***MecBox***

***Developers Guide***

***Versione 1.0 del 08/03/2017***

Sommario

[1. Introduction 4](#_Toc475716813)

[2. What you’ll need 5](#_Toc475716814)

[3. What you’ll build 5](#_Toc475716815)

[4. How to build 5](#_Toc475716816)

[4.1 Maven proxy configuration 6](#_Toc475716817)

[4.2 With Netbeans 8 6](#_Toc475716818)

[4.3 Deploy on Tomcat 8](#_Toc475716819)

[5. How it’s made 9](#_Toc475716820)

[5.1 Server packages 9](#_Toc475716821)

[5.1.1 it.istat.mec.mecbox.domain 10](#_Toc475716822)

[5.1.2 it.istat.mec.mecbox.dao 11](#_Toc475716823)

[5.1.3 it.istat.mec.mecbox.services 11](#_Toc475716824)

[5.1.4 it.istat.mec.mecbox.controller 12](#_Toc475716825)

[5.1.5 it.istat.mec.mecbox.rest 13](#_Toc475716826)

[5.1.6 it.istat.mec.mecbox.forms 14](#_Toc475716827)

[5.1.7 it.istat.mec.mecbox 14](#_Toc475716828)

[5.1.8 it.istat.mec.mecbox.bean 15](#_Toc475716829)

[5.1.9 it.istat.mec.mecbox.security 15](#_Toc475716830)

[5.2 Client packages 16](#_Toc475716831)

[6. How you’ll extend it 18](#_Toc475716832)

[6.1 Use Case: create a function to display the content of a database table 18](#_Toc475716833)

[7. Frameworks 25](#_Toc475716834)

[7.1 Spring 25](#_Toc475716835)

[7.2 JQuery 26](#_Toc475716836)

[7.3 Thymeleaf 27](#_Toc475716837)

[7.4 Boostrap 27](#_Toc475716838)

[7.5 Chart.js 27](#_Toc475716839)

[7.1 Fontawesome 27](#_Toc475716840)

[7.2 Spring Data JPA 28](#_Toc475716841)

# Introduction

Il presente documento ha lo scopo di dettagliare i componenti della MecBox e guidare l’utente alla sua customizzazione.

Mecbox is an open-source application generator used to develop quickly modern web applications using the Spring Framework.

The goal of Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

# What you’ll need

In order to build the mecbox application, your environment should fulfill the following requirements:

* A favorite text editor or IDE
* [JDK 1.8](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Maven 3.0+](http://maven.apache.org/download.cgi)
* Mysql Server

# What you’ll build

You’ll build a template web application that will provide out of the box :

* Authentication & authorization;
* Responsive graphical interface (html, css, js):
  + Tables with enhanced interaction controls (search, export, sorting, etc.);
  + Charts;
* Server side components:
  + CRUD (insert, delete, update);
  + Search filters;

# How to build

[Download](https://github.com/spring-guides/gs-spring-boot/archive/master.zip) the source code from Github at the following url:

<https://github.com/mecdcme/mecbox>

Unzip the source code in your workspace MECBOX\_PATH.

Before building the application you must create a MySQL database. From the command line go to MySQL installation directory MYSQL\_PATH:

cd MYSQL\_PATH\bin;

mysql -u db\_user -p

Then create the database *mecbox* with all tables needed, using the script *mecbox.sql* stored in the MECBOX\_PATH/sql folder:

mysql> source mecbox.sql

If you prefer, you can open and run the script with your favorite client (ex. workbench).

The script will populate the USER/ROLES tables with two users:

Username: [admin@mecbox.it](mailto:admin@mecbox.it)

Password: mecbox

Role: ADMIN

Username: [guest@mecbox.it](mailto:guest@mecbox.it)

Password: mecbox

Role: GUEST

# Maven proxy configuration

In order to download Maven dependencies it is necessary to modify the settings file located in NETBEANS\_PATH/ *java/maven/conf.* Add the following entry within the <proxies> tag:

<proxy>

<host>proxy.istat.it</host>

<port>3128</port>

</proxy>

If you have installed Maven in your environment modify the settings file located in *Users/USER\_NAME/.m2*

# With Netbeans 8

From the main menu select File/Open Project. Then select the folder containing the unzipped maven project and click the button ‘open project’.

