1. **CONFIGURARE MYSQL**

* deschidem MySql
* creem Baza de date ”AdrianGym”
* creem user ”Radu” pentru aceasta baza de date - Gives all privileges to the new user

Nu este nevoie sa creem tabele deoarece Hibernate stie sa creeze tabele din clasele Entity (Membru) daca configuram : spring.jpa.hibernate.ddl-auto=update

Hibernate automatically translates the entity into a table.

1. **Create the Repository - MembruRepository.java**

You need to create the repository that holds user records, as the following listing

import .......Membru;

// This will be AUTO IMPLEMENTED by Spring into a Bean called membruRepository

// CRUD refers Create, Read, Update, Delete

public interface MembruRepository extends CrudRepository<Membru, Long> {

}

Deci in repository noi importam clasa noastra entity (Membru) si definim interfata MembruRepository extends CrudRepository. Spring va crea el automat un bean ( membruRepository) care va crea instantele Membru. ( bean este un sablon dupa care se creeaza instantele). Cum era la noi repository.save(newMembru) nu e nevoie sa implementam save deoarece el se genereaza automat.

Spring automatically implements this repository interface in a bean that has the same name (with a change in the case — it is called  membruRepository.

1. Testare User

* lansez Mysql din XAMPP
* Cmd in c:\xampp\mysql\bin
* c:\xampp\mysql\bin\>mysql -u radu -p12345
* MariaDB [(none)]> show databases;

1. Install mvn ( pentru creare fisier war la instalare pe server extern)

mvn clean install

[Maven – Installing Apache Maven](https://maven.apache.org/install.html)

> set JAVA\_HOME="C:\Program Files (x86)\Java\jre1.8.0\_281"

> set M2\_HOME="C:\Program Files\apache-maven-3.8.1"

- looking at "Environment Variables" inside "System Properties"

1. Kill a port

**taskkill /pid xxxx /f**

**( sau npx kill-port 8080** – trebuie instalat npm )

**Comenzi MySql Command lines:**

**USERS :**

List all users:  **SELECT User,Host FROM mysql.user;**

Create new user: **CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';**

 **CREATE USER 'radu'@'%' IDENTIFIED BY '12345';**

DROP USER 'radu';

Grant **ALL** access to user for **\*** tables: **GRANT ALL ON database.\* TO 'user'@'localhost';**

**GRANT ALL ON sala.\* TO 'radu'@'%';**

Vezi :

[**MySQL CLI Cheatsheet · GitHub**](https://gist.github.com/hofmannsven/9164408)

**Tabele Aplicatie**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Tabela Membru** |  | * Relatie 1🡪 n cu tabela Abonati |
| **Camp** | **Tip** | **Explicatii** |
| Id | N(8) |  |
| Nume | n(8) |  |
| Prenume | N(5) |  |
| Telefon | C(6) |  |
| dataNastere | D | Data nastere |
| data | D | Data de cand e membru |
| Discount | N(3) |  |
| Observatii | C(100) | Observatii ex. probleme sanatate |

|  |  |  |
| --- | --- | --- |
| **Tabela TipAbonament** |  | * Relatie 1🡪 n cu tabela Abonati |
| **Camp** | **Tip** | **Explicatii** |
| Id | N(8) |  |
| Denumire | C(50) | Denumire abonament |
| Durata | N(3) | Durata abonament in zile |
| NrSedinte | N(3) | \*pun 999 daca e nelimitat |
| DataStart | D | Data de cand e valabil acest tip.ab. |
| DataSfarsit | D | Data sfarsi e valabilitate acest tip.ab. |
| Valoare | N(5) | Valoare default |
| Observatii | C(50) |  |

|  |  |  |
| --- | --- | --- |
| **Tabela Abonati**  **( Istoric Abonati)** |  |  |
| **Camp** | **Tip** | **Explicatii** |
| Id | N(8) |  |
| IdMembru | N(8) |  |
| IdTipAbonament | N(8) |  |
| NumeMembru | C(50) | Nume+Prenume |
| DenumireAbonament | C(50) |  |
| DataStart | D | Data inceput ab. |
| DataSfarsit | D | Data sfarsit ab. |
| Valoare | n(5) |  |
| Observatii | C(100) |  |

