

***MecBox***  
***Developers Guide 1.0***

***Versione 1.0 - 2016***

## Summary

1.	Introduction .....	4
2.	What you'll need .....	5
3.	What you'll build .....	5
4.	How to build .....	5
4.1	Maven proxy configuration .....	6
4.1.1	Configure Maven in Eclipse (neon.2) .....	6
4.2	Oracle Datasource configuration .....	8
4.3	With Netbeans 8 .....	10
4.4	With Eclipse (Neon.2) .....	12
4.5	Application server deployment .....	14
5.	How it's made .....	15
5.1	Server packages .....	16
5.1.1	it.istat.mec.mecbox.domain .....	16
5.1.2	it.istat.mec.mecbox.dao .....	17
5.1.3	it.istat.mec.mecbox.services .....	17
5.1.4	it.istat.mec.mecbox.controller .....	18
5.1.5	it.istat.mec.mecbox.rest .....	19
5.1.6	it.istat.mec.mecbox.forms .....	20
5.1.7	it.istat.mec.mecbox .....	20
5.1.8	it.istat.mec.mecbox.bean .....	21
5.1.9	it.istat.mec.mecbox.security .....	21
5.2	Internationalization .....	24
5.3	Client packages .....	26
6.	How you'll extend it .....	28
6.1	Use Case: create a function to display the content of a database table .....	28
6.1.1	Server side implementation .....	29
6.1.2	Client side implementation .....	32
7.	Frameworks .....	36
7.1	Spring .....	36
7.2	jQuery .....	37
7.3	Thymeleaf .....	38

7.4	Bootstrap.....	38
7.5	Chart.js .....	38
7.6	Fontawesome.....	38
7.7	Spring Data JPA .....	39
7.8	LESSCSS.....	40
7.8.1	Variables.....	40
7.8.2	Mixin.....	40
7.8.3	Mixin and Funtions.....	41
7.8.4	Conclusion .....	42

## 1. Introduction

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.

## 2. 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 or later
- Maven 3.5+
- Mysql Server 5.7 or Oracle 12c

## 3. 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);
  - RESTful API
  - Search filters;

## 4. How to build

Download 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 by the web application, using the script *mecbox.sql* stored in the MECBOX\_PATH/sql folder:

```
mysql> source mecbox-mysql.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
Password: mecbox
Role: ADMIN
```

```
Username: guest@mecbox.it
Password: mecbox
Role: GUEST
```

If you use Oracle run the script *mecbox-oracle.sql*. The script requires that the database exists. To find out how to connect MecBox to an Oracle database see section 4.2 .

## 4.1 Maven proxy configuration

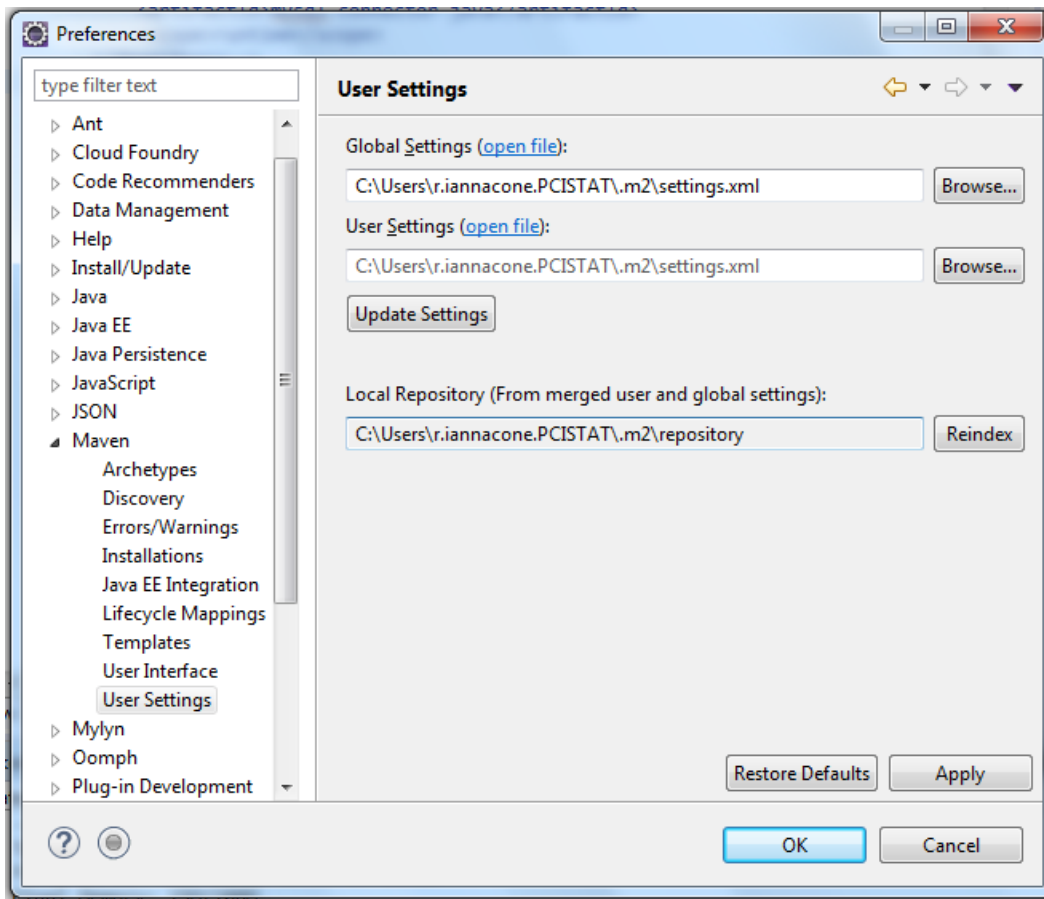
In order to download Maven dependencies it is necessary to modify the settings file located in NETBEANS\_PATH/ *java/maven/conf*. Add the following entry within the <proxies> tag:

```
<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*

### 4.1.1 Configure Maven in Eclipse (neon.2)

To configure Maven go to Window -> Preferences, a dialog box will show. Select in the menu Maven -> User settings, like in the image below, and specify a path where to find the file *settings.xml* in the field Global Settings.



If the file doesn't exist create it and add the rows below to set proxy values:

```
<settings xmlns=http://maven.apache.org/SETTINGS/1.0.0
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
</proxies>
  <proxy>
    <host>proxy.istat.it</host>
    <port>3128</port>
  </proxy>
</proxies>
</settings>
```

## 4.2 Oracle Datasource configuration

To connect MecBox to an Oracle database you have to follow these steps.

1. **Edit the project file *pom.xml*:** uncomment the dependency *com.oracle*

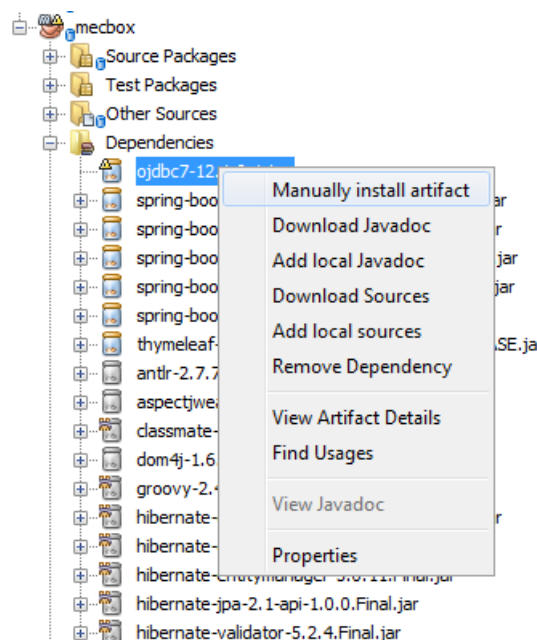
```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc7</artifactId>
  <version>12.1.0.1</version>
</dependency>
```

and comment the dependency *mysql*

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

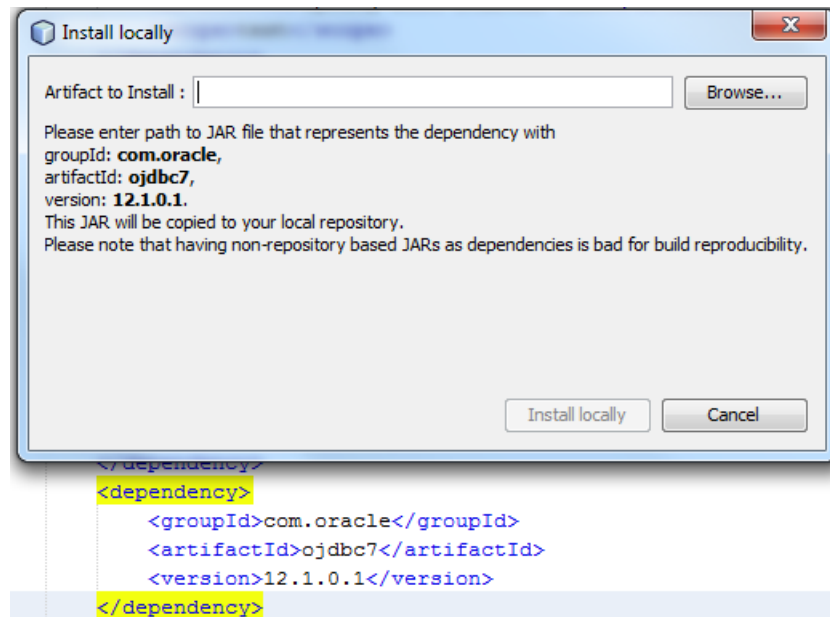
2. **Install Oracle driver:** due to Oracle license restriction, there are no public repositories that provide Oracle driver. Get *ojdbc7.jar* located in *mecbox/lib* and install it in your local maven repository. You can do this by command line or using your IDE.

In Netbeans right-click on the jar and select “Manually install artifact”





Then select *ojdbc7.jar* located in *mecbox/lib* and click “Install locally”.



### 3. Edit *application.properties* file: modify the datasource url for Oracle

```
spring.datasource.url = jdbc:oracle:thin@SERVER:PORT
```

change the dialect commenting Mysql dialect and uncommenting Oracle dialect

```
#Mysql dialect  
#spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect  
#Oracle dialect  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle12cDialect
```

### 4. Edit domain class *User* and *UserRole*: the primary key values of the tables MB\_USERS and MB\_USER\_ROLES are generated using Oracle sequences. Therefore it is necessary to modify the domain classes *User* and *UserRole* adding the annotations *@SequenceGenerator* and *@GeneratedValue*.

In both domains remove the annotation:

```
@GeneratedValue(strategy = GenerationType.AUTO)
```

Insert the following annotations on the variables *userid* e *userroleid*

### Domain User

```
@SequenceGenerator(name = "ID_SEQ_MB_USERS", sequenceName = "SEQ_MB_USERS",  
allocationSize = 1)  
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="ID_SEQ_MB_USERS")  
@Column(name = "userid")  
private Long userid;
```

### Domain UserRole

```
@SequenceGenerator(name = "ID_SEQ_MB_USER_ROLES", sequenceName =  
"SEQ_MB_USER_ROLES")  
@GeneratedValue(strategy=GenerationType.SEQUENCE,  
generator="ID_SEQ_MB_USER_ROLES")  
@Column(name = "user_role_id")  
private Long userroleid;
```

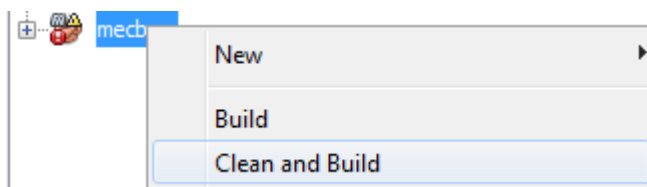
## 4.3 With Netbeans 8

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

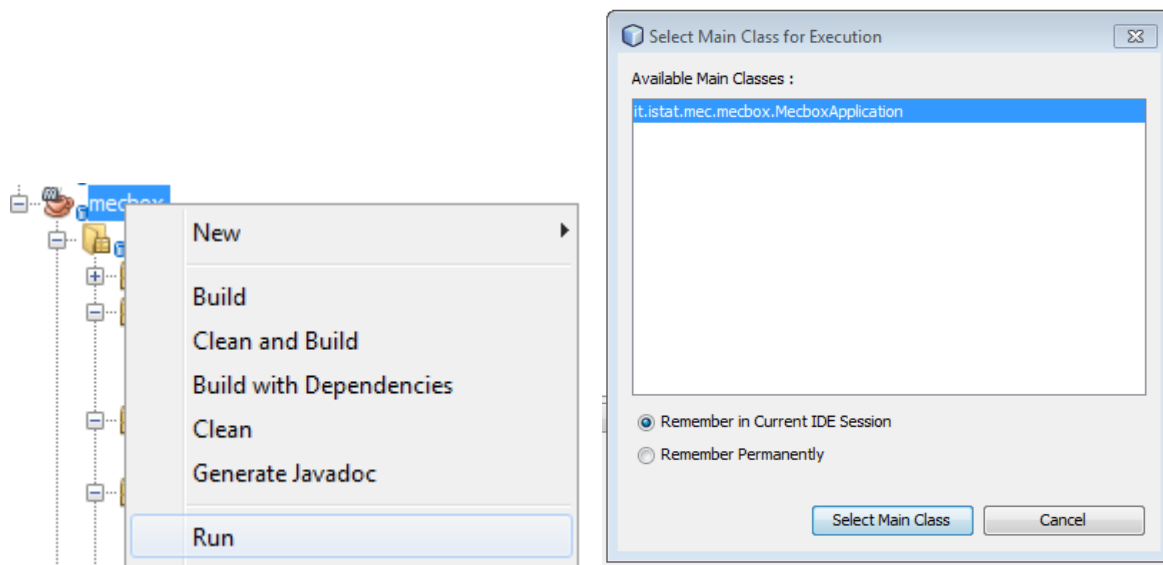
As a first step check the content of the *application.properties* file, located in the path *Other Sources > src/main/resources*:

