REGIONE AUTONOMA DELLA SARDEGNA

SISaR

ENGINEERING

Servizio di Transizione

Titolo: XMPI/JBF/JAUF
Codice: AREAS_AP_01_JBF
Profilo: Analista Programmatore

REGIONE AUTONOMA DELLA SARDEGNA

SISaR

ENGINEERING

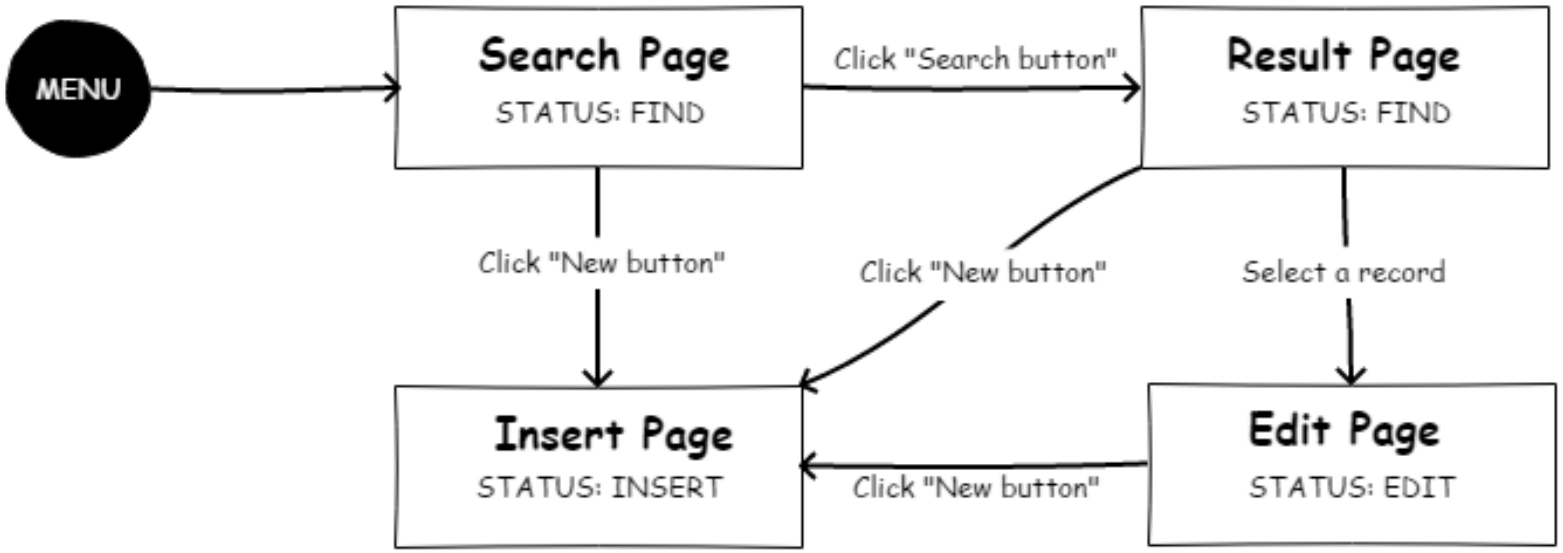
ARGOMENTI DI QUESTE SLIDE

- Modello architetturale MVC
- Esercizi:
  - Gestione testata/dettaglio
  - Lookup e tabelle generiche
  - Action Handler (FIND2, FIND3)
  - Gestione Attach
  - Service Manager e connessioni

