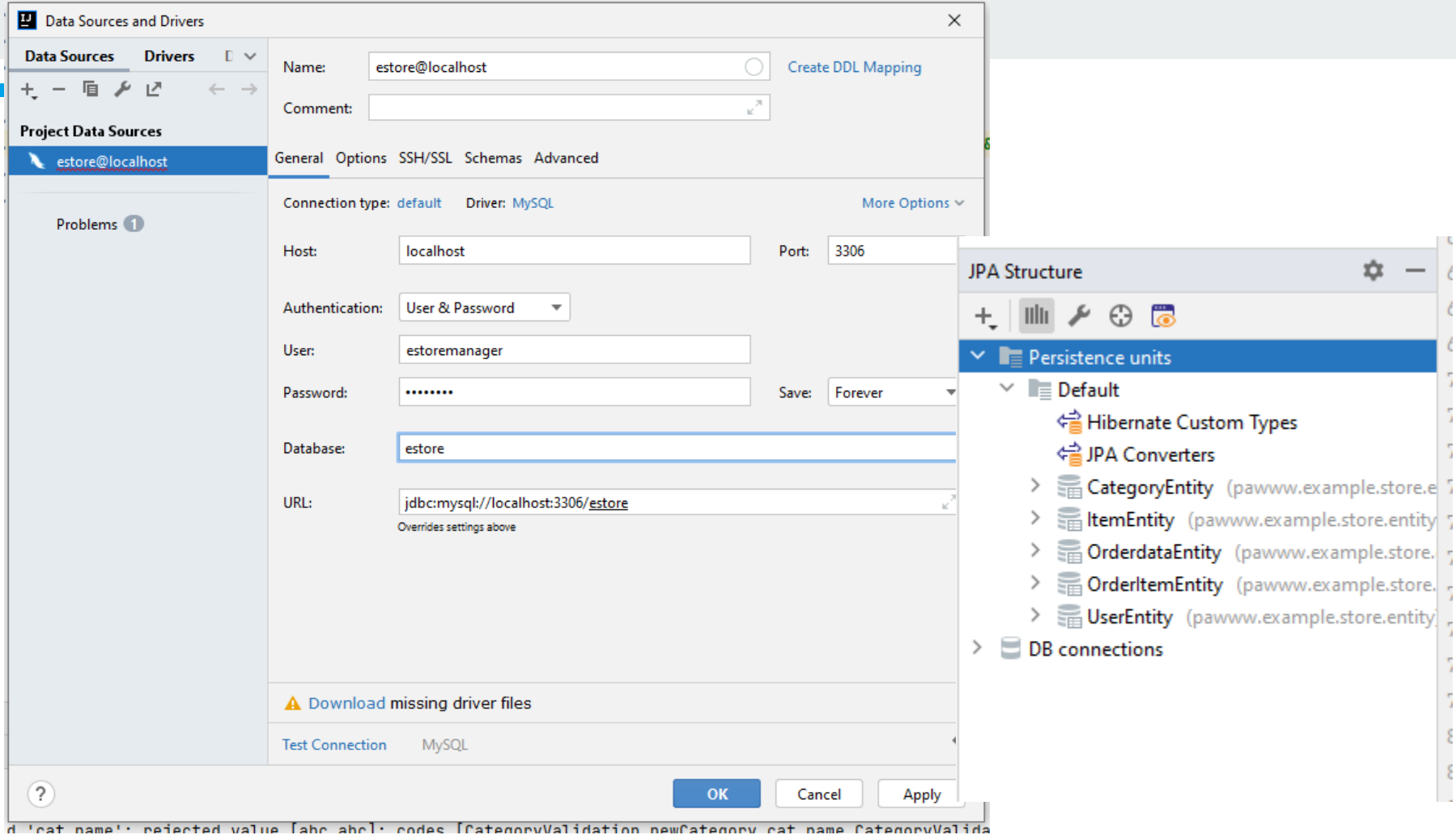
NoSQL

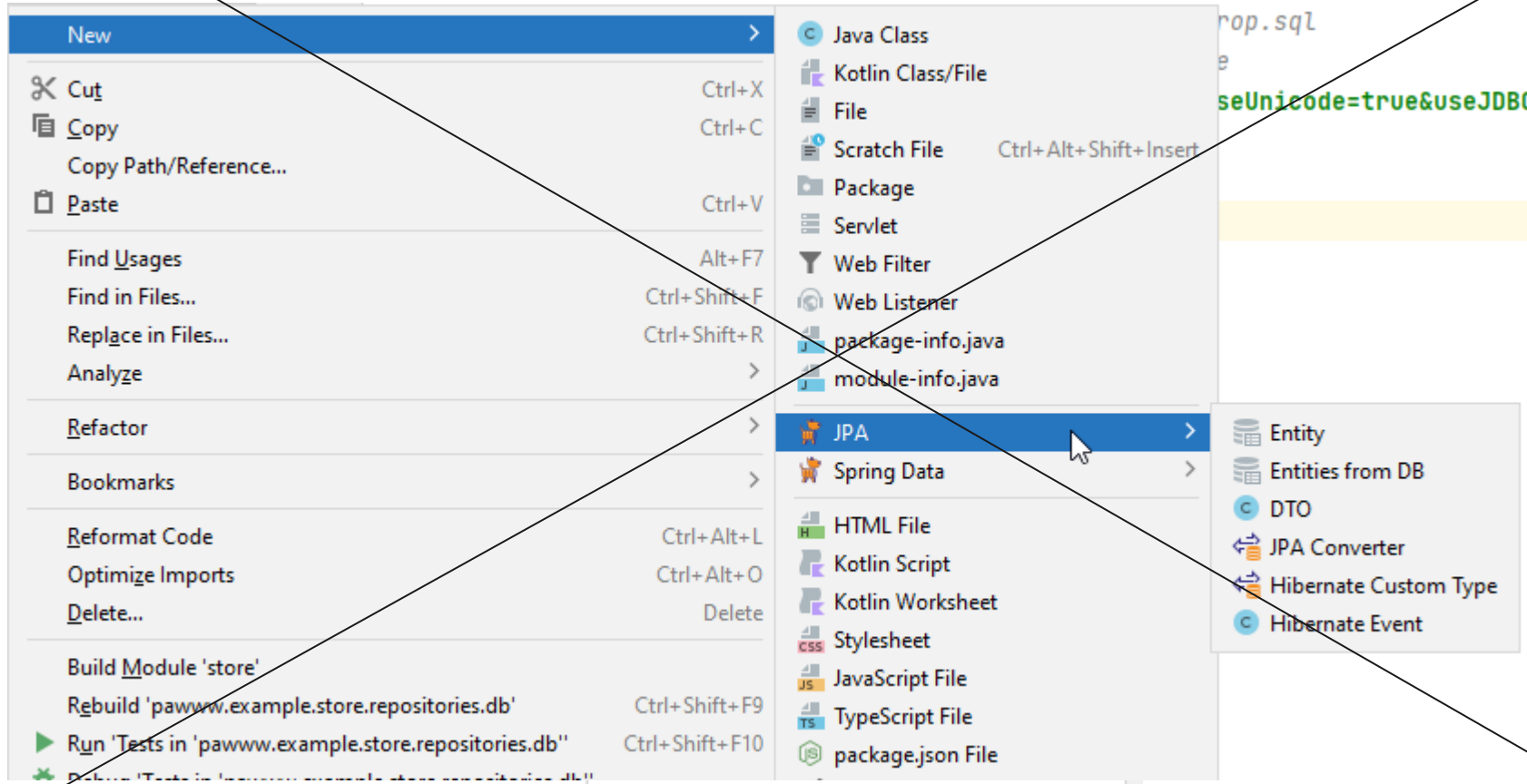
- [ ] **Redis**: Access Redis key-value data stores with Spring Data Redis
- [ ] **Reactive Redis**: Access Redis key-value data stores in a reactive fashion with Spring Data Redis