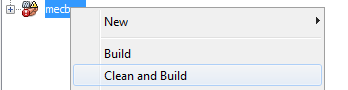
As a first step check the content of the *application.properties* file, located in the path *Other Sources > src/main/resources*:

spring.datasource.url = jdbc:mysql://localhost:3306/mecbox?useSSL=false

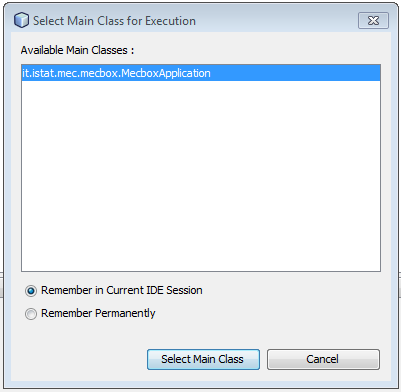
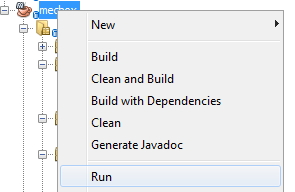
spring.datasource.username = db\_user

spring.datasource.password = db\_password

Now you are ready to perform your first build of the application:



If the build process ended successfully, you are ready to run the application. The application is built using the open source framework **Spring Boot**, which generates an executable jar (that can be run from the command line). Indeed Spring Boot creates a stand-alone Spring based Applications, with an embedded Tomcat, that you can "just run".



As shown above, first of all select the run option, then the IDE will display a modal window and you will have to select the main class (*it.istat.mec.mecbox.MecBoxApplication*). Now you can access the url:

http://localhost:8080/

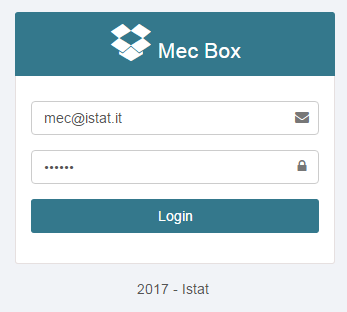
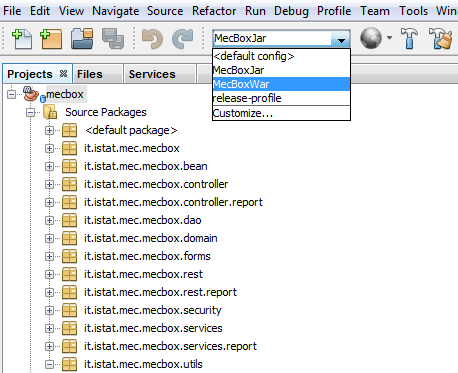


Figure - Login page

# Deploy on Tomcat

The project file pom.xml has been designed to also build the application as a war so you can deploy it on Tomcat.

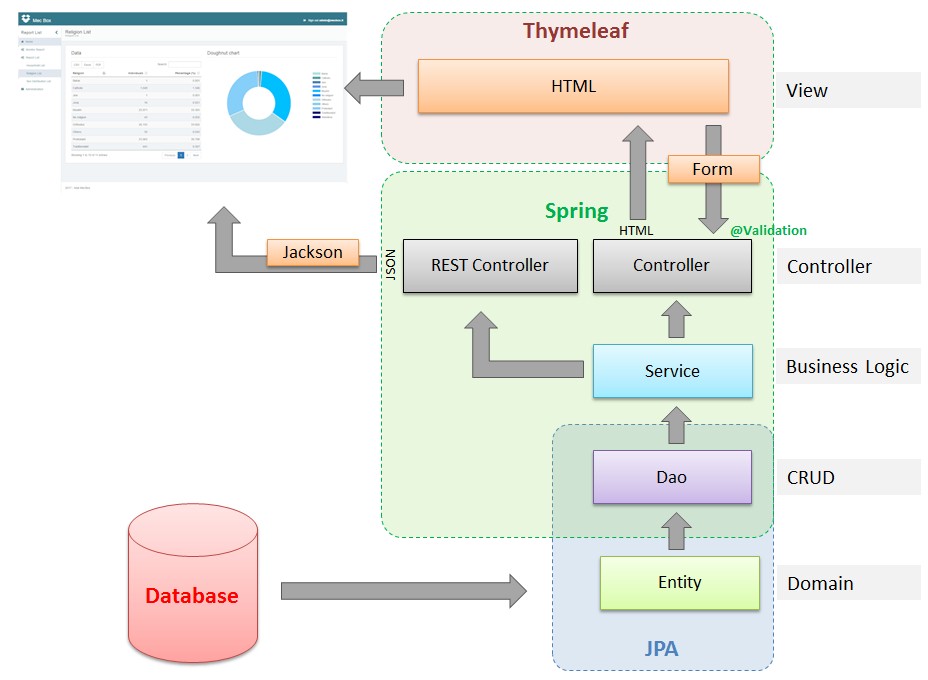
Select the Project configuration *MecBoxWar* and run Build.