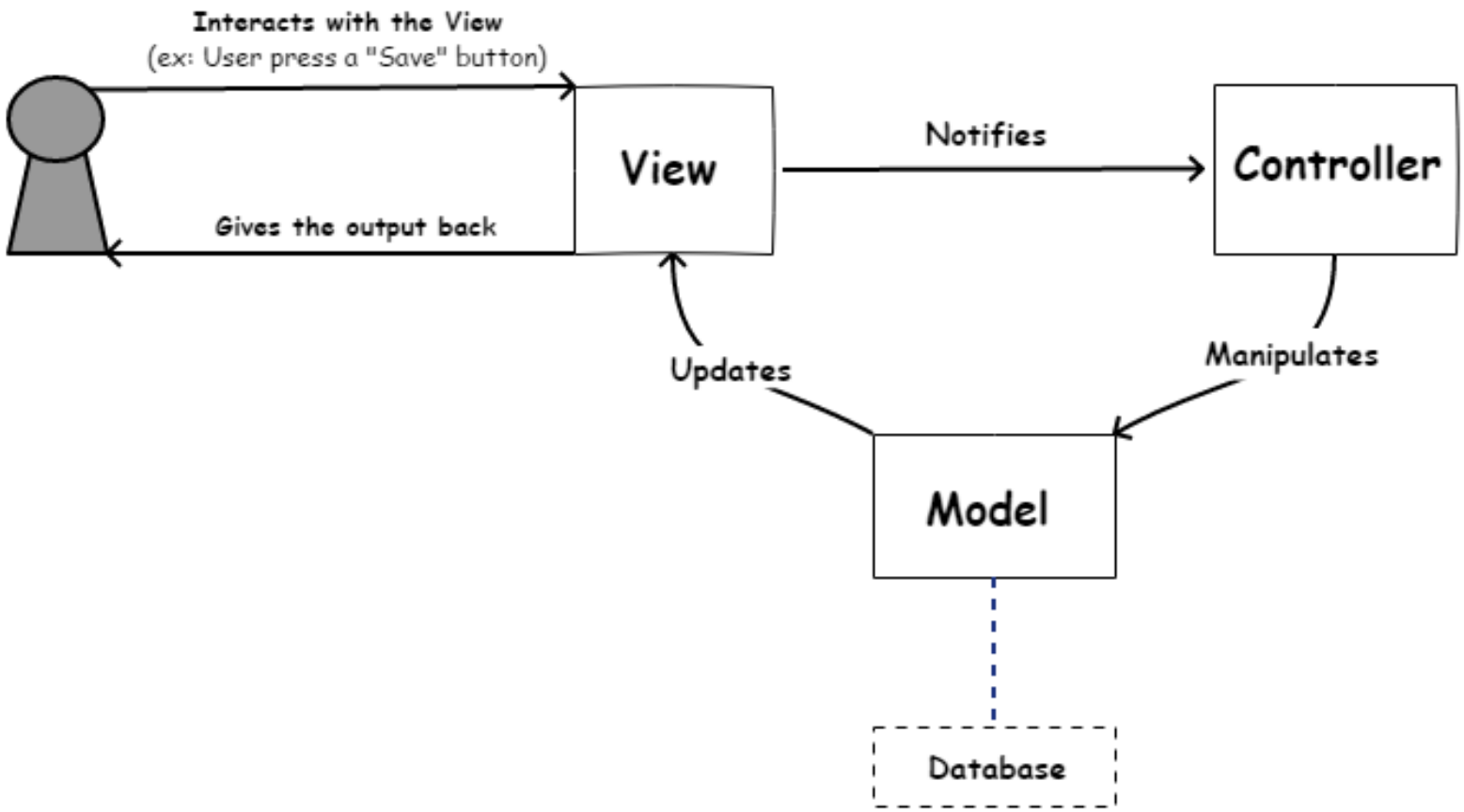
MVC Pattern in AREAS

AREAS uses a simple and standard **MVC pattern** to design all its web pages and functionalities.

We do a quick recap about the web pages in AREAS.



MVC Pattern in AREAS

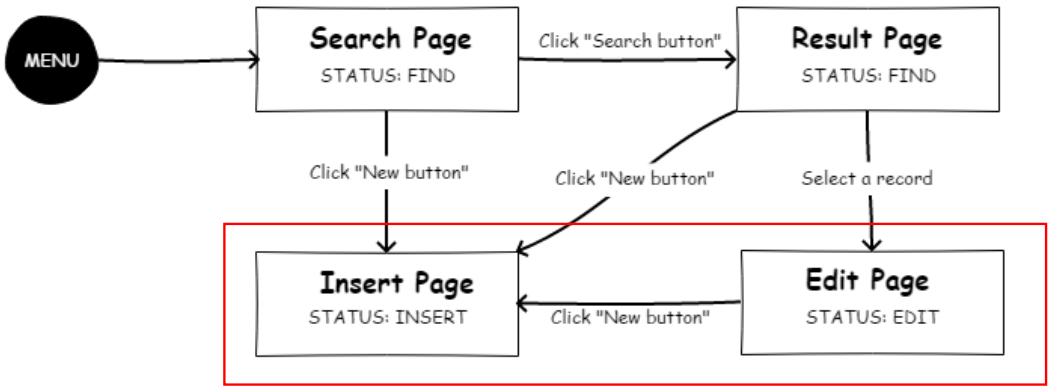


MVC Pattern in AREAS / The Model

In AREAS a **Model** is a simple Java Bean (**POJO**) that also implements some particular Java interfaces to handle the persistence layer and other additional information.