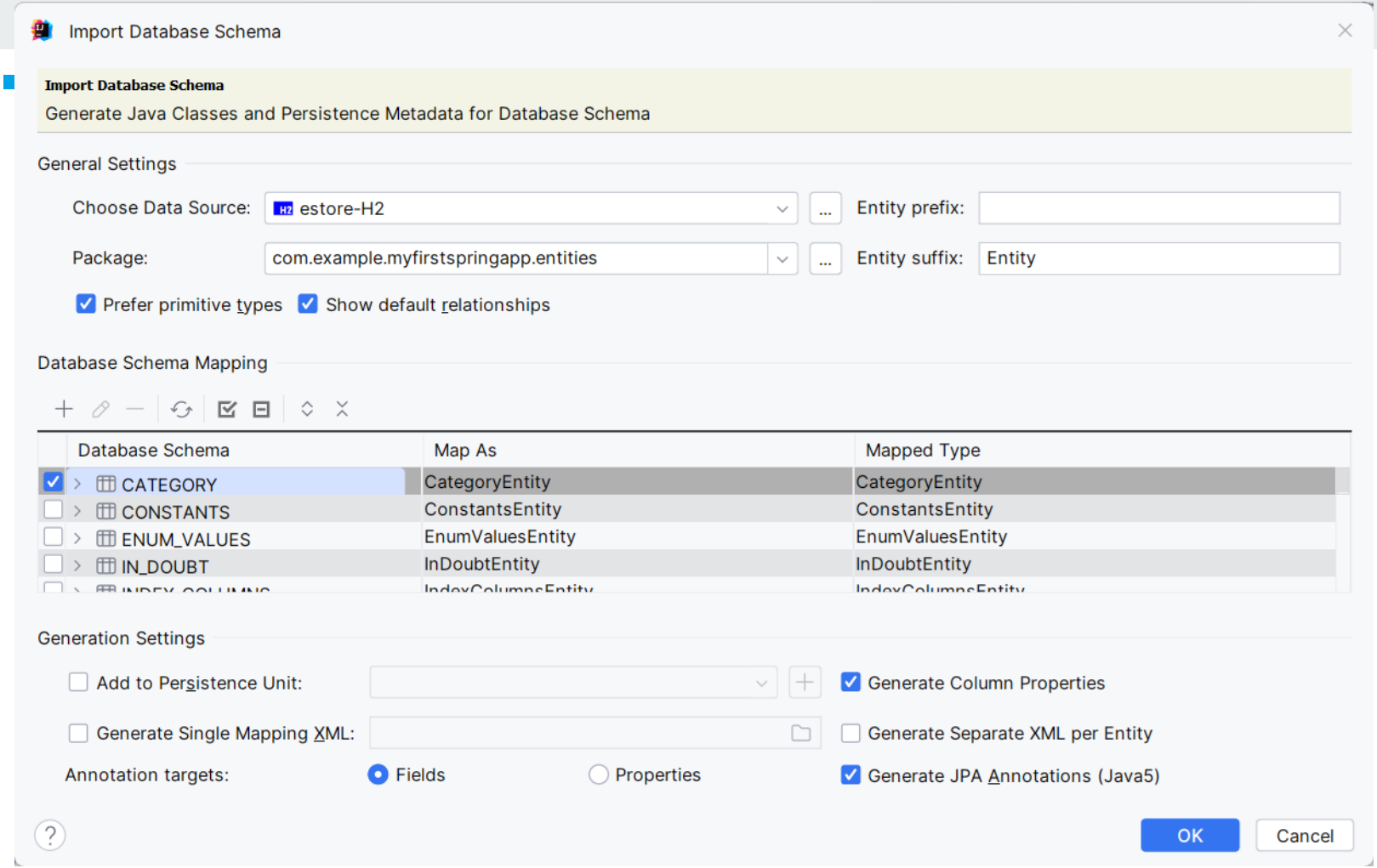
Kroki

1. Utworzenie bazy danych [utworzenie tabel+dane], użytkownika, nadanie praw, itp.
2. Utworzenie/generowanie klas encji
3. Konfiguracja dostępu do bazy
4. [Uruchomienie metody main → utworzenie tabeli]
5. Utworzenie repozytorium
6. Utworzenie serwisu
7. Metody w kontrolerze


Data Sources and Drivers
Data Sources
Drivers
Project Data Sources
estore@localhost
Problems 1
Name: estore@localhost
Create DDL Mapping
Comment:
General Options SSH/SSL Schemas Advanced
Connection type: default
Driver: MySQL
More Options
Host: localhost
Port: 3306
Authentication: User & Password
User: estoremanager
Password: ••••••••
Save: Forever
Database: estore
URL: jdbc:mysql://localhost:3306/estore
Overrides settings above
Download missing driver files
Test Connection
MySQL
OK
Cancel
Apply


Data Sources and Drivers
Data Sources
Drivers
Project Data Sources
estore@localhost
Problems 1
Name: estore@localhost
Create DDL Mapping
Comment:
General
Options
SSH/SSL
Schemas
Advanced
Connection type: default
Driver: MySQL
More Options
Host: localhost
Port: 3306
Authentication: User & Password
User: estoremanager
Password: ••••••••
Save: Forever
Database: estore
URL: jdbc:mysql://localhost:3306/estore
Overrides settings above
Download missing driver files
Test Connection
MySQL
OK
Cancel
Apply
JPA Structure
Persistence units
Default
Hibernate Custom Types
JPA Converters
CategoryEntity (pawww.example.store.e
ItemEntity (pawww.example.store.entity
OrderdataEntity (pawww.example.store.
OrderItemEntity (pawww.example.store.
UserEntity (pawww.example.store.entity
DB connections


New
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+F7
Find in Files... Ctrl+Shift+F
Replace in Files... Ctrl+Shift+R
Analyze
Refactor
Bookmarks
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Build Module 'store'
Rebuild 'pawww.example.store.repositories.db' Ctrl+Shift+F9
Run 'Tests in 'pawww.example.store.repositories.db'' Ctrl+Shift+F10
Java Class
Kotlin Class/File
File
Scratch File Ctrl+Alt+Shift+Insert
Package
Servlet
Web Filter
Web Listener
package-info.java
module-info.java
JPA
Spring Data
HTML File
Kotlin Script
Kotlin Worksheet
Stylesheet
JavaScript File
TypeScript File
package.json File
Entity
Entities from DB
DTO
JPA Converter
Hibernate Custom Type
Hibernate Event
rop.sql
seUnicode=true&useJDBC


Import Database Schema
Import Database Schema
Generate Java Classes and Persistence Metadata for Database Schema
General Settings
Choose Data Source:
estore-H2
Entity prefix:
Package:
com.example.myfirstspringapp.entities
Entity suffix:
Entity
Prefer primitive types
Show default relationships
Database Schema Mapping
Database Schema
Map As
Mapped Type
CATEGORY
CategoryEntity
CategoryEntity
CONSTANTS
ConstantsEntity
ConstantsEntity
ENUM_VALUES
EnumValuesEntity
EnumValuesEntity
IN_DOUBT
InDoubtEntity
InDoubtEntity
Generation Settings
Add to Persistence Unit:
Generate Column Properties
Generate Single Mapping XML:
Generate Separate XML per Entity
Annotation targets:
Fields
Properties
Generate JPA Annotations (Java5)
OK
Cancel

JPA - Java Persistence API

```java
import javax.persistence.*;

@Entity
@Table(name = "item")
public class Item {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Integer id;

    @Column(name = "name", nullable = false, length = 50)
    private String name;

    @Column(name = "price", nullable = false)
    private Double price;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "category", nullable = false)
    private Category category;

    public Integer getId() { return id; }

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

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }
```