Now you find in target folder *mecbox.war* .

# How it’s made

L’applicazione è costituita da diversi componenti che sfruttano le potenzialità dei vari framework utilizzati.



Mecbox is made of several frameworks, which are listed below:

**Server side framework:**

Web site engine:

* Spring MVC (4.3.4)
* Spring Boot (1.4.2)
* Thymeleaf (2.1.5)

Security framework:

* Spring Security (4.1.3)

Persistence:

* Spring Data JPA (1.10.5)
* Hibernate JPA (2.1)

**Client side framework:**

* Bootstrap (3.3.7)
* Fontawesome (4.7.0)
* Datatable (1.10.13)
* Chart JS (2.2.1)
* JQuery (3.1.1)

# Server packages

In the following section are described the JAVA packages of the application. All the packages of the Server Side component have as root **it.istat.mec.mecbox**.

# it.istat.mec.mecbox.domain

This package contains the domain classes (JPA Entities).

In order to understand the meaning of JPA annotations we consider, as an example, the annotations used in the domain classes *User* e *UserRole*, mapping the tables MB\_USERS e MB\_USER\_ROLES:

*@Entity:* this annotation marks a JAVA class as an entity bean of the domain model

*@Table:* allows you to specify the details of the table that will be used to persist the entity in the database

*@JsonManagedReference*, *@JsonBackReference*: these annotations are from Jackson library that is used as a matching data-binding library (POJOs to and from JSON). Such annotations should be used to handle parent/child relationships and avoid in REST services recursive responses.

**More info:**

[Full Listing of Jackson Annotations](https://github.com/FasterXML/jackson-annotations/wiki/Jackson-Annotations) details all available annotations:

<https://github.com/FasterXML/jackson-annotations/wiki/Jackson-Annotations>

Home page of the Jackson Project:

<https://github.com/FasterXML/jackson>

Java Persistence is the API for the management for persistence and object/relational mapping:

<https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html>

# it.istat.mec.mecbox.dao

This package contains the classes that manage the CRUD (Create, Read, Update, Delete) functionalities. For each Entity within the domain package a DAO class, containing the methods to access the database, is defined.

The DAO classes are marked with the annotation *@Repository*.

Further the DAO classes extend the *CrudRepository*. Such interface provides sophisticated CRUD functionality for the entity class that is being managed.

Spring Data provides a JPA module that supports:

* Queries derived from method name using JPA naming convention

(ex. UserDao.findByEmail());

* Queries manually defined as a String using JPA @Query  annotation

(ex. UserRolesDao.findRoleByEmail()).

**More info:**

JPA Repositories reference:

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.repositories>

# it.istat.mec.mecbox.services

This package contains the classes that implement the Business logic. Each class is marked with the annotation *@Service*.

This annotation serves as a specialization of [*@Component*](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Component.html), allowing for implementation classes to be autodetected through classpath scanning.

The DAO objects are injected in a Service class using the annotation *@Autowired.*

*@Autowired*: Marks a constructor, field, setter method or config method as to be autowired by **Spring's dependency injection** facilities. By default, the @Autowired will perform the dependency checking to make sure the property has been wired properly. When Spring can’t find a matching bean to wire, it will throw an exception.

# it.istat.mec.mecbox.controller

Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view. Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

In questo package troviamo i Form Controller. Normalmente troviamo un Controller per ogni Form presente nell’applicazione.

Prendiamo come esempio la classe *UserController* che gestisce la registrazione e la login di un utente.

The *@Controller* annotation indicates that a particular class serves the role of a *controller*. Spring does not require you to extend any controller base class or reference the Servlet API. However, you can still reference Servlet-specific features if you need to.

The *@Controller* annotation acts as a stereotype for the annotated class, indicating its role. The dispatcher scans such annotated classes for mapped methods and detects *@RequestMapping* annotations.

You use the *@RequestMapping* annotation to map URLs such as */users/register* onto an entire class or a particular handler method. Typically the class-level annotation maps a specific request path (or path pattern) onto a form controller, with additional method-level annotations narrowing the primary mapping for a specific HTTP method request method ("GET", "POST", etc.) or an HTTP request parameter condition.

An *@ModelAttribute* on a method argument indicates the argument should be retrieved from the model. If not present in the model, the argument should be instantiated first and then added to the model. Once present in the model, the argument’s fields should be populated from all request parameters that have matching names. This is known as data binding in Spring MVC, a very useful mechanism that saves you from having to parse each form field individually.