The model and the persistence layer are strictly connected, so it's the responsibility of the Model to persist itself inside the database.

One Bean is connected to one record of one table of the database. That makes the model involved in the web pages of the "**Edit page**" and "**Insert page**".




```
public class Dog {
  protected String name;
  protected String owner;

 public String getName() {
    return name;
  }
public void setName(String name) {
    this.name = name;
  }
…other getters and setters…
}
```

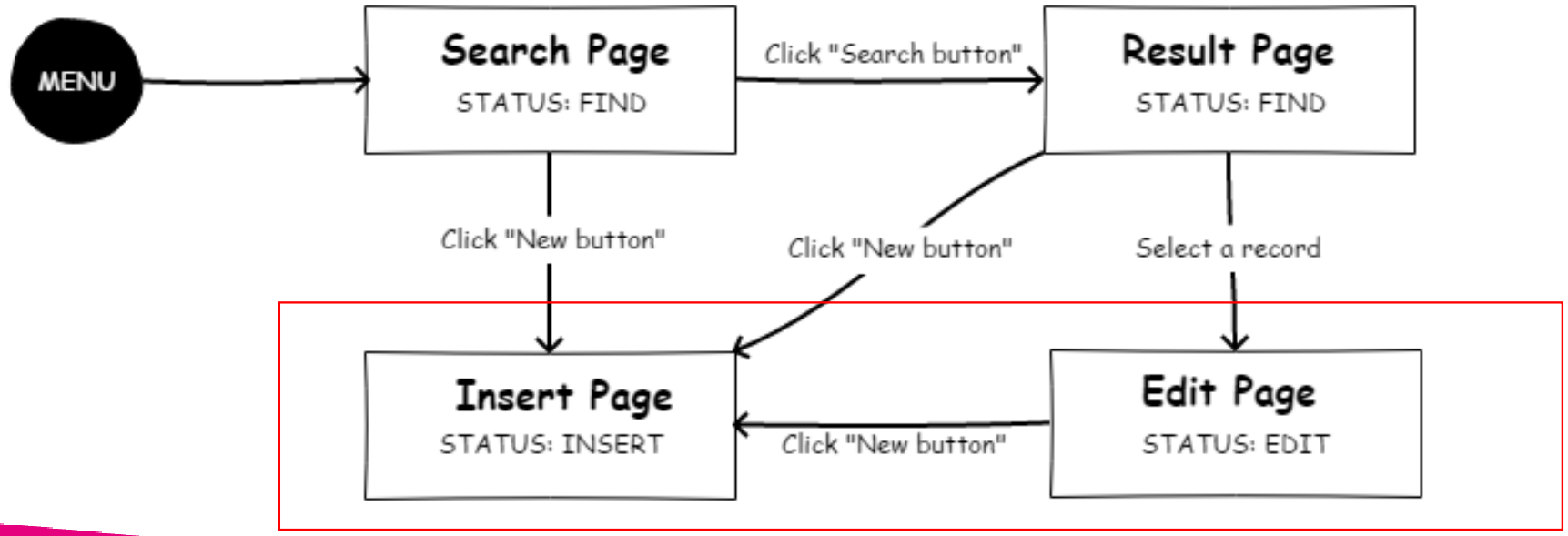
MVC Pattern in AREAS / The Model

More specifically, to design a Model in AREAS we need:

- **A POJO Bean**
- **An additional XML file that maps all the POJO attributes with the database fields**
- The POJO must implement the **Persistent** Java interface in order to implement these methods:
    - **Retrieve**. It's invoked to load a single record of the database into the Bean
    - **Store**. It's invoked to insert or update a record of the database
    - **Remove**. It's invoked to delete a record of the database
- The POJO must implement a **ValidableInterface** in order to implement this method:
    - **validate**. It's invoked before the Store method, to check if the Bean can be stored inside the database. It can throw a warning message (the bean get stored in any case) or an error message (the store method is not invoked).
- The POJO must implement a **DynamicInfoInterface** in order to implement this method:
    - **getObjectInfo**. It gives the bean the opportunity to describe itself more. By this method, the bean can tell the framework, for example, which properties are mandatory, or which properties are not editable.