JPA wymaga od encji pozbawionego argumentów konstruktora

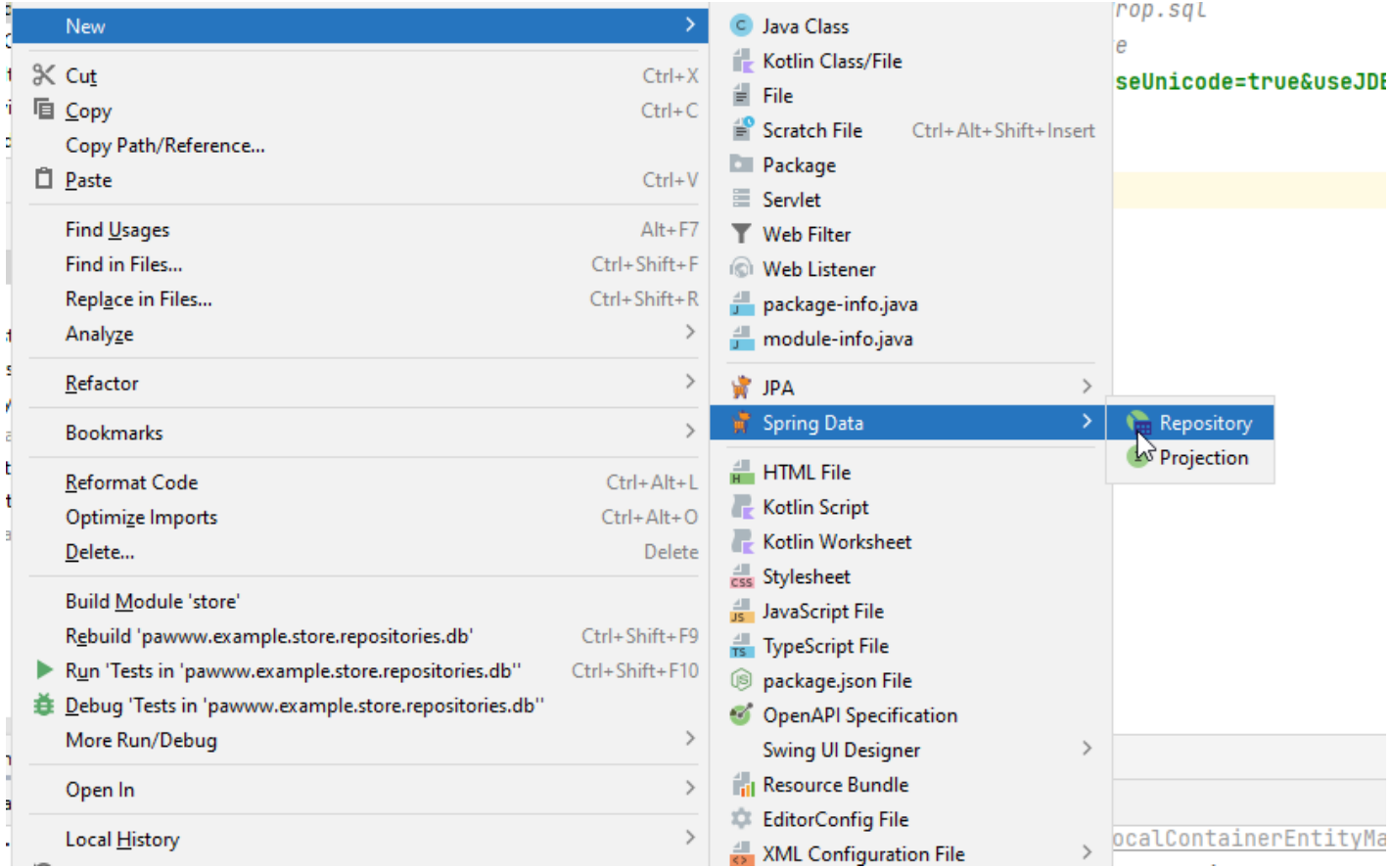
3. Konfiguracja dostępu do bazy

Plik application.properties

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/estore
spring.datasource.username=estoremanager
spring.datasource.password=password
```

```
spring.datasource.url=jdbc:h2:file:/data/estore
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
spring.h2.console.enabled=true
```

4. Utworzenie repozytorium



4. Utworzenie repozytorium

Interfejs jest sparametryzowany. Pierwszy parametr określa typ encji trwale przechowywanej w repozytorium, natomiast drugi określa typ właściwości identyfikatora encji.

5. Utworzenie serwisu

```java
public class StoreService {
    @Autowired
    //ItemRepository items;
    ItemRepositoryDB items;
    @Autowired
    Cart cart;
    @Autowired
    CategoryRepositoryDB categories;



    public void addToCart(Item item){
        this.cart.addItem(item);
    }

    public Item isInCart(String name){
        return this.cart.getItem(name);
    }


//    public Item getItem(int id){
//        return  items.getItem(id);
//    }

    public Item getItem(int id){
        Optional<Item> item=items.findById(id);
        return item.isEmpty()? null:item.get();
    }
```

???

JPA - Java Persistence API

StoreService.java | pom.xml (store) | Item.java | CartController.java | ItemEntity.java | ItemEntityRepository.java

```
1  package pawww.example.store.repositories.db;
2
3  import ...
5
6  public interface ItemEntityRepository extends JpaRepository<ItemEntity, I
7      List<ItemEntity> findAllByCategory
8  }
```

```
CategoryByCategoryAfter
CategoryByCategoryBefore
CategoryByCategoryBetween
CategoryByCategoryContaining
CategoryByCategoryContains
CategoryByCategoryEndingWith
CategoryByCategoryEndsWith
CategoryByCategoryEquals
CategoryByCategoryExists
CategoryByCategoryFalse
CategoryByCategoryGreaterThan
```

Ctrl+Down and Ctrl+Up will move caret down and up in the editor Next Tip

Podczas generowania implementacji repozytorium, Spring Data analizuje wszystkie metody w interfejsie repozytorium, przetwarza ich nazwy i próbuje ustalić przeznaczenie danej metody w kontekście obiektu Można powiedzieć, że Spring Data definiuje coś w rodzaju miniaturowego języka specjalizowanego (ang. domain-specific language).

Dostosowanie do własnych potrzeb repozytoriów JPA

```
public interface IStudentsRepositoryDB_JPA extends CrudRepository<Student, Integer>{

}
```

```
@Repository
public interface IThesisRepositoryDB_JPA extends CrudRepository<Thesis, Integer>{

    public Thesis findByTitle(String title);
    public List<Thesis> findByTitleContaining(String partOfTitle);
    //@Query(value="select * from Thesis t order by length(t.description) desc
    @Query("select t from Thesis t where length(t.description)= (select max(le
    public List<Thesis> findByLongestDesc();

}
```

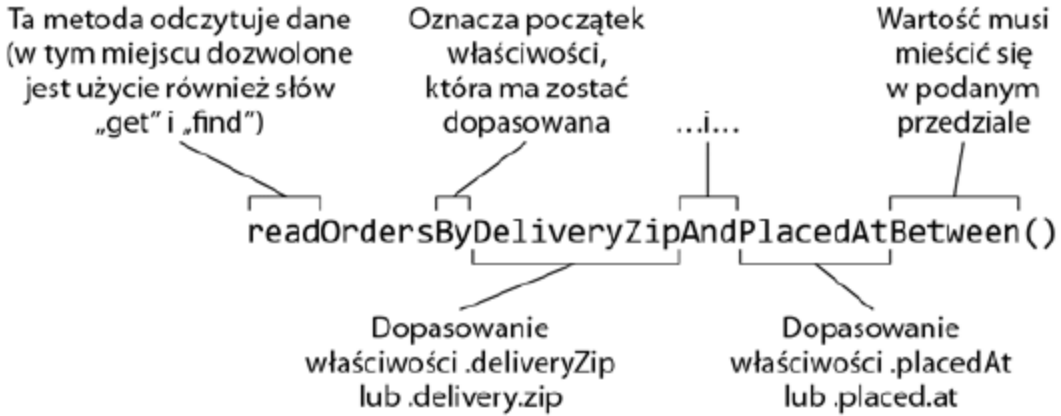
```
    }

    public Iterable<Thesis> findByTitle(String partOfTitle) {
        List<Thesis> theses=repository.findByTitleContaining(partOfTitle);
        System.err.println("title "+theses.size()+" "+theses.get(0).getDescription
        return theses;
    }

    public Thesis save(Thesis thesis) {
        repository.save(thesis);
        return thesis;
    }

}
```

Podczas generowania implementacji repozytorium, Spring Data analizuje wszystkie metody w interfejsie repozytorium, przetwarza ich nazwy i próbuje ustalić przeznaczenie danej metody w kontekście obiektu Można powiedzieć, że Spring Data definiuje coś w rodzaju miniaturowego języka specjalizowanego (ang. domain-specific language).

Dostosowanie do własnych potrzeb repozytoriów JPA

```
public interface IStudentsRepositoryDB_JPA extends CrudRepository<Student, Integer>{