\*NumeMembru - desi pare informatie redundanta e utila mai ales la femei care-si schimba numele

\*DenumireAbonament - Informatie redundanta dar BD fiind mica e utila cand mai uit doar in aceasta tabela

**SPRING BOOT**

**Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.**

1. **Dependencies**

spring-boot-starter-tomcat

* The **Apache HTTP Server**  is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source) [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) [web server](https://en.wikipedia.org/wiki/Web_server) software, released under the terms of [Apache License](https://en.wikipedia.org/wiki/Apache_License) 2.0. Apache is developed and maintained by an open community of developers under the auspices of the [Apache Software Foundation](https://en.wikipedia.org/wiki/Apache_Software_Foundation).
* [Apache Tomcat](https://en.wikipedia.org/wiki/Apache_Tomcat) - another web server developed by the Apache Software Foundation

spring-boot-starter-hateoas

**Hypermedia as the Engine of Application State** (**HATEOAS**) is a constraint of the [REST application architecture](https://en.wikipedia.org/wiki/Representational_state_transfer) that distinguishes it from other network application architectures.

With HATEOAS, a client interacts with a network application whose application servers provide information dynamically through [hypermedia](https://en.wikipedia.org/wiki/Hypermedia). A REST client needs little to no prior knowledge about how to interact with an application or server beyond a generic understanding of hypermedia.

tomcat-embed-jasper

Tomcat 9.0 uses the Jasper 2 JSP Engine to implement the [JavaServer Pages 2.3](https://wiki.apache.org/tomcat/Specifications) specification. (.JSP files)

org.projectlombok

The **Project Lombok** (also known as **Lombok**) is a Java library and annotation processor for the Java platform. The library core features can be used by any Java application, but there are some requirements to make it work with Integrated development environments such IntelliJ IDEA or Eclipse IDE.

The main feature of **Lombok** is to automate the generation of Java Beans getters and setters by using annotations.

**spring-boot-starter-thymeleaf**

Thymeleaf is a Java XML/XHTML/HTML5 template engine that can work both in web (servlet-based) and non-web environments. It is better suited for serving XHTML/HTML5 at the view layer of MVC-based web applications, but it can process any XML file even in offline environments. It provides full Spring Framework integration.

In web applications Thymeleaf aims to be a complete substitute for [JavaServer Pages](https://en.wikipedia.org/wiki/JavaServer_Pages" \o "JavaServer Pages) (JSP), and implements the concept of *Natural Templates*: template files that can be directly opened in browsers and that still display correctly as web pages.

Thymeleaf is [open-source software](https://en.wikipedia.org/wiki/Open-source_software), licensed under the [Apache License 2.0](https://en.wikipedia.org/wiki/Apache_License_2.0).

javax.servlet

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

### What Is a Servlet?

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

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

**hibernate-entitymanager**

## How does JPA/Hibernate Work?

Databases are designed with Tables/Relations. Java objects are designed using OOPS. We would want to store the data from objects into tables and vice-versa.

Earlier approaches involved writing SQL Queries. JDBC, Spring JDBC and myBatis were popular approaches.

However, JPA evolved as a result of a different thought process.

***How about mapping the objects directly to tables/relationships?***

This Mapping is also called ORM - Object Relational Mapping. Before JPA, ORM was the term more commonly used to refer to these frameworks. Thats one of the reasons, Hibernate is called a ORM framework.

JPA allows to map application classes to tables in database.

* Entity Manager - Once the mappings are defined, entity manager can manage your entities. Entity Manager handles all interactions with the database
* JPQL (Java Persistence Query Language) - Provides ways to write queries to execute searches against entities. Important thing to understand is the these are different from SQL queries. JPQL queries already understand the mappings that are defined between entities. We can add additional conditions as needed.
* Criteria API defines a Java based API to execute searches against databases.

## JPA vs Hibernate

Hibernate is one of the most popular ORM frameworks.