```
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/mecbox>

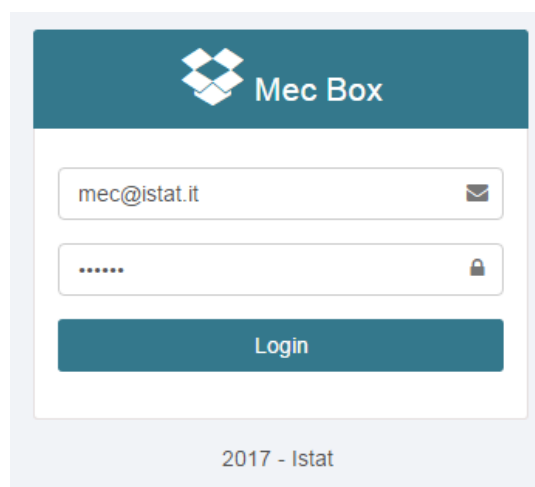
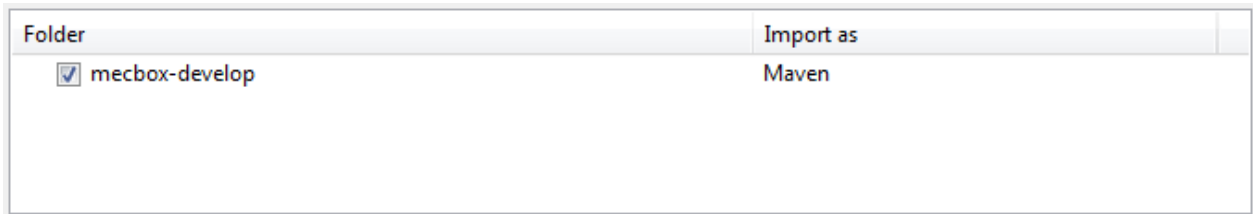


Figure 1 - Login page

## 4.4 With Eclipse (Neon.2)

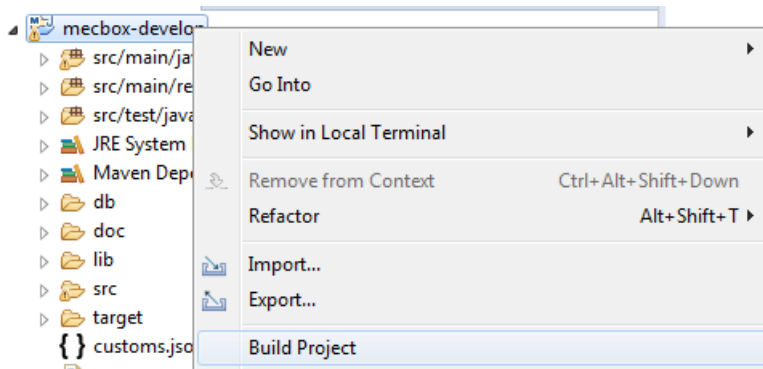
From the main menu select *File/Open Projects from File System...* Then select the folder containing the unzipped maven project and click the button 'Finish'. Eclipse import the project as a Maven project.