As a result of data binding there may be errors such as missing required fields or type conversion errors. To check for such errors add a *BindingResult* argument immediately following the *@ModelAttribute* argument. With a *BindingResult* you can check if errors were found in which case it’s common to render the same form where the errors can be shown with the help of Spring’s *<errors>* form tag.

In addition to data binding you can have a validation invoked automatically by adding the JSR-303 *@Valid* annotation.

Spring Validation, Data Binding, and Type Conversion:

<http://docs.spring.io/spring/docs/current/spring-framework-reference/html/validation.html>

# it.istat.mec.mecbox.rest

In questo package vengono inseriti I Controller di tipo Rest.

In Spring’s approach to building RESTful web services, HTTP requests are handled by a controller identified by the [*@RestController*](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html) annotation.

For example the *UserRestController* handles requests (all HTTP methods) for */users/restgetUser* by returning a new instance of the User class with a specific id. Use *@RequestMapping(method=GET)* to narrow this mapping.

The *@RequestMapping* annotation ensures that HTTP requests to */users/restgetUser* are mapped to the *getUser()* method. *@RequestParam* binds the value of the query string parameter name into the name parameter of the *getUser()* method.

A key difference between a traditional MVC controller and the RESTful web service controller above is the way that the HTTP response body is created. Rather than relying on a [view technology](https://spring.io/understanding/view-templates) to perform server-side rendering of the greeting data to HTML, this RESTful web service controller simply populates and returns a *User* object. The object data will be written directly to the HTTP response as JSON.

This code uses Spring 4’s new [*@RestController*](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html) annotation, which marks the class as a controller where every method returns a domain object instead of a view. It’s shorthand for *@Controller* and *@ResponseBody* rolled together.

The Greeting object must be converted to JSON. Thanks to Spring’s HTTP message converter support, you don’t need to do this conversion manually. Because [Jackson 2](http://wiki.fasterxml.com/JacksonHome) is on the classpath, Spring’s [*MappingJackson2HttpMessageConverter*](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/http/converter/json/MappingJackson2HttpMessageConverter.html) is automatically chosen to convert the *User* instance to JSON.

# it.istat.mec.mecbox.forms

In questo package troviamo tutti I bean che mappano lato server i client form. In questa fase si può implementare la validazione sfruttando JavaBeans Validation:

<http://docs.oracle.com/javaee/6/tutorial/doc/gircz.html>

In addition to the constraints defined by the Bean Validation API Hibernate Validator provides several useful custom constraints (ex. @Email):

<http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#validator-defineconstraints-hv-constraints>

Home page of the Hibernate validator:

<http://hibernate.org/validator/>

# it.istat.mec.mecbox

Nel package principale troviamo la classe di configurazione di Spring Security *WebSecurityConfig.*

The *WebSecurityConfig* class is annotated with @EnableWebSecurity to enable Spring Security’s web security support and provide the Spring MVC integration. It also extends *WebSecurityConfigurerAdapter* and overrides a couple of its methods to set some specifics of the web security configuration.

The *configAuthentication()* method configure spring security to use custom *UserDetailsService* (you find this implementation in services package).

The *configure(HttpSecurity)* method defines which URL paths should be secured and which should not. Specifically, the "/", "/index",“/users/login” and “/users/logout” paths are configured to not require any authentication. All other paths must be authenticated.

When a user successfully logs in, they will be redirected to the previously requested page that required authentication. There is a custom *"/login*" page specified by loginPage(), and everyone is allowed to view it.

The *passwordencoder()* method provides Service interface for encoding passwords. The preferred implementation is BCryptPasswordEncoder.

Set up Spring Security:

<https://spring.io/guides/gs/securing-web/#initial>

Un’altra componente del package root è la classe java standalone *MecBoxApplication* che sfruttando il framework Spring Boot, permette l’esecuzione dell’applicazione senza deploy su Application Server esterni:

*java –jar mecbox-1.0.jar*

# it.istat.mec.mecbox.bean

In questo package vengono inseriti eventuali bean specifici legati alle esigenze di una particolare applicazione.

# it.istat.mec.mecbox.security

In questo package troviamo la classe che rappresenta l’utente all’interno dello Spring Security framework.