JPA defines the specification. It is an API.

* How do you define entities?
* How do you map attributes?
* How do you map relationships between entities?
* Who manages the entities?

Hibernate is one of the popular implementations of JPA.

* Hibernate understands the mappings that we add between objects and tables. It ensures that data is stored/retrieved from the database based on the mappings.
* Hibernate also provides additional features on top of JPA. But depending on them would mean a lock in to Hibernate. You cannot move to other JPA implementations like Toplink.

# Spring Data JPA – Query Methods

Query methods are defined in Responsitory interfaces. A repository interface is a java interface directly or indirectly extended from Spring Data org.springframework.data.repository.Repository (note: this is a marker interface). Spring Data provides pre-defined interfaces like  [CrudRepository](https://www.amitph.com/spring-data-jpa-query-methods/#2_CrudRepository)  or [PagingAndSortingRepository](https://www.amitph.com/spring-data-jpa-query-methods/#3_PagingAndSortingRepository) both of them are sub-interfaces of Repository. There are few pre-defined methods in the Repository interfaces. The important thing is the method names have to follow a standard structure and if they do, spring will use it to derive low level sql query at runtime.

public interface CrudRepository<T, ID>

extends Repository<T, ID> {

<S extends T> S save(S var1);

<S extends T> Iterable<S> saveAll(Iterable<S> var1);

Optional<T> findById(ID var1);

boolean existsById(ID var1);

Iterable<T> findAll();

Iterable<T> findAllById(Iterable<ID> var1);

long count();

void deleteById(ID var1);

void delete(T var1);

void deleteAll(Iterable<? extends T> var1);

void deleteAll();

}

@NoRepositoryBean

public **interface** **PagingAndSortingRepository**<**T**, **ID**> **extends** **CrudRepository**<**T**, **ID**> {

Iterable<T> **findAll**(Sort var1);

Page<T> **findAll**(Pageable var1);

}

*HTML*

**Marcaje de bază**

|  |  |
| --- | --- |
| <HTML> </HTML> | Defineşte un fişier în format Web |
| <HEAD> </HEAD> | Antetul documentului |
| <TITLE> </TITLE> | Tilul documentului |
| <BODY> </BODY> | Corpul paginii HTML |
| BGCOLOR = culoare | Culoarea de fond a paginii |
| TEXT=culoare | Culoarea textului pe pagină |
| LINK=culoare | Culoarea legăturiilor nevizitate din pagină |
| VLINK=culoare | Culoarea legăturiilor vizitate din pagină |
| ALINK=culoare | Culoarea legăturiilor pe durata clicului executat de utilizator |
| BACKGROUND = url | Imaginea de fond pentru pagină |
| <P> | Paragraf |
| <Hn> <Hn> | Nivel de subtitlu al documentului (n = 1-6) |
| <FONT> </FONT> | Specifică atribute ale textului încadrat |
| SIZE=n | Dimensiunea textului este 1-7 |
| FACE="a,b" | Specifică fontul: a, dacă este disponibil, sau b |
| COLOR=s | Culoarea textului: fie un nume de culoare, fie o valoare RGB |
| <BR> | Linie nouă |
| <PRE> </PRE> | Informaţie preformatată |
| <!-- --> | Comenatriu HTML |
| <CENTER> </CENTER> | Centrează materialul în pagină |
| <HR> | Riglă orizontală |
| SIZE=x | Înălţimea riglei în pixeli |
| WIDTH=x | Lăţimea riglei în pixeli sau în procente |
| NOSHADE | Dezactivează afişarea umbrei pentru rigla orizontală |
| ALIGN=x | Alinierea riglei orizontale în pagina (left, center, right) |
| COLOR=x | Culoarea riglei orizontale (numai pentru IE) |
| <A> </A> | Marcaj de tip ancoră |
| HREF=url | Referinţă hipertext |
| HREF=#nume | Referinţă către o ancoră internă |
| Name=nume | Definiţia unei ancore interne |
| <sup></sup> | Text exponenţial, superscript. Exp: 23 se scrie 2<sup>3</sup> |
| <sub></sub> | Tagul subscript. Exp: H2O se scrie H<sub>2</sub>O |