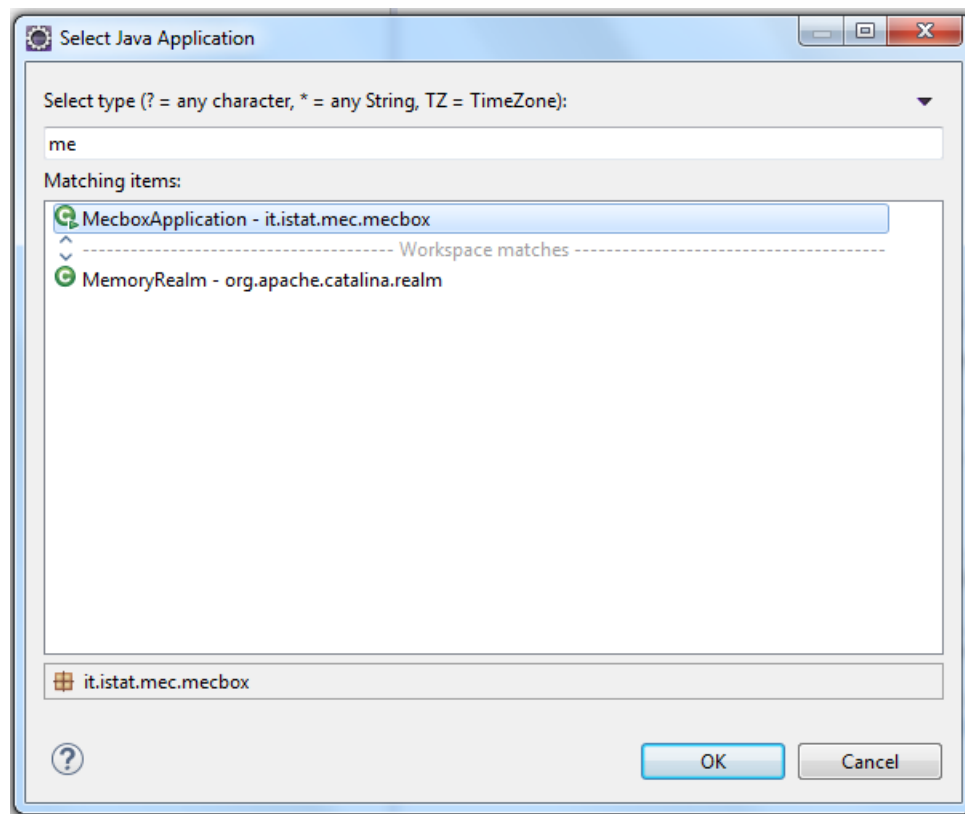
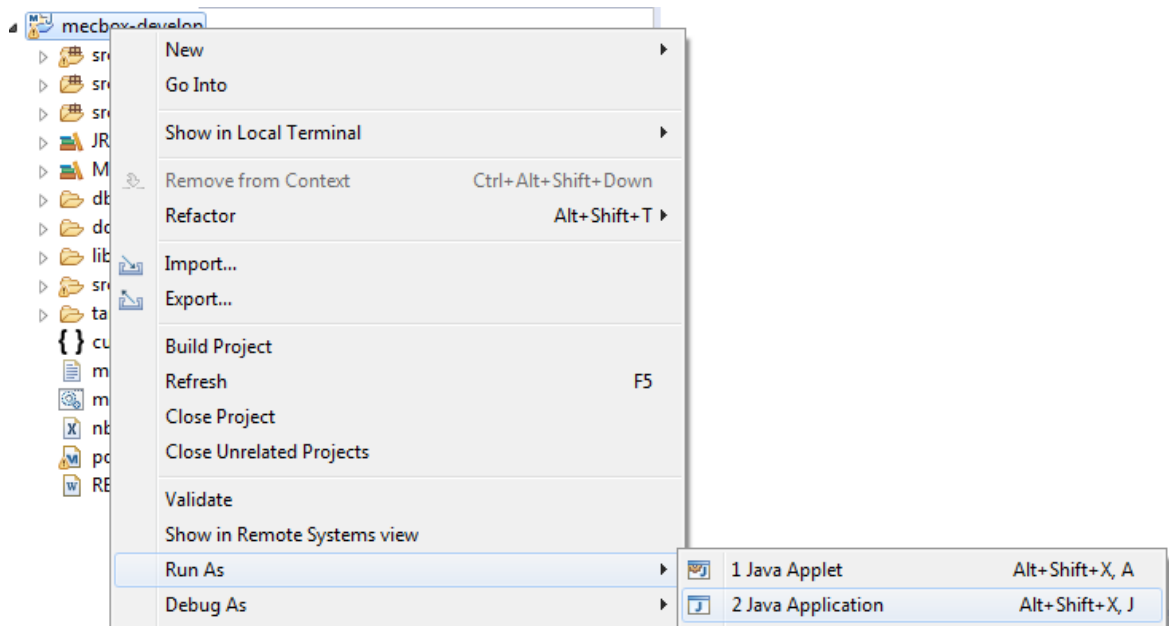
As a first step check the content of the *application.properties* file, located in the path *src/main/resources*:

```
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".



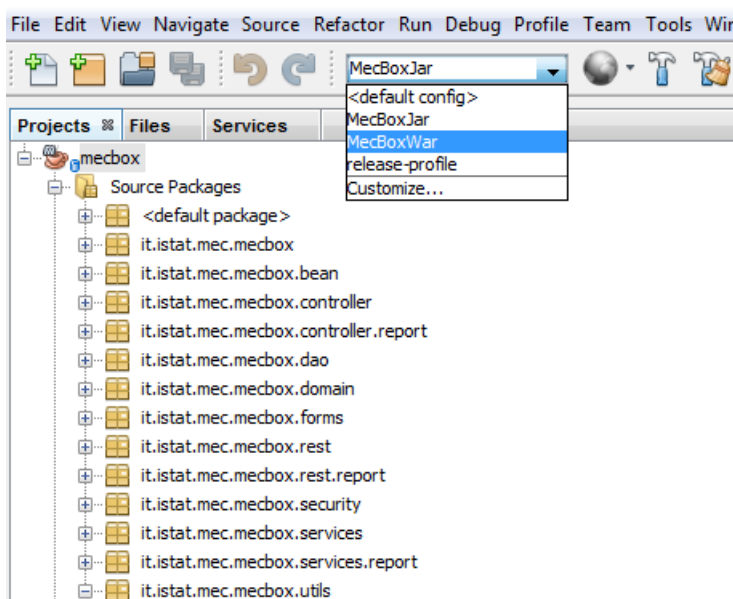
As shown above, first of all select the *Run As/Java Application* option, then the IDE will display a modal window and you will have to select the main class *MecboxApplication*. Now you can access the url:

<http://localhost:8080/mecbox>

## 4.5 Application server deployment

The project file *pom.xml* has been designed to build the application both as a jar and as a war. Therefore *mecBox* can be run both as an executable jar from the command line or it can be deployed on an application server (ex. Tomcat).

Select the Project configuration *MecBoxWar* and run Build.



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

## 5. How it's made

The application is built using several components which integrate state-of-the-art open source java framework.

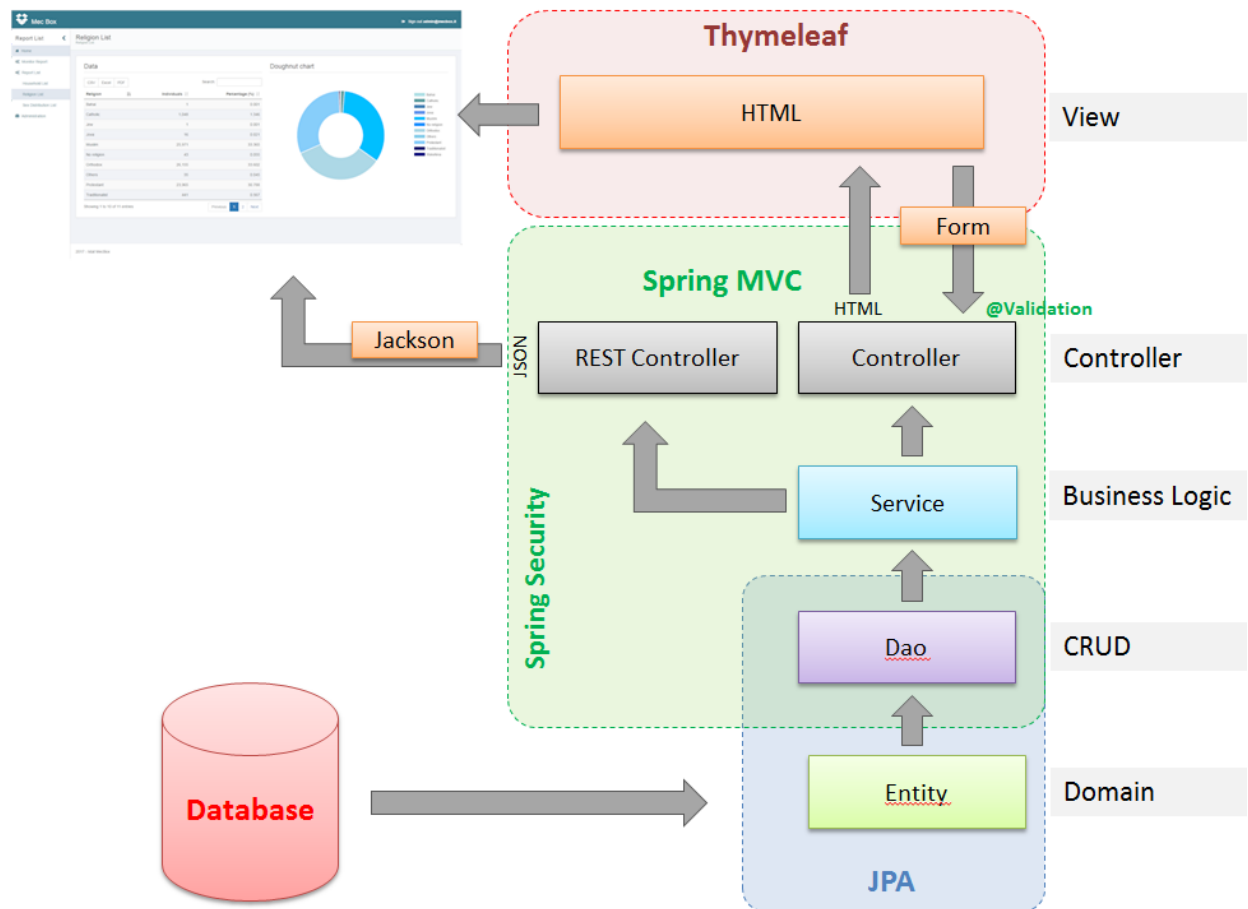


Figure 2 – MecBox data flow

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)

## 5.1 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**.

### 5.1.1 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 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>

### 5.1.2 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>

### 5.1.3 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*, 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.

#### 5.1.4 it.istat.mec.mecbox.controller

This package contains the classes that map URLs onto an entire class or a particular handler method.

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.

In order to understand the meaning of controller annotations we consider, as an example, the annotations used in the controller class *UserController*, that manages the login and exposes the URLs that map the User CRUD functionalities.

*@Controller*: indicates that a particular class serves is 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.

*@RequestMapping*: this annotation maps URLs such as */users/newUser* 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.

*@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.

#### More info:

Spring Validation, Data Binding, and Type Conversion references:

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

#### 5.1.5 it.istat.mec.mecbox.rest

This package contains the REST Controllers. In Spring's approach to build RESTful web services, HTTP requests are handled by a controller identified by the `@RestController` annotation.

In order to understand the meaning of a REST controller annotations we consider, as an example, the annotations used in the `UserRestController` class. Such class handles requests (all HTTP methods) for `/users/`

REST endpoint	HTTP method	Description
<code>/users</code>	GET	Returns the list of users
<code>/users/{id}</code>	GET	Returns user detail for given customer {id}
<code>/users</code>	POST	Creates new user from the post data
<code>/users/{id}</code>	PUT	Replace the details for given user {id}
<code>/users/{id}</code>	DELETE	Delete the user for given user {id}

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 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.

Thanks to Spring's HTTP message converter support, you don't need to do this conversion manually. Jackson 2 is automatically chosen to convert the *User* instance to JSON.

**More info:**

Spring REST Controller:

<https://spring.io/guides/gs/rest-service/>

<https://spring.io/guides/tutorials/bookmarks/>

### 5.1.6 it.istat.mec.mecbox.forms

This package contains the bean objects mapping the HTML forms. The validation rules can be specified using specific annotation (ex. `@NotNull`, `@Size(min = 2, max = 50)`, etc.)

**More info:**

Bean Validation references:

<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://hibernate.org/validator/>

[http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html\\_single/#validator-defineconstraints-hv-constraints](http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#validator-defineconstraints-hv-constraints)

### 5.1.7 it.istat.mec.mecbox

This package contains three classes *WebSecurityConfig*, *MecBoxApplication* and *AppConfiguration*.

*WebSecurityConfig* is Spring Security configuration class. This 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 configures spring security to use custom *UserDetailsService* (you find the source code in the package described in section 5.1.3).

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`.

*MecBoxApplication* main class invokes Spring Boot's *SpringApplication.run()* method to launch the application. Therefore you can run MecBox without deploying it on an application server. Just execute from the command line:

```
java -jar mecbbox-1.0.jar
```

*AppConfiguration* is the internazionalitazion (i18n) configuration class. The annotation *@Configuration* indicates that a class declares one or more *@Bean* methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

More details are provided in section 5.2.

### More info:

Set up Spring Security:

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

Spring Boot tutorial:

<https://spring.io/guides/gs/spring-boot/>

#### 5.1.8 it.istat.mec.mecbox.bean

In this package you can store bean classes bound to specific business needs

#### 5.1.9 it.istat.mec.mecbox.security

This package contains the class *CustomUserDetails* that represents the authenticated user within Spring Security framework.

In this section we briefly describe how the framework has been configured in the application.

## Spring Security User Bean

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

```
package org.springframework.security.core.userdetails;

import java.io.Serializable;
import java.util.Collection;
import org.springframework.security.core.GrantedAuthority;

public interface UserDetails extends Serializable {

    public Collection<? extends GrantedAuthority> getAuthorities();

    public String getPassword();

    public String getUsername();

    public boolean isAccountNonExpired();

    public boolean isAccountNonLocked();

    public boolean isCredentialsNonExpired();

    public boolean isEnabled();
}
```

*CustomUserDetails* class implements *UserDetails* interface to hold on to the User object.

The implementation is very simple in terms of the overriding methods, *CustomUserDetails* just returns the User username and authorities (i.e. list of user roles). The constructor receives an instance of the authenticated User domain and the corresponding list of user roles.

```
package it.istat.mec.mecbox.security;

import java.util.Collection;
import java.util.List;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.util.StringUtils;

import it.istat.mec.mecbox.domain.User;

public class CustomUserDetails extends User implements UserDetails {

    private static final long serialVersionUID = 1L;
    private List<String> userRoles;

    public CustomUserDetails(User user, List<String> userRoles) {
        super(user);
        this.userRoles = userRoles;
    }
}
```

## Spring Security Service

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

```
@Service
public class CustomUserDetailsService implements UserDetailsService {

    private final UserDao userDao;
    private final UserRolesDao userRolesDao;

    @Autowired
    private NotificationService notificationService;

    @Autowired
    private AuthenticationManager am;

    @Autowired
    public CustomUserDetailsService(UserDao userDao, UserRolesDao userRolesDao) {
        this.userDao = userDao;
        this.userRolesDao = userRolesDao;
    }

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userDao.findByEmail(email);
        CustomUserDetails cud;
        if (null == user) {
            notificationService.addErrorMessage("No user present with user:: " + email);
            throw new UsernameNotFoundException("No user present with user: " + email);
        } else {
            List<String> userRoles = userRolesDao.findRoleByEmail(email);
            cud = new CustomUserDetails(user, userRoles);
            return cud;
        }
    }

    public void authenticate(String name, Object password) {
        Authentication request = new UsernamePasswordAuthenticationToken(name, password);
        Authentication result = am.authenticate(request);
        SecurityContextHolder.getContext().setAuthentication(result);
    }
}
```

## 5.2 Internationalization

In order to implement multilanguage support in MecBox different components have been configured.

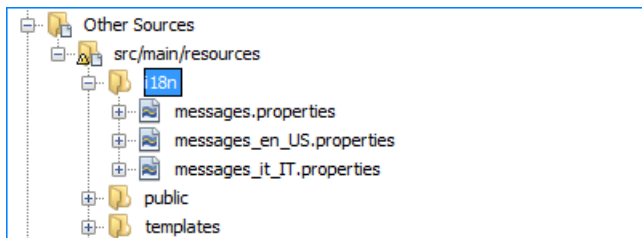
### Configuration bean

Three beans are registered to make Spring MVC application supports the internationalization.

- *LocaleResolver*: it resolves the locales by getting the predefined attribute from user's session.
- *LocaleChangeInterceptor*: this interceptor supports the multiple languages. It has to be added to the interceptor registry overriding the method *addInterceptor()*. The parameter "language" is used to set the locale.
- *ResourceBundleMessageSource*: it accesses the resource bundles using specified basenames

### Spring Resource Bundle

Two properties files to store English and Italian messages:



```
#Layout
Layout.household=Household
Layout.religion=Religion
Layout.sexdistribution=Sex Distribution
```

### Thymeleaf HTML page

In order to resolve the messages specified in the resource bundles Thymeleaf defines two different features of its Standard Dialect:

- The *th:text* attribute, which evaluates its value expression and sets the result of this evaluation as the body of the tag it is in.
- The *#{Layout.household}* expression, specified in the Standard Expression Syntax, specifying that the text to be used by the *th:text* attribute should be the message with



the *Layout.household* key corresponding to whichever locale we are processing the template with.

```
<ul id="process-2" class="nav sidebar-subnav collapse" aria-expanded="false">
  <li id="household-list">
    <a title="Household" th:href="@{/household/householdlist}">
      <i class="fa fa-table"></i>
      <span th:text="#{Layout.household}">Household</span>
    </a>
  </li>
  <li id="religion-list">
    <a title="Religion" th:href="@{/religion/religionlist}">
      <i class="fa fa-pie-chart"></i>
      <span th:text="#{Layout.religion}">Religion</span>
    </a>
  </li>
  <li id="sexdistribution-list">
    <a title="Sex Distribution" th:href="@{/sexdistribution/sexdistributionlist}">
      <i class="fa fa-bar-chart"></i>
      <span th:text="#{Layout.sexdistribution}">Sex Distribution</span>
    </a>
  </li>
</ul>
```

## 5.3 Client packages

In the following section is described how the client packages are structured.

### ➤ public.css

This package contains all stylesheet files related to client frameworks:

- mecbbox
- bootstrap
- fontawesome
- datatable
- metisMenu ( jQuery menu plugin)

### ➤ public.js

This package contains all javascript files related to client frameworks:

- mecbbox
- bootstrap
- jQuery
- Datatable
- Chart.js
- metisMenu (jQuery menu plugin)

### ➤ public.fonts

This package contains Fontawesome fonts.

### ➤ templates

This package contains all html files.

In order to generate dynamic html pages Mecbox uses *Thymeleaf*.

**Thymeleaf** is a modern server-side Java template engine for both web and standalone environments. Thymeleaf's main goal is to bring *natural templates* to your development workflow (ex. template.html in template package, displayed in Figure 3). HTML templates written in Thymeleaf still look and work like HTML, letting the actual templates that are run in your application keep working as useful design artifacts.

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head th:replace="layout :: site-head"/>
  <body>
    <!-- Body -->
    <div class="wrapper">

      <header th:replace="layout :: site-header" />
      <aside th:replace="layout :: site-aside" />

      <!-- START MAIN CONTAINER -->
      <section>
        <div class="content-wrapper">
          <div class="content-heading">
            Template page
            <small>template</small>
          </div>
          <div class="container-fluid">
            <div class="row mb-lg">
              <div class="col-lg-6">
                <h4>To Do</h4>
              </div>
            </div>
          </div>
        </div>
      </section>
      <!-- END MAIN CONTAINER -->

      <footer th:replace="layout :: site-footer" />
    </div><!-- wrapper -->
  </body>
</html>

```

Figure 3 - MecBox Thymeleaf template

The HTML template page refers to the application layout which is defined in *layout.html*. This means that the 'static' part of the application is defined once, while the developer must implement only the 'dynamic' elements of the page.

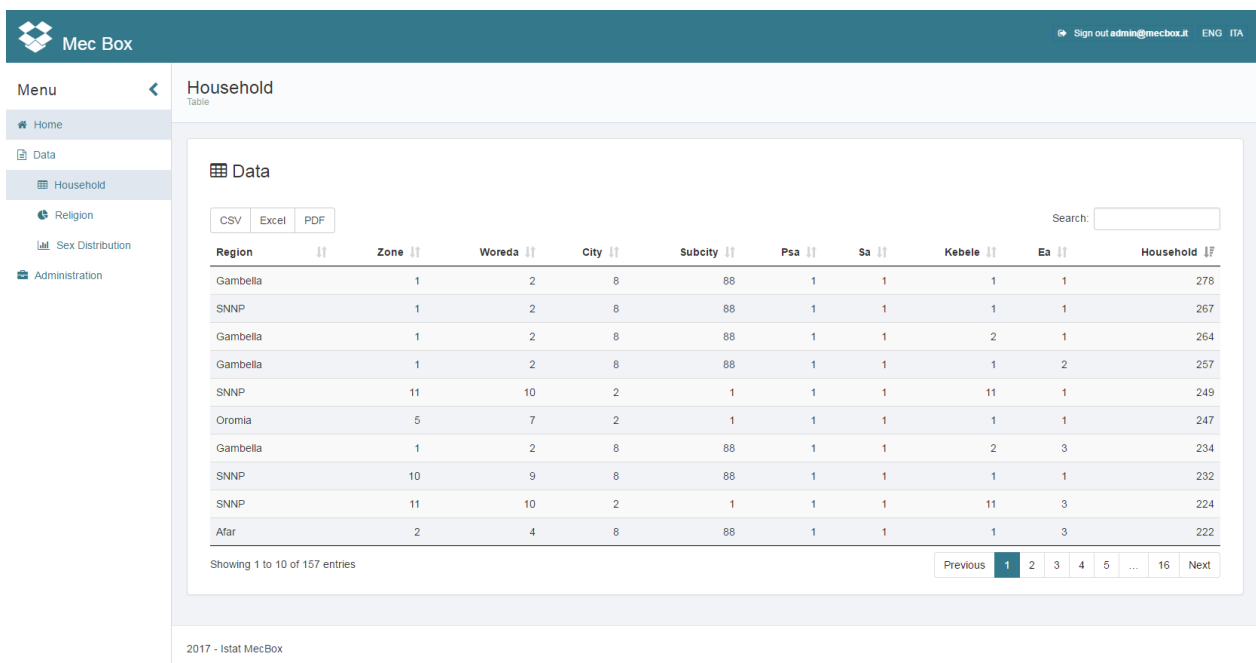
More details are provided in section 0 .