}
```

```
@Repository
```

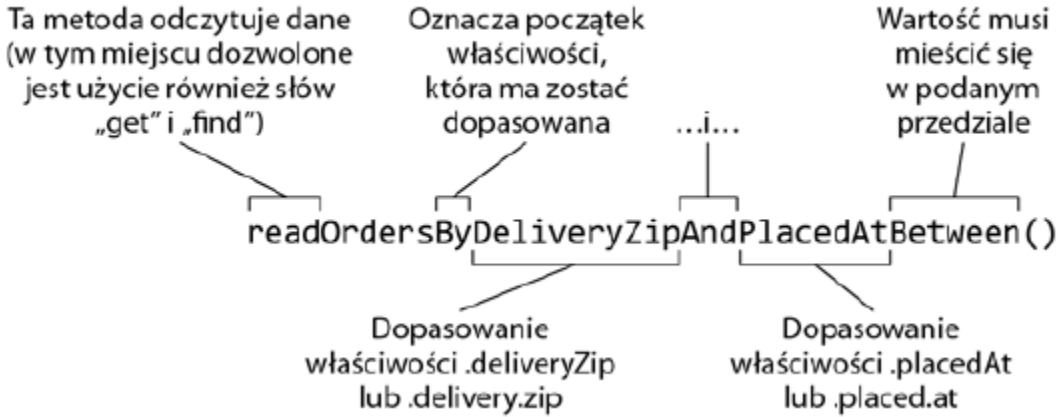



```
 nativeQuery = true)
scription)) from Thesis tt)")
```

```
        System.err.println("title "+theses.size()+" "+theses.get(0).getDescription());
        return theses;
    }

    public Thesis save(Thesis thesis) {
        repository.save(thesis);
        return thesis;
    }

}
```

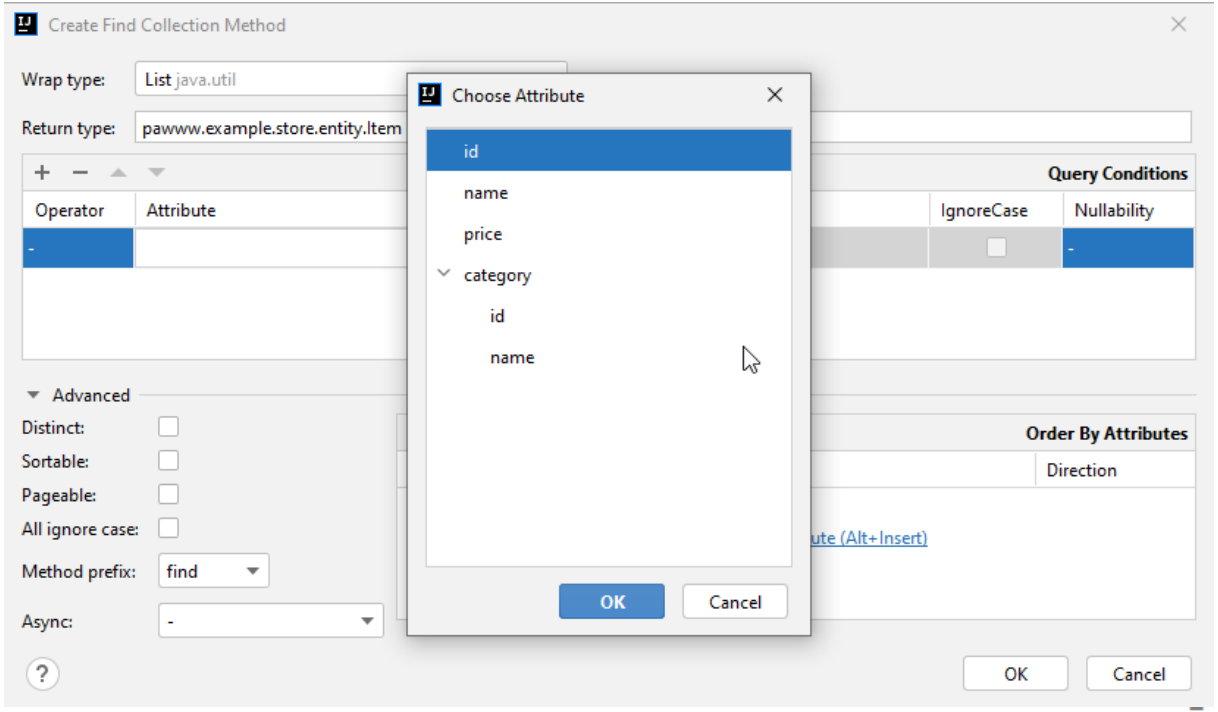
Nazwa metody repozytorium składa się z czasownika w języku angielskim, opcjonalnego podmiotu, słowa By i predykatu.

Dostosowanie do własnych potrzeb repozytoriów JPA



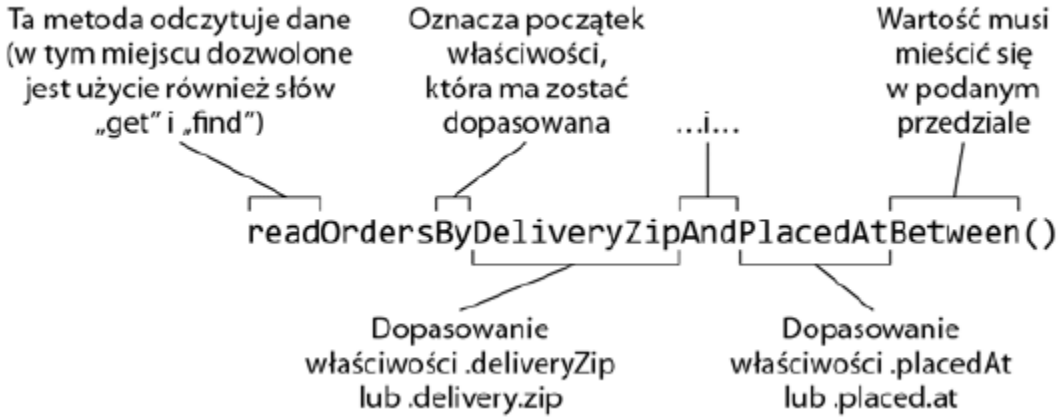
- IsAfter, After, IsGreaterThan, GreaterThan.
- IsGreaterThanEqual, GreaterThanEqual.
- IsBefore, Before, IsLessThan, LessThan.
- IsLessThanEqual, LessThanEqual.
- IsBetween, Between.
- IsNull, Null.
- IsNotNull, NotNull.
- IsIn, In.
- IsNotIn, NotIn.
- IsStartingWith, StartingWith, StartsWith.
- IsEndingWith, EndingWith, EndsWith.
- IsContaining, Containing, Contains.
- IsLike, Like.
- IsNotLike, NotLike.
- IsTrue, True.
- IsFalse, False.
- Is, Equals.
- IsNot, Not.
- IgnoringCase, IgnoresCase.

Dostosowanie do własnych potrzeb repozytoriów JPA