In Spring Securirty a User/Principal is an instance of the *UserDetails* interface.

public interface UserDetails extends Serializable {

      GrantedAuthority[] getAuthorities();

      String getPassword();

      String getUsername();

      boolean isAccountNonExpired();

      boolean isAccountNonLocked();

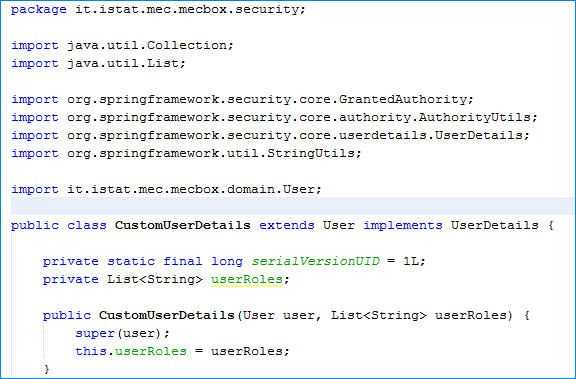
      boolean isCredentialsNonExpired();

      boolean isEnabled();

}

*CustomerUserDetails* class implements *UserDetails* interface to hold on to the Customer object.

The implementation is very simple in terms of the overriding methods, *CustomerUserDetails* just return instance variable values, just like normal getters and setters. The interesting part is in the constructor that receives a list of Strings for the roles that the User/Customer has.



To use your custom dao class, you have to create a bean which implements *UserDetailsService* interface (you find this implementation in services package) and override the *loadUserByUserName()* method of this interface.



# Client packages

* **public.css**

In questo package troviamo gli stylesheet relativi a:

* + mecbox
  + bootstrap
  + fotawesome
  + datatable
  + metisMenu ( jQuery menu plugin)

* **public.js**

In questo package troviamo tutti i file javascript relativi a:

* + mecbox
  + bootstrap
  + jQuery
  + Datatable
  + Chart.js
  + metisMenu (jQuery menu plugin)
* **public.fonts**

In questo package troviamo I componenti del framework Fontawesome

* **public.templates**

In questo package troviamo tutti I moduli html

# How you’ll extend it

# Use Case: create a function to display the content of a database table

In Figure 1 is reported the result of the use case described within this section. You will learn how to visualize the data stored in a database table (ex. *Religion*). Two approaches will be described:

* **Asynchronous**: the content of the chart is loaded via an AJAX call. A javascript function performs a GET request to a rest API.
* **Synchronous**: the entire HTML of the table is provided by the server

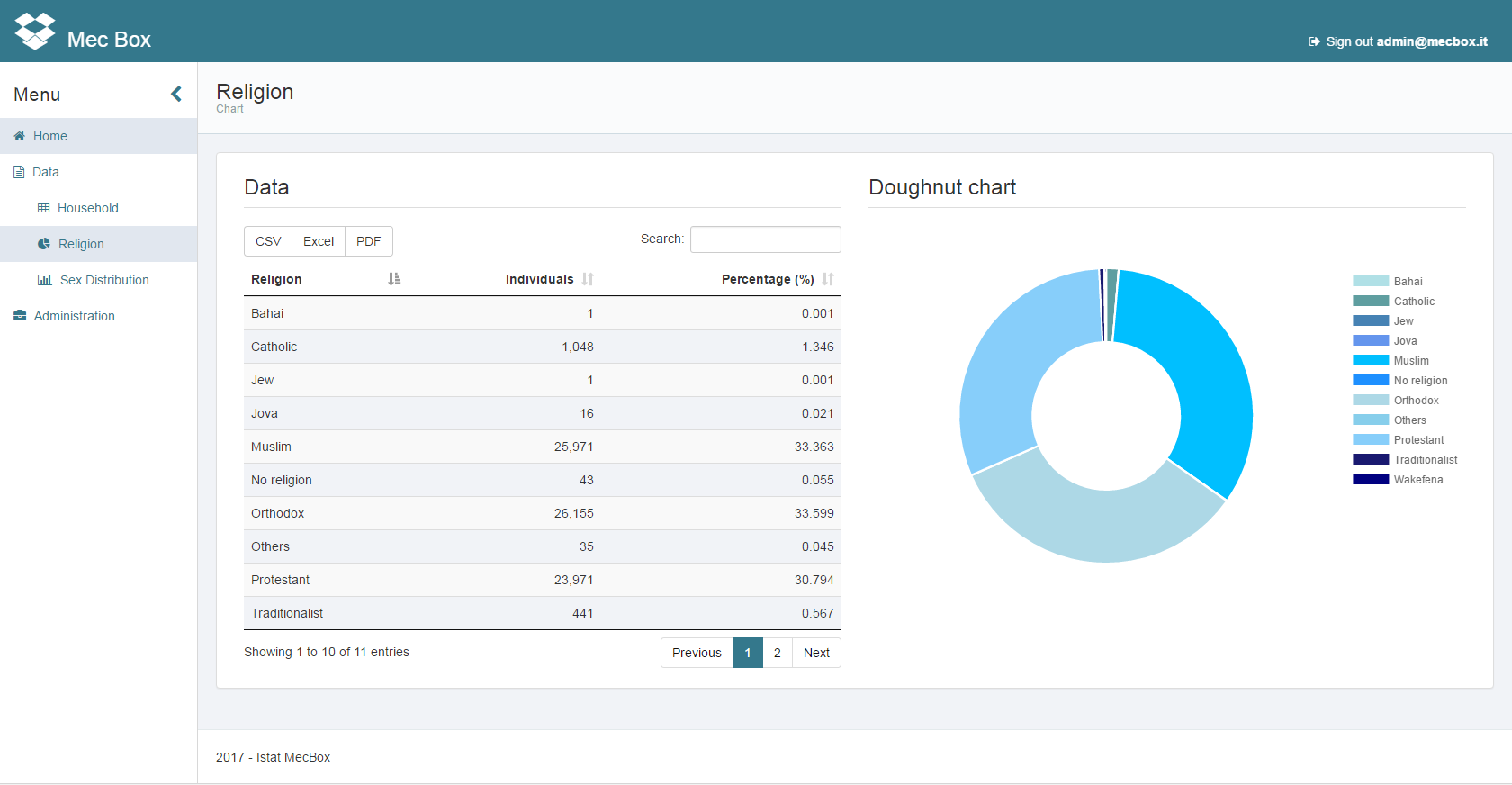
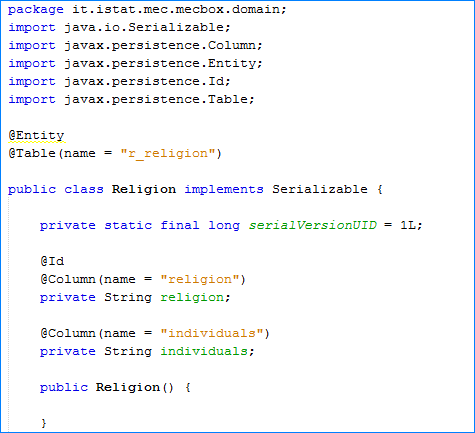