## 6. How you'll extend it

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

In Figure 4 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. *Household*). 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



The screenshot shows the 'Mec Box' web application interface. On the left is a sidebar menu with options: Home, Data, Household (selected), Religion, Sex Distribution, and Administration. The main content area is titled 'Household Table'. It features a 'Data' section with tabs for CSV, Excel, and PDF. Below these is a search bar and a table with 11 columns: Region, Zone, Woreda, City, Subcity, Psa, Sa, Kebele, Ea, and Household. The table displays 10 rows of data. At the bottom, it indicates 'Showing 1 to 10 of 157 entries' and includes a pagination control with 'Previous', '1' (selected), '2', '3', '4', '5', '...', '16', and 'Next'.

Region	Zone	Woreda	City	Subcity	Psa	Sa	Kebele	Ea	Household
Gambella	1	2	8	88	1	1	1	1	278
SNNP	1	2	8	88	1	1	1	1	267
Gambella	1	2	8	88	1	1	2	1	264
Gambella	1	2	8	88	1	1	1	2	257
SNNP	11	10	2	1	1	1	11	1	249
Oromia	5	7	2	1	1	1	1	1	247
Gambella	1	2	8	88	1	1	2	3	234
SNNP	10	9	8	88	1	1	1	1	232
SNNP	11	10	2	1	1	1	11	3	224
Afar	2	4	8	88	1	1	1	3	222

Figure 4 - Display data in a table

### 6.1.1 Server side implementation

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

```
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "mb_household")
public class Household implements Serializable {

    private static final long serialVersionUID = 1L;

    @Column(name = "region")
    private String region;

    @Column(name = "zone")
    private Integer zone;

    @Id
    @Column(name = "household")
    private Long household;

    public Household() {

    }

}
```

**Data Access Object (DAO):** create a new DAO class that extends *CrudRepository* (ex. *HouseholdDao.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 *findByRegion()*, you must specify the method in the DAO class interface. A detailed description of JPA repositories is provided in section [5.1.2](#).

```
import it.istat.mec.mecbox.domain.Household;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface HouseholdDao extends CrudRepository<Household, Long> {

    public Household findByRegion(String region);

}
```

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

```
import it.istat.mec.mecbox.domain.Household;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import it.istat.mec.mecbox.dao.HouseholdDao;

@Service
public class HouseholdService {

    @Autowired
    private HouseholdDao householdDao;

    public List<Household> findAll() {
        return (List<Household>) this.householdDao.findAll();
    }

}
```

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

```
import it.istat.mec.mecbox.domain.Household;
import it.istat.mec.mecbox.services.report.HouseholdService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HouseholdController {

    @Autowired
    private HouseholdService householdService;

    @RequestMapping(value = "/household/householdlist")
    public String householdList(Model model) {

        List<Household> householdList = householdService.findAll();
        model.addAttribute("householdList", householdList);

        return "household/householdlist";
    }

}
```

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

```
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import it.istat.mec.mecbox.domain.SexDistribution;
import it.istat.mec.mecbox.services.report.SexDistributionService;

@RestController
public class SexDistributionRestController {

    @Autowired
    private SexDistributionService sexdistributionService;

    @RequestMapping(value = "/sexdistribution/restlist")
    public List<SexDistribution> sexdistributionlist(Model model) {

        List<SexDistribution> sexdistribution = sexdistributionService.findAll();
        return sexdistribution;
    }
}
```

## 6.1.2 Client side implementation

### HTML:

Create a new HTML page copying the content of the file `template.html` in the package `templates` (ex. `templates.household.householdlist.html`)

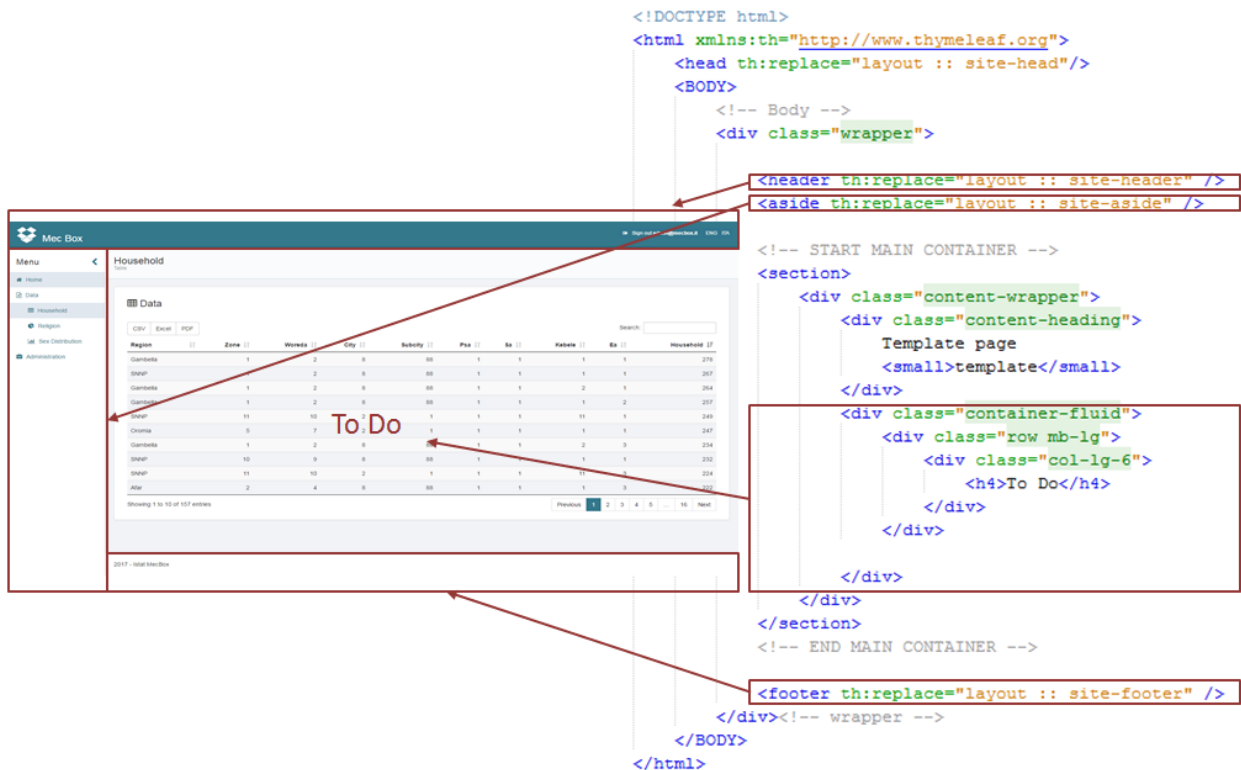


Figure 5 - MecBox HTML template

The template provides the implementation of the header, the footer and the sidebar. In order to implement your HTML page you should modify the content of the section fragment:

- **content-heading:** the title of your page
- **content-fluid:** the dynamic content of the page