```
IsAfter, After, IsGreaterThan, GreaterThan.
IsGreaterThanEqual, GreaterThanEqual.
IsBefore, Before, IsLessThan, LessThan.
IsLessThanEqual, LessThanEqual.
IsBetween, Between.
IsNull, Null.
IsNotNull, NotNull.
IsIn, In.
IsNotIn, NotIn.
IsStartingWith, StartingWith, StartsWith.
IsEndingWith, EndingWith, EndsWith.
IsContaining, Containing, Contains.
IsLike, Like.
IsNotLike, NotLike.
IsTrue, True.
IsFalse, False.
Is, Equals.
IsNot, Not.
IgnoringCase, IgnoresCase.
```

Dostosowanie do własnych potrzeb repozytoriów JPA



zadanie

- IsAfter, After, IsGreaterThan, GreaterThan.
- IsGreaterThanEqual, GreaterThanEqual.
- IsBefore, Before, IsLessThan, LessThan.
- IsLessThanEqual, LessThanEqual.
- IsBetween, Between.
- IsNull, Null.
- IsNotNull, NotNull.
- IsIn, In.
- IsNotIn, NotIn.
- IsStartingWith, StartingWith, StartsWith.
- IsEndingWith, EndingWith, EndsWith.
- IsContaining, Containing, Contains.
- IsLike, Like.
- IsNotLike, NotLike.
- IsTrue, True.
- IsFalse, False.
- Is, Equals.
- IsNot, Not.
- IgnoringCase, IgnoresCase.

Spring Security

Programowanie aplikacji WWW w języku Java

Przykład

Sklep internetowy, funkcjonalności:

a) **użytkownik -** przeglądanie produktów (nazwa, cena, kategoria)
b) wkładanie, wyjmowanie produktów do koszyka (sesja)
c) **użytkownik zalogowany -** kupowanie
d) przeglądanie historii zamówień
e) wyszukiwanie
f) filtrowanie
g) **admin -** zarządzanie produktami

Zastosowanie Spring Security

- Służy do zabezpieczenia aplikacji:
  - Uwierzytelnianie (logowanie)
  - Autoryzacja (sprawdzenie praw do zasobów)
- Konfiguracja interceptorów za pomocą potoku reguł
- Konfiguracja praw do zasobów na podstawie przypisanych ról/grup

Zależności

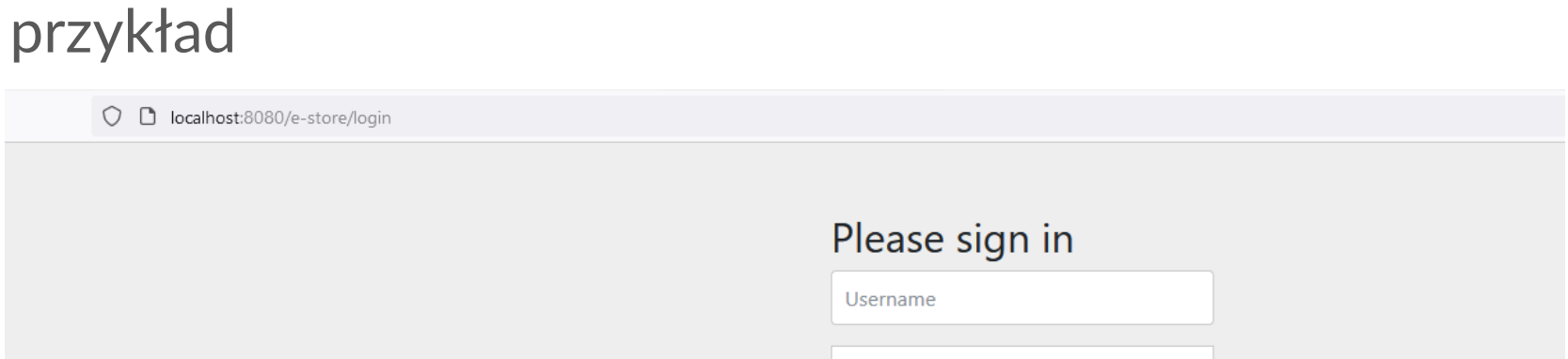
```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
```

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>  :ifactId>
    <version>3.1.1.RELEASE</version>
</dependency>
```

W ogólnym przypadku pierwsza zależność jest jedynym i wystarczającym elementem wymaganym do zapewnienia bezpieczeństwa aplikacji. Po uruchomieniu aplikacji konfiguracja automatyczna wykryje istnienie Spring Security w ścieżce dostępu klas i przeprowadzi podstawową konfigurację zabezpieczeń

Czy to wystarczy?

przykład




```
Using generated security password: 54029433-fa61-4018-9360-820bb91efcf0

This generated password is for development use only. Your security configuration must be updated before running your application in production.
```

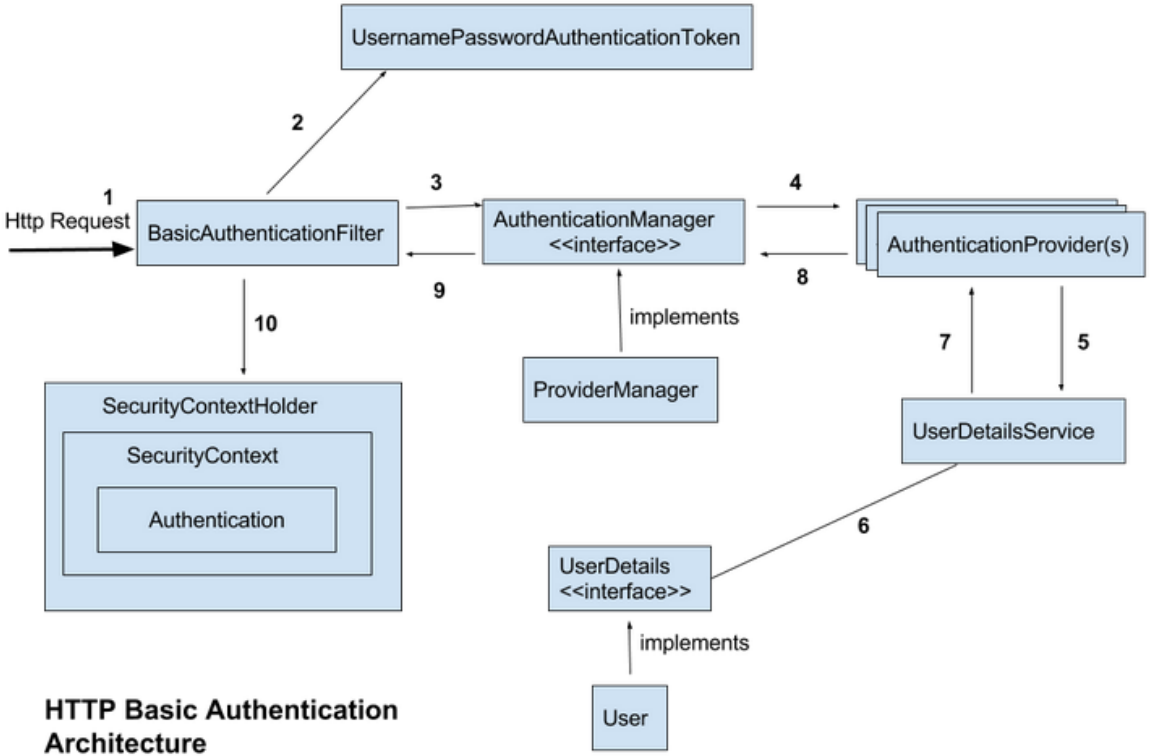
Czy to wystarczy?

- wszystkie ścieżki żądań HTTP wymagają uwierzytelnienia;
- nie są wymagane żadne specjalne role lub autoryzacje;
- nie ma strony rejestracji;
- uwierzytelnianie odbywa się za pomocą prostego formularza;
- istnieje tylko jeden użytkownik o nazwie user.

Schemat działania Spring Security

1. Logowanie
2. Weryfikacja danych w funkcji uwierzytelniającej (filtry)
3. Tworzony jest token uwierzytelnienia, który przekazany jest do wybranego dostawcy (provider) rozwiązania zabezpieczenia.
4. Dostawca zwraca status logowania oraz uprawnienia (role) użytkownika.
5. Umieszczenie statusu i uprawnień w kontekście aplikacji

Schemat działania Spring Security



Nasz cel

- obsługa uwierzytelniania za pomocą dedykowanej strony logowania
- zapewnienie obsługi wielu użytkowników i dodanie strony rejestracji nowego użytkownika.
- zastosowanie różnych reguł bezpieczeństwa dla odmiennych ścieżek żądań.