**Marcaje pentru liste**

|  |  |
| --- | --- |
| <DD> | Descriere definiţie |
| <DL> </DL> | Lista de tip definiţie |
| <DT> | Termen de definiţie |
| <LI> | Element de listă |
| <OL> | Listă ordonată (numerotată) |
| TYPE=tip | Tipul numerotării. Valori posibile: A, a, I, i, 1 |
| START=x | Numărul de început al listei ordonate |
| <UL> | Listă neordonată (marcată) |
| TYPE=formă | Forma marcajului. Valori posibile: circle, square, disc. |

**Formatarea caracterelor**

|  |  |
| --- | --- |
| <B> </B> | Afişează text cu caractere aldine |
| <I> </I> | Afişează text cu caractere cursive |
| <U> </U> | Afişează text subliniat |
| <TT> </TT> | Text cu font monospaţiu |
| <CITE> </CITE> | Citare bibliografică |
| <CODE> </CODE> | Listing de program |
| <EM> </EM> | Stil logic de evidenţiere |
| <KBD> </KBD> | Text de la tastatură |
| <STRONG> </STRONG> | Evidenţiere logică puternică |
| <VAR> </VAR> | Program sau variabilă |
| <BASEFONT SIZE = n> | Specifică dimensiunea implicită a fontului din pagină |

**Marcaje pentru cadre**

|  |  |
| --- | --- |
| <FRAMESET> </FRAMESET> | Definirea redactării paginii |
| COLS=x | Numărul şi mărimea relativă a coloanelor |
| ROWS=x | Numărul şi mărimea relativă a liniilor |
| BORDER=x | Specifică starea "on" (activă) sau "off" (inactivă) pentru chenarul cadrului FRAMESET (1 sau 0) |
| FRAMEBORDER = x | Specifică mărimea chenarului |
| FRAMESPACING = x | Mărimea spaţiului dintre două cadre adiacente |
| <FRAME> | Definiţia unui anumit cadru |
| SRC=url | URL-ul sursă pentru cadru |
| NAME=nume | Numele cadrului (utilizat împreună cu TARGET=nume ca parte componentă a marcajului de tip ancoră<a> |
| SCROLLING=scrl | Defineşte opţiunea barei de derulare.Valori posibile: on(activă), off(inactivă), auto (automat) |
| FRAMEBORDER=x | Mărimea chenarului din jurul cadrului |
| MARGINHEIGHT=x | Spaţiul suplimentar de deasupra şi dedesubtul unui anumit cadru |
| MARGINWIDTH=x | Spaţiu suplimetar la stânga şi la dreapta unui anumit cadru |
| <NOFRAMES> </NOFRAMES> | Secţiunea de pagină afişată pentru utilizatorii care nu pot vedea un cadru |
| <IFRAME> | Cadru intern (numai pentru IE) |
| SRC=url | Sursa cadrului |
| NAME=s | Numele ferestrei (util pentru TARGET) |
| HEIGHT=x | Înăţtimea cadrului înglobat |
| WIDTH=x | Lăţimea cadrului înglobat |