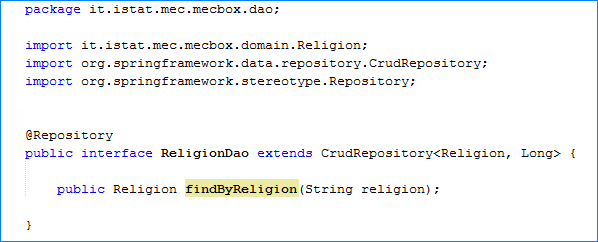
Figure - Display data in a table and a chart

**Server side implementation**

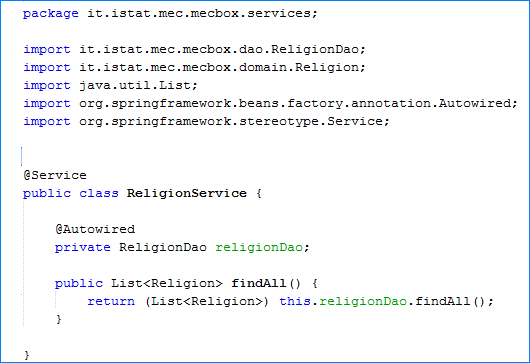
**Domain**: create a Domain class to model a database table (ex. *Religion.java*)



**Data Access Object (DAO)**: create a new DAO class that extends CrudRepository (ex. *ReligionDao.java*). This repository provides out-of-the-box the main CRUD functionalities such as *findAll()*, *findById()*, etc. If you need to implement a functionality that is not exposed by the CrudRepository, such as *findByReligion()*, you must specify the method in the DAO class interface. A detailed description of JPA repositories is provided in section [5.1.2](#_it.istat.mec.mecbox.dao) .



**Service:** create a new Service class to access the methods exposed by the DAO classes (ex. *ReligionService.java*). The DAO class can be injected using the annotation *@Autowired* (which is described in section [5.1.3](#_it.istat.mec.mecbox.services)). If you need to implement business logic you are in the right place!



**Controller:** create a new Controller class (ex. *ReligionController.java*) to map one or more urls (ex. */religion/religionlist*). In the example below, the method *religionList* is invoked when the server receives a */religion/religionlist* GET request. In order to retrieve the data from the database, the method *findAll()* of the service *ReligionService* is invoked. The Service class is injected using the annotation *@Autowired*. To access the list of Religion objects in the view it is necessary to add an attribute to the model object.