Spring Security

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends
WebSecurityConfigurerAdapter {
  @Bean
  public UserDetailsService userDetailsService() {
{...}}
@Bean
      public SecurityFilterChain
securityFilterChain(HttpSecurity http) throws Exception
```

konfiguracja usług związanych z uwierzytelnieniem użytkowników

konfiguracja łańcucha filtrów

Spring Security

magazyn danych użytkownika, m.in.:

- w pamięci magazyn danych użytkownika;
- oparty na JDBC/JPA magazyn danych użytkownika;
- oparty na LDAP magazyn danych użytkownika;
- niestandardowa usługa przechowywania informacji o użytkowniku

InMemoryUserDetailsManager

```
@Bean
public UserDetailsService userDetailsService() {
    UserDetails user =
            User.withUsername("user1")
                    .password("better")
                    .roles("USER")
                    .build();
    UserDetails admin =
            User.withUsername("admin")
                    .password("thebest")
                    .roles("ADMIN")
                    .build();
    System.out.println(user.getUsername()+" "+user.getPassword()+" "+user.getAuthorities());
    System.out.println(admin.getUsername()+" "+admin.getPassword()+" "+admin.getAuthorities());
    return new InMemoryUserDetailsManager(user,admin);
}
```

securityFilterChain(HttpSecurity http)

- access(String) - jeśli wartością wyrażenia SpEL jest true
- anonymous() - dostęp dla anonimowych użytkowników
- authenticated() - dostęp dla uwierzytelnionych użytkowników
- denyAll() - bezwarunkowo odmawia dostępu
- fullyAuthenticated() - dostęp po zalogowaniu (nie dla zapamiętanych)
- hasAnyAuthority(String...) - przynajmniej jedno z podanych uprawnień.
- hasAnyRole(String...) - przynajmniej jedną z podanych ról.
- hasAuthority(String) - jeśli posiada wybrane uprawnienie
- hasIpAddress(String) - jeżeli żądanie pochodzi z podanego adresu IP
- hasRole(String) - jeśli posiada podaną rolę.
- not() - neguje efekt wszystkich powyższych metod dostępu.
- permitAll() - bezwarunkowo przyznaje dostęp.
- rememberMe() - uwierzytelnienie z użyciem opcji zapamiętania

securityFilterChain(HttpSecurity http)

```java
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
            .authorizeHttpRequests((requests) -> requests
                    .requestMatchers("/").permitAll()
                    .anyRequest().authenticated()
            )
            .formLogin((form) -> form
                    .loginPage("/login")
                    .permitAll()
            )
            .logout((logout) -> logout.logoutSuccessUrl("/").permitAll());

    return http.build();
}
```

Strona logowania

W widoku logowania należy zwrócić uwagę na ścieżkę dostępu dla przekazywanego żądania oraz nazwy pól przeznaczonych do podania nazwy użytkownika i hasła. Domyślnie Spring nasłuchuje żądań logowania pod adresem `/login` i oczekuje, że nazwami pól dla nazwy użytkownika i hasła będą odpowiednio `username` i `password`. Można to skonfigurować. Przedstawiona tutaj konfiguracja zmienia ścieżkę dostępu i nazwy pól.

```
.and()
.formLogin()
.loginPage("/login")
.loginProcessingUrl("/authenticate")
.usernameParameter("user")
.passwordParameter("pwd")
```

.formLogin().loginPage()

SecurityConfig.java | login.html

History

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
      xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Spring Security Example </title>
    </head>
    <body>
        <div th:if="${param.error}">
            Invalid username and password.
        </div>
        <div th:if="${param.logout}">
            You have been logged out.
        </div>
        <form th:action="@{/login}" method="post">
            <div><label> User Name : <input type="text" name="username"/> </label></div>
            <div><label> Password: <input type="password" name="password"/> </label></div>
            <div><input type="submit" value="Sign In"/></div>
        </form>
    </body>
</html>
```

```
@Controller
public class LoginController {

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

}
```

Znacznik <sec:authorize/>

```
<div sec:authorize="isAuthenticated()">
    This content is only shown to authenticated users.
</div>
<div sec:authorize="hasRole('ROLE_ADMIN')">
    This content is only shown to administrators.
</div>
<div sec:authorize="hasRole('ROLE_USER')">
    This content is only shown to users.
</div>
```

znaczniki

obiekt

```
<div th:if="${#authorization.expression('hasRole(''ROLE_ADMIN'')')}">
    This will only be displayed if authenticated user has role ROLE_ADMIN.
</div>
```

Znacznik <sec:authentication/>

```
Logged user: <span sec:authentication="name">Bob</span><br/>
Roles: <span sec:authentication="authorities">USER</span>
```

znaczniki

obiekt

```
<h2 th:text="|You are logged as ${#authentication.name}|">
    The value of the "name" property of the authentication object should appear here.
</h2>
```

thymeleaf-extras-springsecurity6

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity6">

...
<span sec:authorize="isAuthenticated()">
 <form th:action="@{/logout}" method="post">
   <input type="submit" value="Sign Out" />
 </form>
   <h1 sec:authorize="isAuthenticated()">Hello <span
sec:authentication="name"></span></h1>
</span>
<div sec:authorize="!isAuthenticated()">
 <form th:action="@{/login}" method="post">
   <input type="submit" value="Log in" />
 </form>
</div>
```

Znacznik <sec:authorize-url/>

```
<div sec:authorize-url="/confidentData">
    This will only be displayed if authenticated user can call the "/confidentData" URL.
    <p><a th:href="@{/confidentData}" >Confident data</a></p>
</div>
<div sec:authorize-url="GET /confidentData">
    This will only be displayed if authenticated user can call the "/confidentData" URL.
    <p><a th:href="@{/confidentData}" >Confident data</a></p>
</div>
```

sec:authorize i sec:authorize-url

```
<a sec:authorize-url="/items/" th:href="@{/items/}" th:text="#{menu.print_all.msg}">Print items</a>

<a sec:authorize="hasRole('ROLE_USER')" th:href="@{/cart/}" th:text="#{menu.show_cart.msg}">Show
cart</a>

<a sec:authorize-url="/items/category/add" th:href="@{/items/category/add}" th:text="'Add
category'">Add category</a>

<a sec:authorize="hasRole('ROLE_ADMIN')" th:href="@{/items/manage/add}" th:text="'Add item'">Add
item</a>
```

oparty na JDBC/JPA magazyn danych użytkownika

konfiguracja

Schemat bazy link

```
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(50) not null,
    enabled boolean not null);

create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fk_authorities_users foreign key(username) references users(username));
    create unique index ix_auth_username on authorities (username,authority);
```

oparty na JDBC/JPA magazyn danych użytkownika

```java
@Entity
@Table(name = "USERS", schema = "PUBLIC", catalog = "ESTORE")
@Data
public class UsersEntity {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "USERNAME", nullable = false, length = 50)
    private String username;
    @Basic
    @Column(name = "PASSWORD", nullable = false, length = 200)
    private String password;
    @Basic
    @Column(name = "ENABLED", nullable = false)
    private byte enabled;
}
```

oparty na JDBC/JPA magazyn danych użytkownika

```java
@Entity
@Table(name = "AUTHORITIES", schema = "PUBLIC", catalog = "ESTORE")
@Data
public class AuthoritiesEntity {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "ID", nullable = false)
    private int id;
    @Basic
    @Column(name = "USERNAME", nullable = false, length = 50,insertable=false, updatable=false)
    private String username;
    @Basic
    @Column(name = "AUTHORITY", nullable = false, length = 50)
    private String authority;
}
```

oparty na JDBC/JPA magazyn danych użytkownika

```
public interface UsersEntityRepository extends JpaRepository<UsersEntity,
String> {
   UsersEntity findByUsername(String username);
}
```

```
 public interface AuthoritiesEntityRepository extends
 JpaRepository<AuthoritiesEntity, Integer> {
   List<AuthoritiesEntity> findByUsername(String username);
 }
```

oparty na JDBC/JPA magazyn danych użytkownika

```
@Service
public class UsersAuthDBService  implements UserDetailsService {
    @Autowired
    UsersEntityRepository users;
    @Autowired
    AuthoritiesEntityRepository authorities;
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UsersEntity user = users.findByUsername(username)!=null?users.findByUsername(username):null;
        if(user==null)
            throw new UsernameNotFoundException("User " +username +" not found !");
        else {
            UserDetails userDetails=new org.springframework.security.core.userdetails.User(
                    user.getUsername(),
                    user.getPassword(),
                    authorities.findByUsername(username)
                            .stream()
                            .map(role-> {
                                System.out.println("autorities "+ role.getAuthority());
                                return new SimpleGrantedAuthority(role.getAuthority());
                            })
                            .collect(Collectors.toSet()));

            return userDetails;

        }
```

na JDBC) JPA magazyn danych użytkownika

```
@Service
public class UsersAuthDBService  implements UserDetailsService {
    @Autowired
    UsersEntityRepository users;
    @Autowired
    AuthoritiesEntityRepository authorities;
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UsersEntity user = users.findByUsername(username)!=null?users.findByUsername(username):null;
        if(user==null)
            throw new UsernameNotFoundException("User " +username +" not found !");