**Marcaje pentru tabele**

|  |  |
| --- | --- |
| <TABLE> </TABLE> | Tabel HTML |
| BORDER=x | Chenarul tabelului |
| CELLPADDING=x | Spaţiul suplimentar în cadrul celulelor tabelului |
| CELLSPACING=x | Spaţiul suplimentar între celulele tabelului |
| WIDTH=x | Lăţimea impusă tabelului |
| FRAME=valoare | Ajustarea fină a tabelului |
| RULES=valoare | Ajustarea fină a riglelor tabelului |
| BORDERCOLOR = culoare | Specifică culoarea chenarului tabelului |
| BORDERCOLORLIGHT = culoare | Cea mai deschisă culoare din cele două culori specificate |
| BORDERCOLORDARK = culoare | Cea mai închisă culoare din cele două culori specificate |
| ALIGN=left | Aliniază tabelul la marginea din stânga a paginii, iar textul curge în partea dreaptă |
| ALIGN=right | Aliniază tabelul la marginea din dreapta a paginii, iar textul curge în partea stângă |
| HSPACE=x | Spaţiu suplimetar pe orizontală în jurul tabelului |
| VSPACE=x | Spaţiu suplimetar pe verticală în jurul tabelului |
| COLS=x | Specifică numărul de coloane ale unui tabel |
| <COLGROUP> </COLGROUP> | Defineşte un set de definiţii de coloane cu ajutorul marcajului <col> |
| <COL WIDTH=x> | Defineşte lăţimea unei coloane exprimată în pixeli |
| <THEAD> </THEAD> | Defineşte titlul tabelului |
| <BODY> </TBODY> | Defineşte corpul tabelului |
| <TR> </TR> | Linie de tabel |
| BGCOLOR=culoare | Specifică culoarea de fond pentru întreaga linie |
| ALIGN=aliniere | Alinierea celulelor de pe linia curentă (left, center, right) |
| <TD> </TD> | Celula de date a tabelului |
| BGCOLOR=culoare | Specifică culoarea de fond pentru celula de date |
| COLSPAN=x | Numărul de coloane pe care se întinde celula curentă de date |
| ROWSPAN=x | Numărul de linii pe care se întinde celula curentă de date |
| ALIGN=aliniere | Alinierea materialului din cadrul celulei de date.Valori posibile: (left, right, center) |
| VALIGN=aliniere | Alinierea pe verticală a materialului din cadrul celulei de date.Valori posibile: (top, bottom, middle) |
| BACKGROUND=url | Specifică imaginea de fond pentru celula tabelului |
| NOWRAP | Nu permite despărţirea textului pe linii în cadrul unei celule |
| ALIGN=baseline | Aliniază celule de date cu linia de bază a textului adiacent |
| ALIGN=caracter | Aliniază o coloană faţă de un anumit carcater (caracterul prestabilit este ".") |
| ALIGN=justify | Aliniază atât marginea din stânga cât şi marginea din dreapta a unui text |

**Adăugarea imaginilor**

|  |  |
| --- | --- |
| <IMG > | Marcajul de introducere a imaginilor |
| SRC=url | Sursa fişierului grafic |
| ALT=text | Textul alternativ de afişat, dacă este necesar |
| ALIGN=aliniere | Alinierea imaginii în pagină. Valori posibile: top (sus), middle (în mijloc), bottom (jos), left (în stânga), right (la dreapta) |
| HEIGHT=x | Înălţimea imaginii (în pixeli) |
| WIDTH=x | Lăţimea imaginii |
| BORDER=x | Chenarul din jurul imaginii, atunci când aceasta este utilizată ca hiperlegătură |
| HSPACE=x | Spaţiul suplimentar pe orizontală din jurul imaginii (în pixeli) |
| VSPACE=x | Spaţiul suplimentar pe verticală din jurul imaginii (în pixeli) |

*Thymeleaf*

Note that Thymeleaf has integrations for both versions 3.x and 4.x of the Spring Framework, provided by two separate libraries called **thymeleaf-spring3** and **thymeleaf-spring4**. These libraries are packaged in separate **.jar** files (**thymeleaf-spring3-{version}.jar** and **thymeleaf-spring4-{version}.jar**) and need to be added to your classpath in order to use Thymeleaf’s Spring integrations in your application.