Cambiare il nome della pagina html restituita

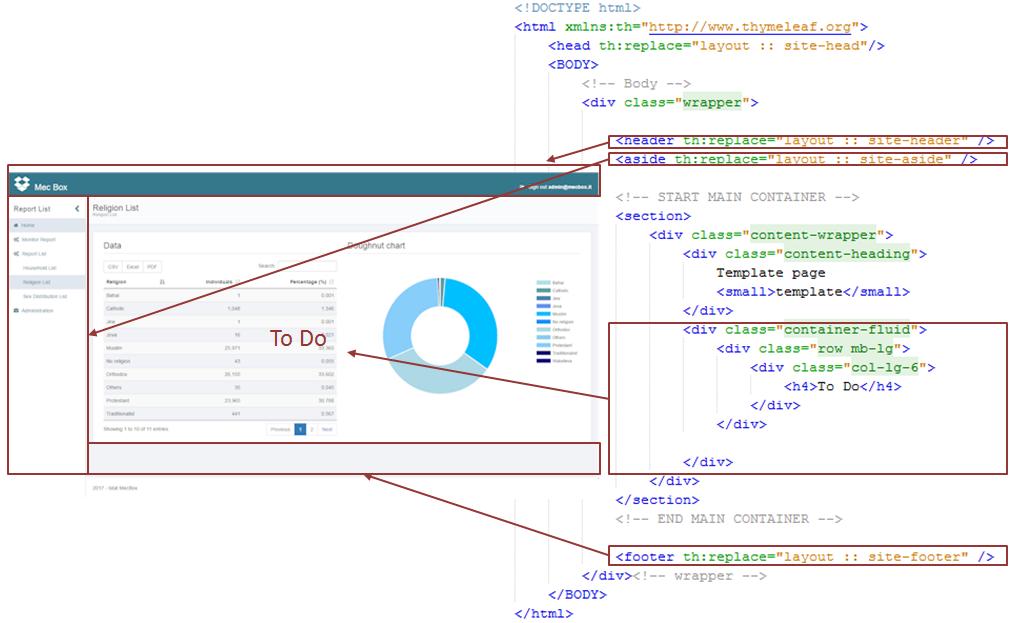
**Rest Controller**: create a new RestController class (ex. *ReligionRestController.java*) to expose a rest API (ex. */religion/restlist*). In the example, the method *religionList* is invoked when the server receives a */religion/restlist* GET request. The database data is retrieved invoking the method *findAll()* of the service *ReligionService*. The returned list of *Region* objects is automatically transformed in JSON format by the Jackson library.



**Client side implementation**

**HTML**:

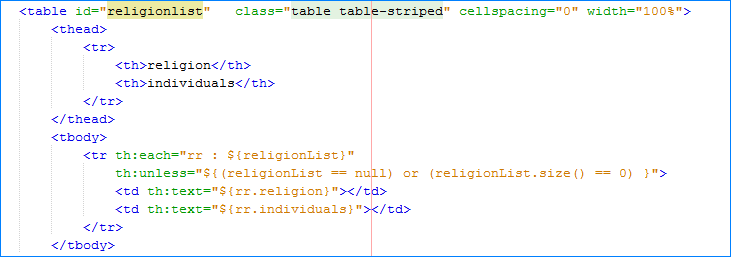
Creare una nuova pagina html partendo da template.html nel package templates (ex *templates.religion.ReligionList.html*):



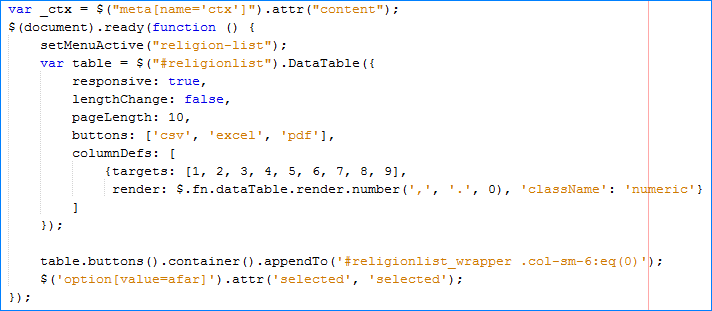
Il template fornisce già l’implementazione dell’header del footer e della barra laterale. La personalizzazione consiste nel modificare la parte contenuta nel *content-wrapper* costituita dal *content-heading* (il titolo) ed il *content-fluid* (il body).