                                                    org.springframework.security.core.userdetails.User(


                                                    ername(username)

                                                    {
                                                    ut.println("autorities "+ role.getAuthority());
                                                    ew SimpleGrantedAuthority(role.getAuthority());

                                                    lectors.toSet()));
}
```

Metoda `loadByUsername()`
nigdy nie może zwrócić wartości `null`.

Dlatego też jeśli wynikiem wywołania `findByUsername()`
jest `null`, metoda zgłosi wyjątek `UsernameNotFoundException`.

W razie powodzenia zwrócony będzie znaleziony obiekt `User`.

oparty na JDBC/JPA magazyn danych użytkownika

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

@Bean
public PasswordEncoder passwordEncoder() {
   int rounds = 12;
   return new BCryptPasswordEncoder(rounds);
}
   @Bean
   public AuthenticationManager authenticationManager(UserDetailsService customUserDetailsService)
{
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(customUserDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        List<AuthenticationProvider> providers =  List.of(authProvider);
        return new ProviderManager(providers);
   }
```

Szyfrowanie haseł w kontrolerze

```java
@Controller
public class LoginController {

    @Autowired
    BCryptPasswordEncoder bCryptpasswordEncoder;


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

    @GetMapping("/register")
    public String register(@RequestParam String username, @RequestParam String password){
        System.out.println("Register data "+username+" "+password);
        System.out.println("Encrypted password "+ bCryptpasswordEncoder.encode(password));
        return "login";
    }
```

Serwisy REST

Programowanie aplikacji WWW w języku Java

Przykład

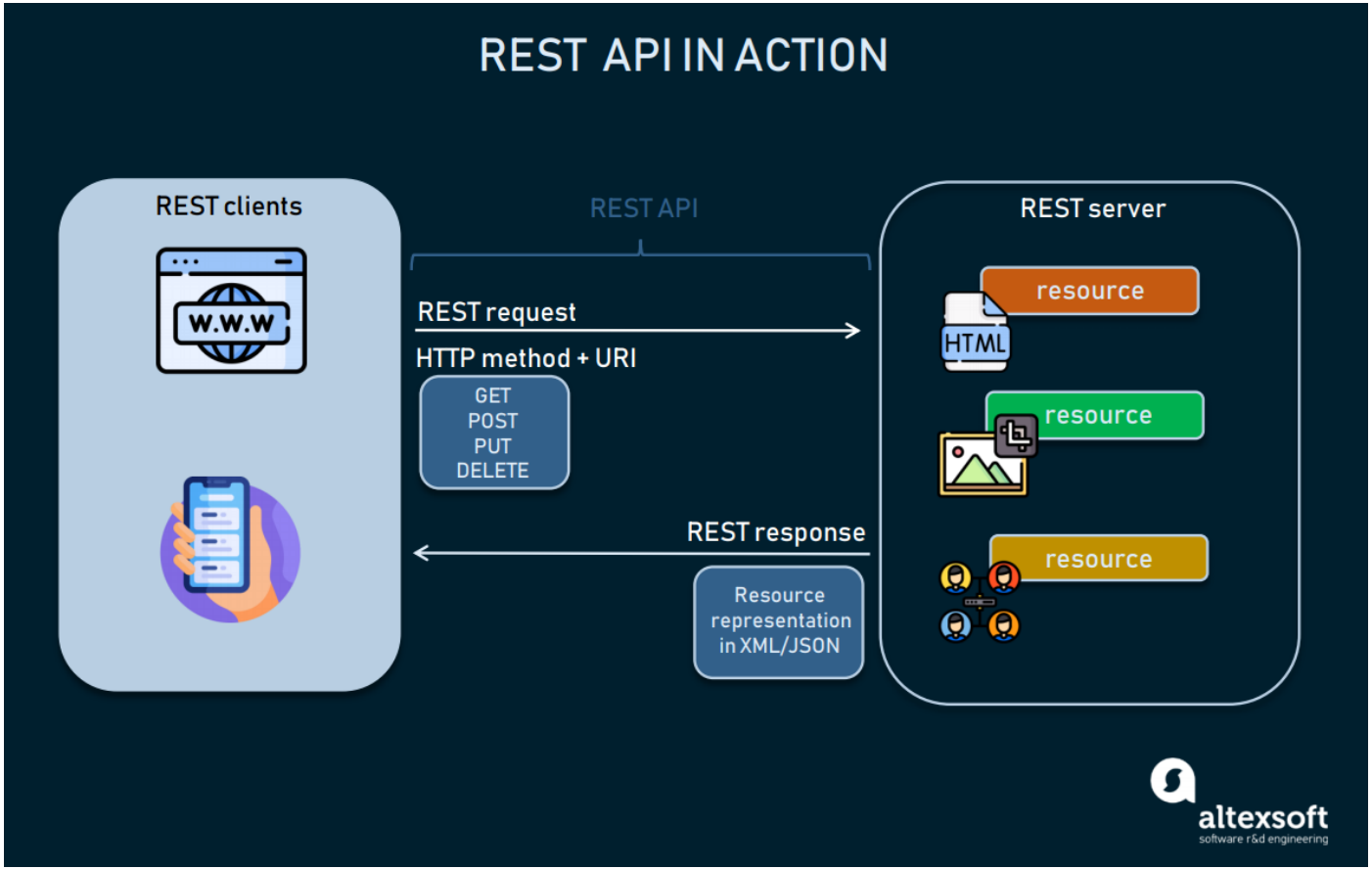
Sklep internetowy, funkcjonalności:

a) użytkownik - przeglądanie produktów (nazwa, cena, kategoria)
b) wkładanie, wyjmowanie produktów do koszyka (sesja)
c) użytkownik zalogowany - [illegible]
d) przeglądanie historii zamówień
e) wyszukiwanie
f) filtrowanie
g) admin - zarządzanie produktami

dodatkowe uwierzytelnianie

Pobieranie danych od producentów z autoryzacją

Architektura REST



Serwisy REST

Ewentualnie kontroler mógłby zostać oznaczony adnotacją @Controller, podobnie jak każdy inny kontroler w Spring MVC. Jednak wówczas wszystkie metody procedur obsługi trzeba by było oznaczyć za pomocą adnotacji `@ResponseBody`, aby otrzymać ten sam wynik. Jeszcze inną możliwością jest zwrot obiektu `ResponseEntity`.

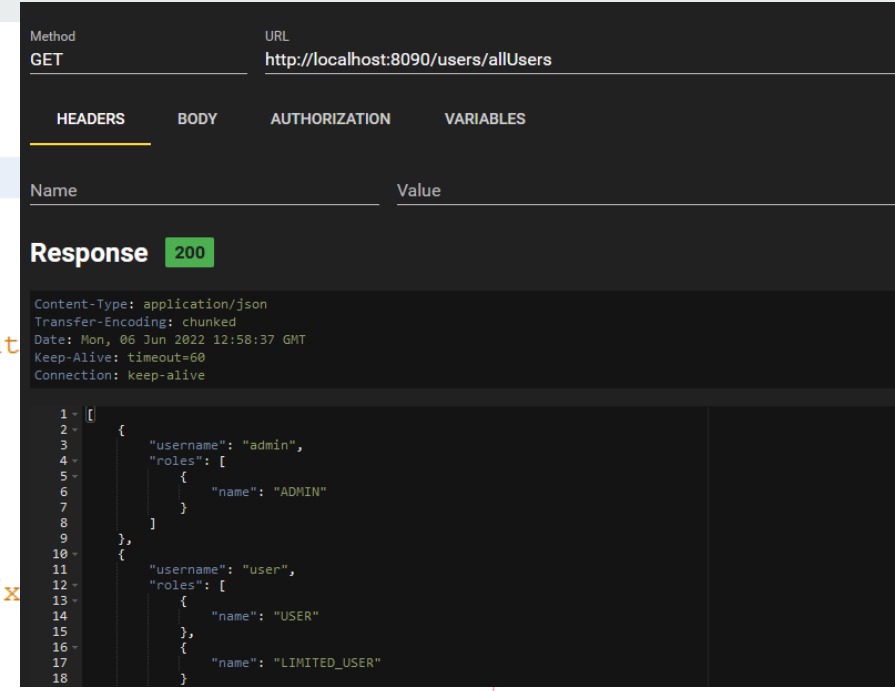
```
@RestController
@RequestMapping("/items")
public class NewRestController {
    @Autowired
    ItemsRep items;

    @RequestMapping(method=RequestMethod.GET,produces={"application/xml","application/json"})
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody List<Item> list() {
        return items.getItems();
    }
```

????