Quick summary of the Standard Expression features:

* Simple expressions:
  + Variable Expressions: **${...}**
  + Selection Variable Expressions: **\*{...}**
  + Message Expressions: **#{...}**
  + Link URL Expressions: **@{...}**
* Literals
  + Text literals: **'one text'**, **'Another one!'**,…
  + Number literals: **0**, **34**, **3.0**, **12.3**,…
  + Boolean literals: **true**, **false**
  + Null literal: **null**
  + Literal tokens: **one**, **sometext**, **main**,…
* Text operations:
  + String concatenation: **+**
  + Literal substitutions: **|The name is ${name}|**
* Arithmetic operations:
  + Binary operators: **+**, **-**, **\***, **/**, **%**
  + Minus sign (unary operator): **-**
* Boolean operations:
  + Binary operators: **and**, **or**
  + Boolean negation (unary operator): **!**, **not**
* Comparisons and equality:
  + Comparators: **>**, **<**, **>=**, **<=** (**gt**, **lt**, **ge**, **le**)
  + Equality operators: **==**, **!=** (**eq**, **ne**)
* Conditional operators:
  + If-then: **(if) ? (then)**
  + If-then-else: **(if) ? (then) : (else)**
  + Default: **(value) ?: (defaultvalue)**

All these features can be combined and nested:

**4.1 Messages**

As we already know, **#{...}** message expressions allow us to link this:

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

…to this:

home.welcome=¡Bienvenido a nuestra tienda de comestibles!

# *Standard URL Syntax*

The Thymeleaf standard dialects –called Standard and SpringStandard– offer a way to easily create URLs in your web applications so that they include any required URL preparation artifacts. This is done by means of the so-called link expressions, a type of Thymeleaf Standard Expression: **@{...}**

## *Absolute URLs*

Absolute URLs allow you to create links to other servers. They start by specifying a protocol name (**http://** or **https://**)

<a th:href="@{http://www.thymeleaf/documentation.html}">

They are not modified at all (unless you have an URL Rewriting filter configured at your server and performing modifications at the **HttpServletResponse.encodeUrl(...)** method):

<a href="http://www.thymeleaf/documentation.html">

## *Context-relative URLs*

The most used type of URLs are context-relative ones. These are URLs which are supposed to be relative to the web application root once it is installed on the server. For example, if we deploy a **myapp.war** file into a Tomcat server, our application will probably be accessible as **http://localhost:8080/myapp**, and **myapp** will be the context name.

Context-relative URLs start with **/**:

<a th:href="@{/order/list}">

If our app is installed at **http://localhost:8080/myapp**, this URL will output:

<a href="/myapp/order/list">

## *Server-relative URLs*

Server-relative URLs are very similar to context-relative URLs, except they do not assume you want your URL to be linking to a resource inside your application’s context, and therefore allow you to link to a different context in the same server:

<a th:href="@{~/billing-app/showDetails.htm}">

The current application’s context will be ignored, therefore although our application is deployed at **http://localhost:8080/myapp**, this URL will output:

<a href="/billing-app/showDetails.htm">

## *Protocol-relative URLs*

Protocol-relative URLs are in fact absolute URLs which will keep the protocol (HTTP, HTTPS) being used for displaying the current page. They are typically used for including external resources like styles, scripts, etc.:

<script th:src="@{//scriptserver.example.net/myscript.js}">...</script>

…which will render unmodified (except for URL rewriting), like:

<script src="//scriptserver.example.net/myscript.js">...</script>

## *Adding parameters*

How do we add parameters to the URLs we create with **@{...}** expressions? Simple:

<a th:href="@{/order/details(id=3)}">

Which would output as:

<a href="/order/details?id=3">

You can add several parameters, separating them with commas:

<a th:href="@{/order/details(id=3,action='show\_all')}">

Which would output as:

<!-- Note ampersands (&) should be HTML-escaped in tag attributes... -->

<a href="/order/details?id=3&amp;action=show\_all">

You can also include parameters in the form of path variables similarly to normal parameters but specifying a placeholder inside your URL’s path:

<a th:href="@{/order/{id}/details(id=3,action='show\_all')}">

Which would output as:

<a href="/order/3/details?action=show\_all">

## *URL fragment identifiers*

Fragment identifiers can be included in URLs, both with and without parameters. They will always be included at the URL base, so that:

<a th:href="@{/home#all\_info(action='show')}">

…would output as:

<a href="/home?action=show#all\_info">

## *URL rewriting*

Thymeleaf allows you to configure URL rewriting filters in your application, and it does so by calling the **response.encodeURL(...)** method in the **javax.servlet.http.HttpServletResponse** class of the Servlet API for every URL generated from a Thymeleaf template.

