***MecBox***

***Developers Guide***

***Versione 1.0 del 15/12/2016***

Sommario

[1. Introduction 3](#_Toc472667375)

[2. What you’ll need 3](#_Toc472667376)

[3. What you’ll build 3](#_Toc472667377)

[4. How to build 3](#_Toc472667378)

[4.1 With Netbeans 3](#_Toc472667379)

[5. How it’s made 5](#_Toc472667380)

[5.1 Web Site & Rest Service 5](#_Toc472667381)

[6. How you’ll extend it 8](#_Toc472667382)

[7. Framework 8](#_Toc472667383)

[7.1 Spring 8](#_Toc472667384)

[7.2 JQuery 9](#_Toc472667385)

[7.3 Thymeleaf 10](#_Toc472667386)

[7.4 Boostrap 10](#_Toc472667387)

[7.1 Fontawesome 10](#_Toc472667388)

[7.2 Spring Data JPA 10](#_Toc472667389)

# Introduction

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

# 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 root -p

mysql> create database mecbox;

Then create the USER/ROLES tables, using the script *create\_user\_roles.sql* stored in the MECBOX\_PATH/sql folder:

mysql> use mecbox;

mysql> source create\_user\_roles.sql

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

Username: [mec@istat.it](mailto:mec@istat.it)

Password: mecbox

Role: ADMIN

Username: [guest@istat.it](mailto:guest@istat.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’.

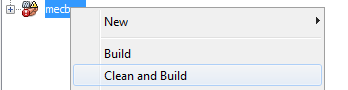
As a first step check the content of the *application.properties* file, located in the path *Other Sources > src/main/resources*:

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

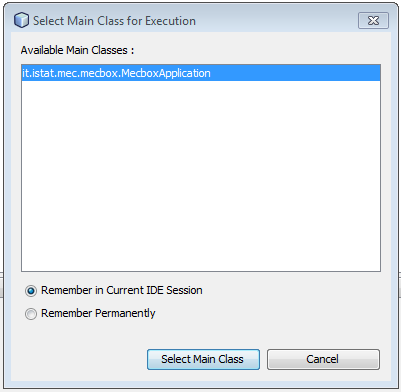
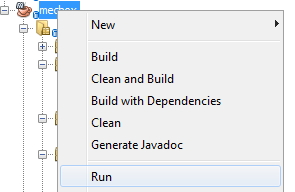
spring.datasource.username = root

spring.datasource.password = root

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/

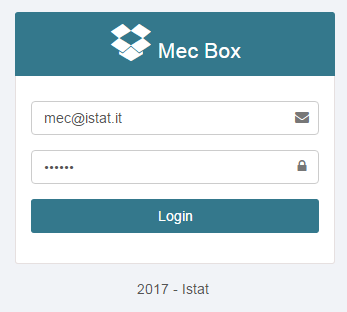


Figura 1 - Login page

# Deploy on Tomcat

If you nedd to deploy mecbox as a war follow these steps:

1. Modify project file *pom.xml*
   1. change entry packaging:

*<packaging>war</packaging>*

* 1. add the dependency:

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-tomcat</artifactId>*

*<scope>provided</scope>*

*</dependency>*

1. Modify main class *MecboxApplication* to extend *SpringBootServletInitializer* (add import *org.springframework.boot.web.support.SpringBootServletInitializer*)

# How it’s made

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

In particolare queste le componenti principali e i relativi framework utilizzati:

Web Site & Rest Service:

* Spring MVC
* Spring Boot

Security:

* Spring Security

Interfaccia:

* Bootstrap
* Thymeleaf,
* Datatable,
* Fontawesome

Persistence

* Spring Data JPA
* Hibernate

# Web Site & Rest Service

Di seguito vengono descritti i package della mecbox, in ordine di apparizione, con le loro principali caratteristiche. Tutti i package della componente Server Side hanno come radice **it.istat.mec.mecbox**.

* **it.istat.mec.mecbox.domain**

In questo package troviamo tutti gli JPA Entity classes.

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>

JPA Entity classes are user defined classes whose instances can be stored in a database.  
To store data in an ObjectDB database using JPA you have to define entity classes that represent your application data object model.

Prendiamo come esempio gli entity bean *User* e *UserRole* che rappresentano rispettivamente la tabella *USERS* e *USER\_ROLES.*

Per marcare un Bean come un oggetto del domain model (entity bean) si fa uso dell’annotazione *@Entity.*

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

The *@Table* annotation provides four optional attributes, allowing you to override the name of the table, its catalogue, and its schema, and enforce unique constraints on columns in the table.

The *@JsonManagedReference*, *@JsonBackReference* annotations allow you to indicate and handle parent/child relationships expressed with pair of matching properties. These annotation are from Jackson library, a suite of data-processing tools for Java (and the JVM platform), including the flagship streaming [JSON](https://en.wikipedia.org/wiki/JSON) parser / generator library, matching data-binding library (POJOs to and from JSON).

[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>

* **it.istat.mec.mecbox.dao**

In questo package troviamo tutte le classi che si occupano del CRUD (Create, Read, Update, Delete) dell’applicazione. Per ogni Entity presente nel package *domain* troviamo un Dao (Data Access Object) con i metodi per accedera al database.

The central interface in Spring Data repository abstraction is *Repository*. It takes the the domain class to manage as well as the id type of the domain class as type arguments. This interface acts primarily as a marker interface to capture the types to work with and to help you to discover interfaces that extend this one. The *CrudRepository* provides sophisticated CRUD functionality for the entity class that is being managed.

The JPA module supports defining a query manually as String or have it being derived from the method name.

Although getting a query derived from the method name is quite convenient, one might face the situation in which either the method name parser does not support the keyword one wants to use or the method name would get unnecessarily ugly. So you can either use JPA named queries through a naming convention or rather annotate your query method with @Query .

JPA Repositories reference:

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

* **it.istat.mec.mecbox.services**

In questo package troviamo le classi che implementano la logica di business.

Vengono identificate tramite l’utilizzo dell’annotazione *@Service* .

This annotation Indicate that a class is a "Business Service Facade" (in the Core J2EE patterns sense), or something similar. This annotation is a general-purpose stereotype and individual teams may narrow their semantics and use as appropriate.

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.

* **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-0.0.1-SNAPSHOT.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.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

n 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.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 baean which implements *UserDetailsService* interface (you find this implementation in services package) and override the *loadUserByUserName()* method of this interface.

# How you’ll extend it

# Framework

Se ‘ventamo qlc

Client Side

<https://spring.io/understanding/view-templates>

Persistence

# 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**
    2. **Spring Mvc**
    3. **Spring Boot**

# 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
  + 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.

# 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.

# 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.

# Fontawesome

# 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 easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier 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/