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>

<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
</dependency>
```

Serwisy REST

```java
@RestController
@RequestMapping("/items")
public class NewRestController {
    @Autowired
    ItemsService items;

    @GetMapping(produces={"application/json","applicat
    @ResponseStatus(HttpStatus.OK)
    public List<NewItem> list() {
        return items.getAll();
    }

    @GetMapping(value="/{id}",produces={"application/x
    @ResponseStatus(HttpStatus.OK)
    public NewItem get(@PathVariable int id) {
        return items.getItem(id);
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> put(@PathVariable int id, @RequestBody NewItem input) {
        items.updateItem(id, input);
        return new ResponseEntity<>("Product modified",HttpStatus.OK);
```



Serwisy REST

```
@RestController
@RequestMapping("/items")
public class NewRestController {
    @Autowired
    ItemsService items;

    @GetMapping(produces={"application/json","application/xml"})
    @ResponseStatus(HttpStatus.OK)
    public List<NewItem> list() {
        return items.getAll();
    }

    @GetMapping(value="/{id}",produces={"application/xml","application/json"})
    @ResponseStatus(HttpStatus.OK)
    public NewItem get(@PathVariable int id) {
        return items.getItem(id);
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> put(@PathVariable int id, @RequestBody NewItem input) {
        items.updateItem(id, input);
        return new ResponseEntity<>("Product modified",HttpStatus.OK);
```

Serwisy REST

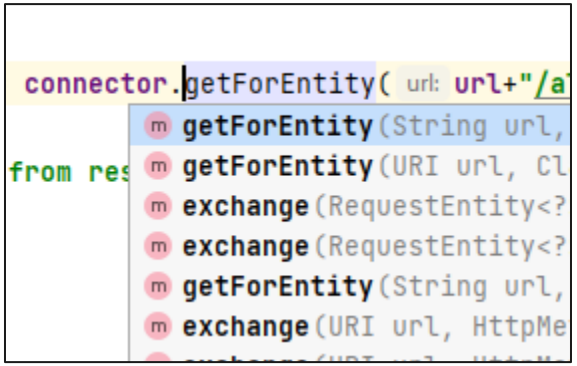
```java
@PutMapping(value = "/modify",consumes = "application/json")
@ResponseStatus(HttpStatus.ACCEPTED)
public User modify(@RequestBody User user) {
    try {
        return service.updateRoles(user);
    } catch(BadRequestBody ex){
        throw new ResponseStatusException((HttpStatus.NO_CONTENT));
    }
}
```

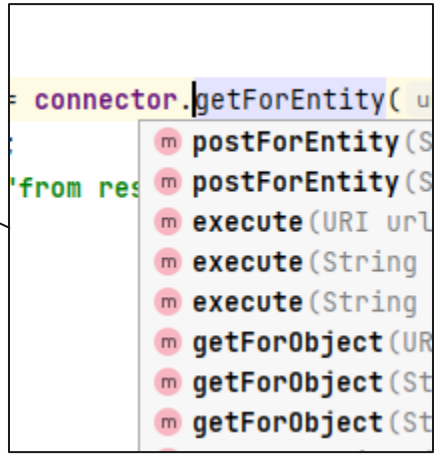
BadRequestBody.java | UserDetailsServiceImpl.java | UsersRestController.java | pom.xml (storeAuth)

```java
package pawww.example.store.auth.storeauth.controller.exceptions;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value= HttpStatus.BAD_REQUEST, reason = "The object contains no data")
public class BadRequestBody extends RuntimeException{

}
```

Klient RestTemplate

```java
@Component
public class RestConnector {
    @Autowired
    UserRepository userRepository;

    @Autowired
    RestTemplate connector;

    String url="http://localhost:8090/users";

    public boolean checkUser(String username) {
        ResponseEntity<String> response = connector.getForEntity(
                url: url+"/checkusername/"+username, String.class);
        System.out.println(response.toString());
        return true;
        }
```





Klient RestTemplate - logowanie metodą POST

```
@Component
public class RestConnector {
    @Autowired
```

```
public RestUser getUser(String username, String password) {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);

    MultiValueMap<String, String> map= new LinkedMultiValueMap<String, String>();
    map.add("username", username);
    map.add("password", password);

    HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<MultiValueMap<String, String>>(map, headers);

    RestUser user= connector.postForObject(
            url: url+"/getuser",request, RestUser.class);
    if(user!=null) System.out.println("from rest client "+user.getUsername()+" "+user.getRoles());
    return user;

}
```

Serwer RestTemplate - logowanie metodą POST

```
@PostMapping("/getUser")
public ResponseEntity<?> getUser(
        @RequestParam("username") final String username, @RequestParam("password") final String password) {
    User user=this.service.findUser(username);
    if(user==null) return new ResponseEntity<>( body: "User not found",HttpStatus.NO_CONTENT);
    System.out.println(user.getId()+" "+user.getPassword()+" "+password);
    if(user.getPassword().equals(password)){
        return new ResponseEntity<>(user,HttpStatus.FOUND);
    }else return new ResponseEntity<>( body: "User not found",HttpStatus.NO_CONTENT);

}
```

```
    RestUser user= connector.postForObject(
            url: url+"/getuser",request, RestUser.class);
    if(user!=null) System.out.println("from rest client "+user.getUsername()+" "+user.getRoles());
    return user;

}
```

Klient RestTemplate
Z uwierzytelenieniem

```
HttpHeaders createHeaders(String username, String password){
    return new HttpHeaders() {{
        String auth = username + ":" + password;
        byte[] encodedAuth = Base64.encodeBase64(
                auth.getBytes(Charset.forName("US-ASCII")) );
        String authHeader = "Basic " + new String( encodedAuth );
        set( "Authorization", authHeader );
    }};
}
```

```
public List<RestUser> getUsers() {

    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if(auth==null) return null;
    Collection<? extends GrantedAuthority> roles=auth.getAuthorities();
    String username=auth.getName();
    User user=userRepository.findById(username).get();
    String password=user.getPassword();
    System.out.println(username+" "+password);
    if(roles.contains(new SimpleGrantedAuthority( role: "ROLE_ADMIN"))) {
        ResponseEntity<RestUser[]> response=connector.exchange
                ( url: url+"/allUsers", HttpMethod.GET, new HttpEntity<>(createHeaders(username, password)), RestUser[].class);
        RestUser[] users=response.getBody();
        if(users!=null) System.out.println("from rest client "+users.length);
        return Arrays.asList(users);

    }
   return  null;
```

Zabezpieczenie żądań

```
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    UserDetailsServiceImpl userDetailsService;

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable() HttpSecurity
                .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
                .antMatchers( ...antPatterns: "/all").permitAll()
                .antMatchers(HttpMethod.POST, ...antPatterns: "/register").permitAll()
                .antMatchers( ...antPatterns: "/allUsers").hasRole("ADMIN")
                .antMatchers(HttpMethod.PUT, ...antPatterns: "/modify").hasRole("USER")
                .anyRequest().authenticated()
                .and().httpBasic();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());

    }
```

```
@RestController
@RequestMapping(path="/")
@CrossOrigin(origins="*")
public class UsersRestController {
```

Zabezpieczanie na poziomie metod aplikacji

```
@RestController
@RequestMapping("/rest")
public class SecureRestController {
```

pom.xml [wyklad13] | secDemo.html | hello.html | login.html

```
    }

    @PreAuthorize("hasRole('ROLE_USER')")
    @PutMapping("/{id}")
    public ResponseEntity<?> put(@PathVariable String id, @RequestBody Item input) {
        return new ResponseEntity<>("Product created",HttpStatus.CREATED);
    }

    @PreAuthorize("hasRole('ROLE_ADMIN')")
    @PostMapping
    public ResponseEntity<?> post(@RequestBody Item input) {
        return new ResponseEntity<>("Product modified",HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable String id) {
        return new ResponseEntity<>("Product deleted",HttpStatus.MOVED_PERMANENTLY);
    }
```

Testowanie komunikacji