This is the standard way of supporting URL rewriting operations in Java web applications, and allows URLs to:

* Automatically detect whether the user has cookies enabled or not, and add the **;jsessionid=...** fragment to the URL if not —or if it is the first request and cookie configuration is still unknown.
* Automatically apply proxy configuration to URLs when needed.
* Make use (if configured so) of different CDN (Content Delivery Network) setups, in order to link to content distributed among several servers.

A very common (and recommended) technology for URL Rewriting is [URLRewriteFilter](http://tuckey.org/urlrewrite/).

## *Only for th:href’s?*

Do not think URL **@{...}** expressions are only used in **th:href** attributes. They can, in fact, be used anywhere just like variable expressions (**${...}**) or message externalization / internationalization ones (**#{...}**).

For example, you could use them in forms…

<form th:action="@{/order/processOrder}">

…or as a part of other expression. Here as a parameter of an externalized/internationalized string:

<p th:text="#{orders.explanation('3', @{/order/details(id=3,action='show\_all')})}">

## *Using expressions in URLs*

What if we needed to write an URL expression like this:

<a th:href="@{/order/details(id=3,action='show\_all')}">

…but neither **3** nor **'show\_all'** could be literals, because we only know their value at run time?

No problem! Every URL parameter value is in fact an expression, so you can easily substitute your literals with any other expressions, including i18n, conditionals…:

<a th:href="@{/order/details(id=${order.id},action=(${user.admin} ? 'show\_all' : 'show\_public'))}">

What’s more: an URL expression like:

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

…is in fact a shortcut for:

<a th:href="@{'/order/details'(id=${order.id})}">

Which means that the URL base itself can be specified as an expression, for example a variable expression:

<a th:href="@{${detailsURL}(id=${order.id})}">

…or an externalized/internationalized text:

<a th:href="@{#{orders.details.localized\_url}(id=${order.id})}">

…even complex expressions can be used, including conditionals, for example:

<a th:href="@{(${user.admin}? '/admin/home' : ${user.homeUrl})(id=${order.id})}">

Want it cleaner? Use **th:with**:

<a th:with="baseUrl=(${user.admin}? '/admin/home' : ${user.homeUrl})"

th:href="@{${baseUrl}(id=${order.id})}">

…or…

<div th:with="baseUrl=(${user.admin}? '/admin/home' : ${user.homeUrl})">

...

<a th:href="@{${baseUrl}(id=${order.id})}">...</a>

...

</div>

*<!DOCTYPE html>*

*<html lang="en" xmlns:th="*[*http://www.thymeleaf.org*](http://www.thymeleaf.org/)*">*

*<head>*

*<meta charset="ISO-8859-1">*

*<title>Employee</title>*

*<link rel="stylesheet"*

*href="*[*https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css*](https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css)*"*

*integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"*

*crossorigin="anonymous">*

*</head>*

*<body>*

*<div class="container my-2" align="center">*

*<h3>Employee List</h3>*

*<a th:href="@{/addnew}" class="btn btn-primary btn-sm mb-3" >Add Employee</a>*

*<****table*** *style="width:80%" border="1"*

*class = "table table-striped table-responsive-md">* **<!—Tabel HTML 🡪**

*<thead> <!--* Defineşte titlul tabelului 🡪

*<tr>* <!—Linie de tabel 🡪

*<th>Name</th> <!--* Celula header tabel>

*<th>Email</th> <!--* Celula header tabel>

*<th>Action</th> <!--* Celula header tabel>

*</tr>* <!—Linie de tabel 🡪

*</thead> <!--* Defineşte titlul tabelului 🡪

*<tbody>* <!— Corpul Tabelului ->

*<tr th:each="employee:${allemplist}">* <!—Linie de tabel 🡪

*<td th:text="${employee.name}"></td> <!--* Celula de date a tabelului>

*<td th:text="${employee.email}"></td><!--* Celula de date a tabelului>

*<td> <a th:href="@{/showFormForUpdate/{id}(id=${employee.id})}"*

*class="btn btn-primary">Update</a>*

*<a th:href="@{/deleteEmployee/{id}(id=${employee.id})}"*

*class="btn btn-danger">Delete</a>*

*</td>*

*</tr>* <!—Linie de tabel 🡪

*</tbody>* <!— Corpul Tabelului ->

*</****table****>* **<!—Tabel HTML 🡪**

*</div>*

*</body>*

*</html>*

**HTML**

The <th> tag defines a header cell in an HTML table.