**Table**

Per visualizzare il risultato della tabella d’esempio sfruttando Thymeleaf  è sufficiente iterare l’attribute del Model che è stato impostato nel controller:



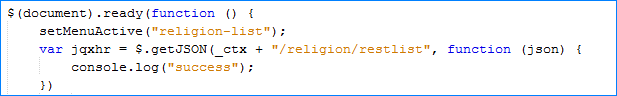
Per visualizzare la tabella tramite *Datatable* implementare una funzione javascript che imposta visualizzazione pulsanti e paginazione:



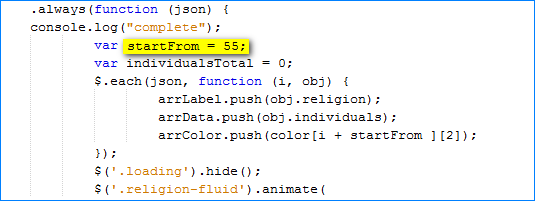
Chiama nuova funzione DisplayDataTable(tableDiv)

**Chart**

Per implementare un grafico (ex D*oughnutChart*) con *Chart.js* viene evocato con una chiamata ajax il rest controller specifico (in questo caso *religion/restlist* ) che restituisce direttamente i dati in formato json.



A questo punto è necessario leggere il json per valorizzare gli array necessari al grafico: dati, label e colori.



Per gestire il range dei colori è necessario impostare la variabile *startForm* che indica il punto di partenza nell’array della scala dei colori presente nel file *mecBoxChart.js* .

Ora il grafico può essere creato:



# Frameworks

Se ‘ventamo qlc

# Spring

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform. A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

**Features**

* Dependency Injection
* Aspect-Oriented Programming including Spring's declarative transaction management
* Spring MVC web application and RESTful web service framework
* Foundational support for JDBC, JPA, JMS

Reference:

<https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html>

* + 1. **Spring Security**

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications.

**Features**

* Comprehensive and extensible support for both Authentication and Authorization
* Protection against attacks like session fixation, clickjacking, cross site request forgery, etc
* Servlet API integration
* Optional integration with Spring Web MVC

Reference:

<http://docs.spring.io/spring-security/site/docs/4.2.2.BUILD-SNAPSHOT/reference/htmlsingle/>

* + 1. **Spring Boot**

Spring Boot allow to create stand-alone, production-grade Spring based Applications that you can "just run".

**Features**

* Create stand-alone Spring applications
* Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
* Provide opinionated 'starter' POMs to simplify your Maven configuration
* Automatically configure Spring whenever possible
* Provide production-ready features such as metrics, health checks and externalized configuration
* Absolutely **no code generation** and **no requirement for XML** configuration

Reference:

http://docs.spring.io/spring-boot/docs/2.0.0.BUILD-SNAPSHOT/reference/htmlsingle/

# JQuery

jQuery is a lightweight JavaScript library.

It takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

* HTML/DOM manipulation
* CSS manipulation
* HTML event methods
* Effects and animations
* AJAX
* Utilities

Api Documentation:

<http://api.jquery.com/>

* + 1. **DataTables**

DataTables is a plug-in for the [jQuery](https://jquery.com/) Javascript library. It is a highly flexible tool, based upon the foundations of progressive enhancement, and will add advanced interaction controls to any HTML table.

DatatTables Manual:

<https://datatables.net/manual/index>

# Thymeleaf

Thymeleaf is a server-side Java template engine for both web and standalone environments.

[Thymeleaf](http://www.thymeleaf.org/) defines itself as an  XML / XHTML / HTML5 template engine.

It is not based on JSPs but rather on some plain HTML files with a little bit of namespace magic.

Thymeleaf Documentation:

<http://www.thymeleaf.org/documentation.html>

# Boostrap

HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. Over Bootstrap has dozen reusable components built to provide iconography, dropdowns, input groups, navigation, alerts, and much more.

Bootstrap Getting Started:

<http://getbootstrap.com/getting-started/>

# Chart.js

JS chart library.

Documentation:

<http://www.chartjs.org/docs/>

# Fontawesome

Font Awesome gives you a set of scalable vector icons that can instantly be customized — size, color, drop shadow, and anything that can be done with CSS.

Get starter:

<http://fontawesome.io/get-started/>

# Spring Data JPA

Spring Data JPA (**Java Persistence API** ), part of the larger [Spring Data](http://projects.spring.io/spring-data) family, makes it easy to implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It allow to build Spring-powered applications that use data access technologies.

<http://projects.spring.io/spring-data-jpa/>

Accessing data with JPA

https://spring.io/guides/gs/accessing-data-jpa/