## Table

To display in a HTML table the data stored in the *Household* table using Thymeleaf you simply iterate the model attribute set in the controller:

```
<table id="householdlist" class="table table-striped" cellspacing="0" width="100%">
  <thead>
    <tr>
      <th><span th:text="#{Householdlist.region}">region</span></th>
      <th><span th:text="#{Householdlist.zone}">zone</span></th>
      <th><span th:text="#{Householdlist.household}">household</span></th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="hh : ${householdList}"
      th:unless="${(householdList == null) or (householdList.size() == 0) }">
      <td th:text="${hh.region}"></td>
      <td th:text="${hh.zone}"></td>
      <td th:text="${hh.household}"></td>
    </tr>
  </tbody>
</table>
```

In order to render the HTML table with Datatable (described in section 7.2.1) you should implement a Javascript function to configure Datatable features (ex. export buttons, sorting, pagination, etc.).

```
var _ctx = $("meta[name='ctx']").attr("content");
$(".loading").hide();
$(document).ready(function () {
  setMenuActive("household-list");
  var table = $("#householdlist").DataTable({
    dom: "<'row'<'col-sm-6'B><'col-sm-6'f>>" + "<'row'<'col-sm-12'tr>>" +
      "<'row'<'col-sm-5'i><'col-sm-7'p>>",
    responsive: true,
    lengthChange: false,
    pageLength: 10,
    buttons: [{
      extend: 'csvHtml5',
      filename: 'household',
      title: 'household'
    }, {
      extend: 'excelHtml5',
      filename: 'household',
      title: 'household'
    }, {
      extend: 'pdfHtml5',
      filename: 'household',
      title: 'household'
    }],
    columnDefs: [{
      targets: [1, 2, 3, 4, 5, 6, 7, 8, 9],
      render: $.fn.dataTable.render.number(',', '.', 0), 'className': 'numeric'
    }],
    "fnPreDrawCallback": function() {}, "fnDrawCallback": function() {}
  });
});
```

## Chart

To display a graph using *Chart.js* (ex. *DoughnutChart*) you should perform the following steps:

1. execute an Ajax request to a specific rest controller (ex. */religionrest*), which returns the data in JSON format

```
$(document).ready(function () {  
    setMenuActive("religion-list");  
    var jqxhr = $.getJSON(_ctx + "/religionrest", function (json) {  
        console.log("success");  
    })  
});
```

2. parse the returned JSON object and set the Label, Data and Color arrays. In order to use the color palette, stored in *mecBoxChart.js*, choose a color from the *color* array. Set the variable *startFrom* with the index of the selected color.

```
.always(function (json) {  
    console.log("complete");  
    var startFrom = 55;  
    var individualsTotal = 0;  
    $.each(json, function (i, obj) {  
        arrLabel.push(obj.religion);  
        arrData.push(obj.individuals);  
        arrColor.push(color[i + startFrom ][2]);  
    });  
    $('<div>loading</div>').hide();  
    $('<div>religion-fluid</div>').animate(  
</pre>
```

3. Create a new instance of a Chart object.

```
var chartDiv = document.getElementById("chart-area").getContext("2d");  
new Chart(chartDiv, configDoughnut);
```

If you want to display your data in a Doughnut Chart or Bar Chart simply invoke the functions *renderDoughnut()* or *renderBar()* defined in *mecbox.js*.

The code of the function *renderDoughnut()* is displayed in the following image.

```

//function to render chart: type Doughnut
function renderDoughnut(arrData,arrColor,arrLabel) {

    var configDoughnut = {
        type: 'doughnut',
        data: {
            datasets: [{
                data: arrData,
                backgroundColor: arrColor
            }],
            labels: arrLabel
        },
        options: {
            responsive: true,
            legend: {
                position: 'right'
            },
            animation: {
                animateScale: true,
                animateRotate: true
            }
        }
    };

    var myChartDoughnut = document.getElementById("doughnut-chart-area").getContext("2d");
    var myDoughnut = new Chart(myChartDoughnut, configDoughnut);
}

```

## 7. Frameworks

In the following sections the frameworks used in MecBox are briefly described.

### 7.1 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>

#### 7.1.1 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/>

#### 7.1.2 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/>

## 7.2 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/>

### 7.2.1 DataTables

DataTables is a plug-in for the jQuery 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>

## 7.3 Thymeleaf

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

## 7.4 Bootstrap

Bootstrap is an 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/>

## 7.5 Chart.js

Chart.js is a Javascript chart library.

Documentation:

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

## 7.6 Fontawesome

Fontawesome 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 started:

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

## 7.7 Spring Data JPA

Spring Data JPA (Java Persistence API), part of the larger 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.

Project homepage:

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

Accessing data with JPA

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

## 7.8 LESSCSS

We considered using LESSCSS to make CSS more maintainable, themeable and extendable. This technique permits managing the CSS language, in a very simplest way thanks to variables definition, component declaration which can be modified in according to specific draw.

### 7.8.1 Variables

It is possible to declare the color palettes of the document, font-family, and other visual properties.

```
@box-background-color:  rgb(109, 158, 235);  
@box-color:  rgb(85,85,85,1);
```

### 7.8.2 Mixin

Another very interesting advantage of LESSCSS is to be able to define mixins, they consist of creating a block that can be reused several times during development:

```
.box-font {  
  font-family: Arial, sans-serif;  
  font-size:10px;  
}
```

Inside our `.less` file, we can, once defined before this block, re-arrange it as follows:

```
#header {  
  h1,h2,h3 {  
    .box-font;  
  }  
}
```

then compile to CSS

```
#header h1, #header h2, #header h3 {  
  font-family: Arial, sans-serif;  
  font-size:10px;  
}
```



### 7.8.3 Mixin and Functions

When we use Mixin in combination with Variables, we actually get the real Functions

- declare variable:

```
@box-color: rgb(85,85,85,1);
```

- define the Mixin

```
.box-font {  
  font-family: Arial, sans-serif;  
  font-size:10px;  
  color: @box-color;  
}
```

- Then compiles to CSS

```
#header h1, #header h2, #header h3 {  
  font-family: Arial, sans-serif;  
  font-size:10px;  
  color: rgb(85,85,85,1);  
}
```

Also remember LESSCSS has color-change functions for example:

```
Lighten(@box-background-color: , 10%);  
return a color which is 10% *lighter* than @box-background-color;  
  
darken(@box-background-color: , 10%);  
return a color which is 10% *darker* than @box-background-color;
```

To use LESSCSS you have to follow these steps.

- Edit the project file pom.xml

```
<dependency>  
  <groupId>org.lesscss</groupId>  
  <artifactId>lesscss</artifactId>  
  <version>1.7.0.1.1</version>  
</dependency>
```

- import *org.lesscss.LessCompiler*
- and then instantiate the LESS compiler

```
LessCompiler lessCompiler = new LessCompiler();  
lessCompiler.compile(new File("file.less"), new File("file.css"));
```

#### 7.8.4 Conclusion

Thanks to the lesscss compiler we have been able to customize bootstrap.css in a very easy way: using bootstrap variables and mixins stored in public/css/less we obtain custom\_bootstrap.css