An HTML table has two kinds of cells:

* Header cells - contains header information (created with the <th> element)
* Data cells - contains data (created with the [<td>](https://www.w3schools.com/TAGs/tag_td.asp) element)

The text in <th> elements are bold and centered by default.

The text in <td> elements are regular and left-aligned by default.

**THYMELEAF**

**th:insert** is the literal meaning of insert, insert the specified code fragment into the main tag  
**th:replace** is the literal meaning of replacement, replacing the main tag with the specified code snippet

**th:name** => This would be the name of the value that you will be either passing to another page (Exemplar scenario).

**th:value** => This would be the actual value that you would be passing. It could be obtained from a model or straight from the database explicitly.

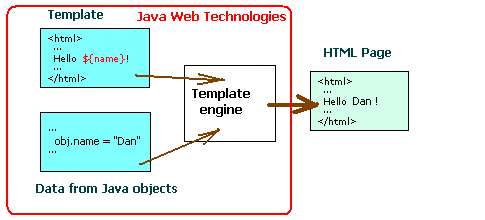
In Thymeleaf : **th:field="${something}"**  vs  **th:field="\*{something}"**

Five types:

* ${...} : Variable expressions. These are OGNL expressions (or Spring EL if you have spring integrated)
* \*{...} : Selection expressions. Same as above, excepted it will be executed on a previously selected object only
* #{...} : Message (i18n) expressions. Used to retrieve locale-specific messages from external sources
* @{...} : Link (URL) expressions. Used to build URLs
* ~{...} : Fragment expressions. Represent fragments of markup and move them around templates

**Thymeleaf vs Angular**

When you create a web page into a server-side Java Web Application, you will follow the schema below:



When you want to create a server-side Java Web Application you can use one of the following solutions [**Java Server Faces (JSF)**](https://learn-it-with-examples.com/development/java/web-tier/java-jsf-example.html), [Java Server Pages (JSP)](https://learn-it-with-examples.com/development/java/web-tier/java-jsp-example.html), [servlets](https://learn-it-with-examples.com/development/java/web-tier/java-servlet-example.html), template engines like **Thymeleaf**, etc.

- As you can see [Thymeleaf](https://learn-it-with-examples.com/development/java/thymeleaf/what-thymeleaf-is.html" \t "_blank) is closer to the HTML format and for this reason Thymeleaf is named "*a natural template*"

 - Angular is a Single Page Application while Thymeleaf is a "multipage" application

 - **With Thymeleaf, the page is created on the server and with Angular, the page is created on the client**

 - Both give you full control to the web design

 - Thymeleaf could be seen as an improvement to the Java Server Pages (JSP) while [Angular](https://learn-it-with-examples.com/development/html-css-javascript/angular/angular-hello-world-example.html) is a (relatively) new way of creating the front-end

## Single-page application vs Multi-page application: What Is Better for Your Project?

MPA or SPA? There is no definite answer which of them is better as everything depends on the web app features you are going to develop and the tasks the app should deal with. Customers should make a choice based on their business needs and technical requirements for a specific project.

If your website has to show a large number of products or services and SEO(**Search engine optimization** ) is an important channel for you to attract customers, a multi-page application is a right choice for you. If you want to provide maximum functionality in limited web space, create a dynamic platform, or flexibly manage large amounts of data, a single-page app is your more suitable option.

Each of both architectures has its own pros and cons. Which one will suit you better? Everything depends on the specific project and specific business requirements. If you need to find the best solution, [contact us](https://lvivity.com/contact), and let us dwell on the details.