Exercise 1

Based on what we just learned, we can start doing this exercise by designing the Model



MVC Pattern in AREAS / The Query

As we already said, one Bean is connected to one record of the database only. That makes the model involved in the web pages of the "**Edit page**" and "**Insert page**".

But when we search something trough the "**Search page**" we need to model a **list of records** (the result of the search) and not only one record. To do this, we introduce the concept of the "**Query class**".

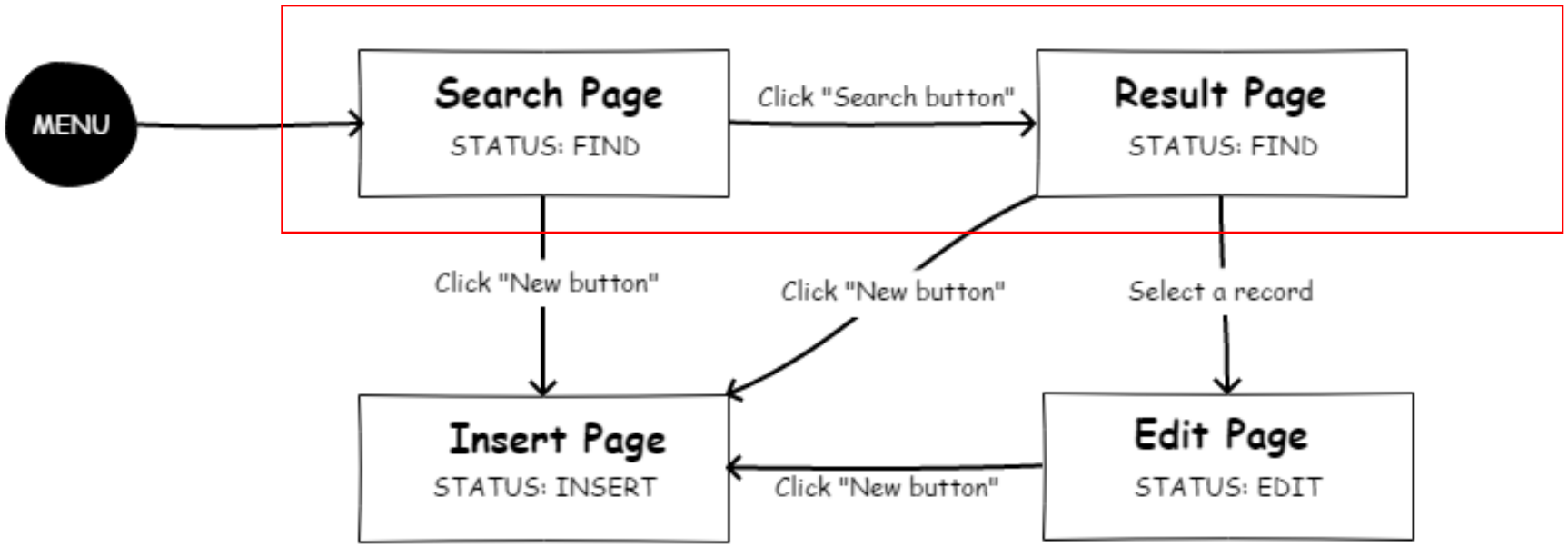
The purpose of this Java class is to build the SQL statement that will be executed on the Database. The result of the query is then parked inside an ArrayList variable, in memory.

More specifically, to realize a Query class in AREAS we need:

- A Java class extending the abstract class **CustomQueryProfile** and implementing this method:
    - **getCustomSQLString**. It's invoked when the user presses the "Search" button. By using the method **getFilterValue** we can access the filters and search criteria the user typed into the "Search page". The output of this method is the SQL that will be later executed by the framework.

Exercise 1

Based on what we just learned, we can keep doing this exercise by designing the Query



MVC Pattern in AREAS / The Views

The "**Views**" in AREAS are realized using plain Java JSP. Usually is enough to implement just two JSP:

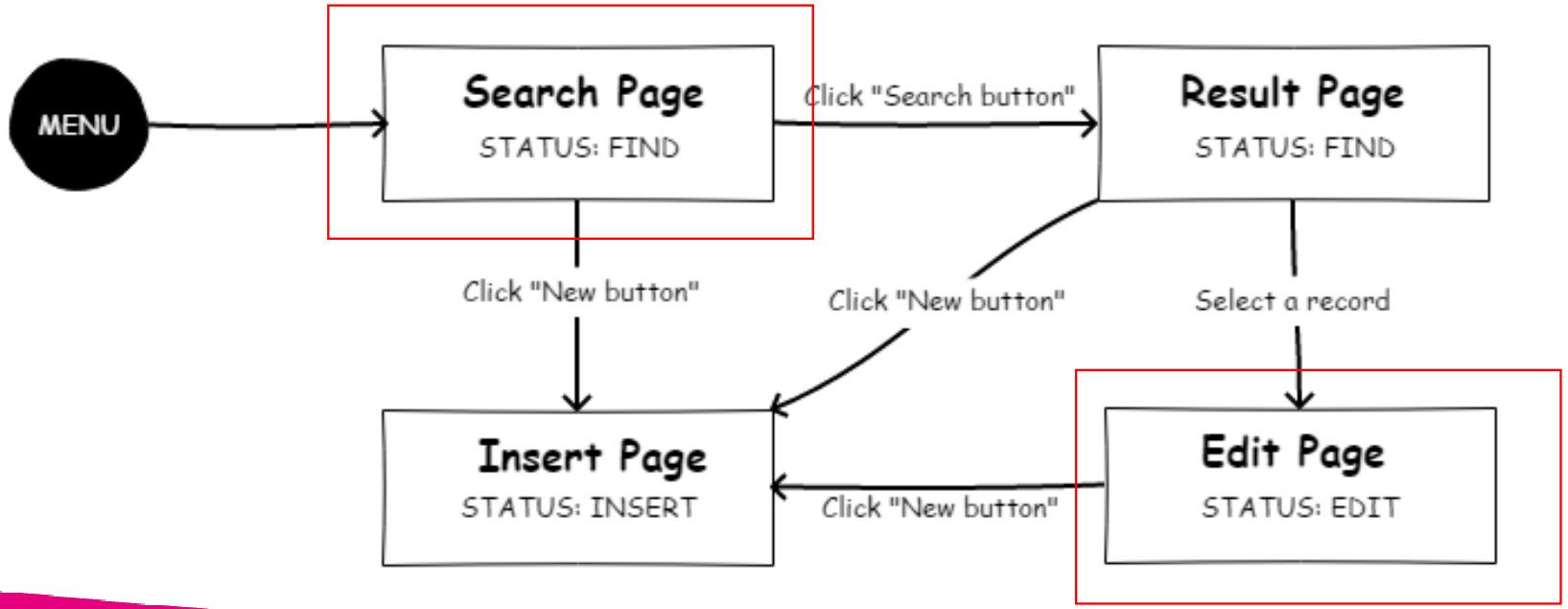
- **Find JSP**. It's the JSP that design the "Search page", showing the user the all possible filters he can type in
- **Edit JSP**. It's the JSP that design the "Edit page" and "Insert page", allowing the user to edit or insert a new record

The binding between Views and Model is guaranteed by a set of custom **Java Tag Library**. Inside this library, there is also a set of useful tags to implements conditions, loops and some advanced widgets (grids, dual list…etc..).

The interaction between the client (View) and server (Controller) is managed by invoking specific Javascript functions (**doAction**).

Exercise 1

Based on what we just learned, we can keep doing this exercise by designing all the needed Views



MVC Pattern in AREAS / The Controller

The **Controller** side is implemented by a Java Class, and its duty is to take care about all the action the user can submit through the Views.

More specifically, to realize a Controller in AREAS we need:

- A Java class extending the abstract class **W3ControllerPersistent** and implementing this method:
  - **init**. It's invoked when the user accesses our web pages in order to initialize the Controller. Some of the things we need to take care in this method are:
    - **Declaring our Views.** By using the method **setReturnPage**
    - **Declaring our Query class.** By using the method **setFindResultBuilder**

Exercise 1

Based on what we just learned, we can keep doing this exercise by designing the Controller. Once it is done, we can deploy what we just created!

MVC Pattern in AREAS / The Controller Dispatcher

If you give a deeper look on the Controllers in AREAS, you will notice that they are not servlet. That means that all the user's requests are not addressed directly to our controllers.

In AREAS there is one unique servlet, which is the entry point of all the user's requests. This servlet is called **ControllerDispatcher**.

The duty of the Dispatcher is to receive the user request and to address it to the right controller (managing also the initialization process).

To locate the right controller, all the controllers are identified by a **mapping.**

MVC Pattern in AREAS / The Mapping

A **mapping** is the logical name of a Controller, and it reppresents how the controller will be invoked via URL. Let's take the login page as an example: http://10.10.10.10:8080/areas/mainLogin.do

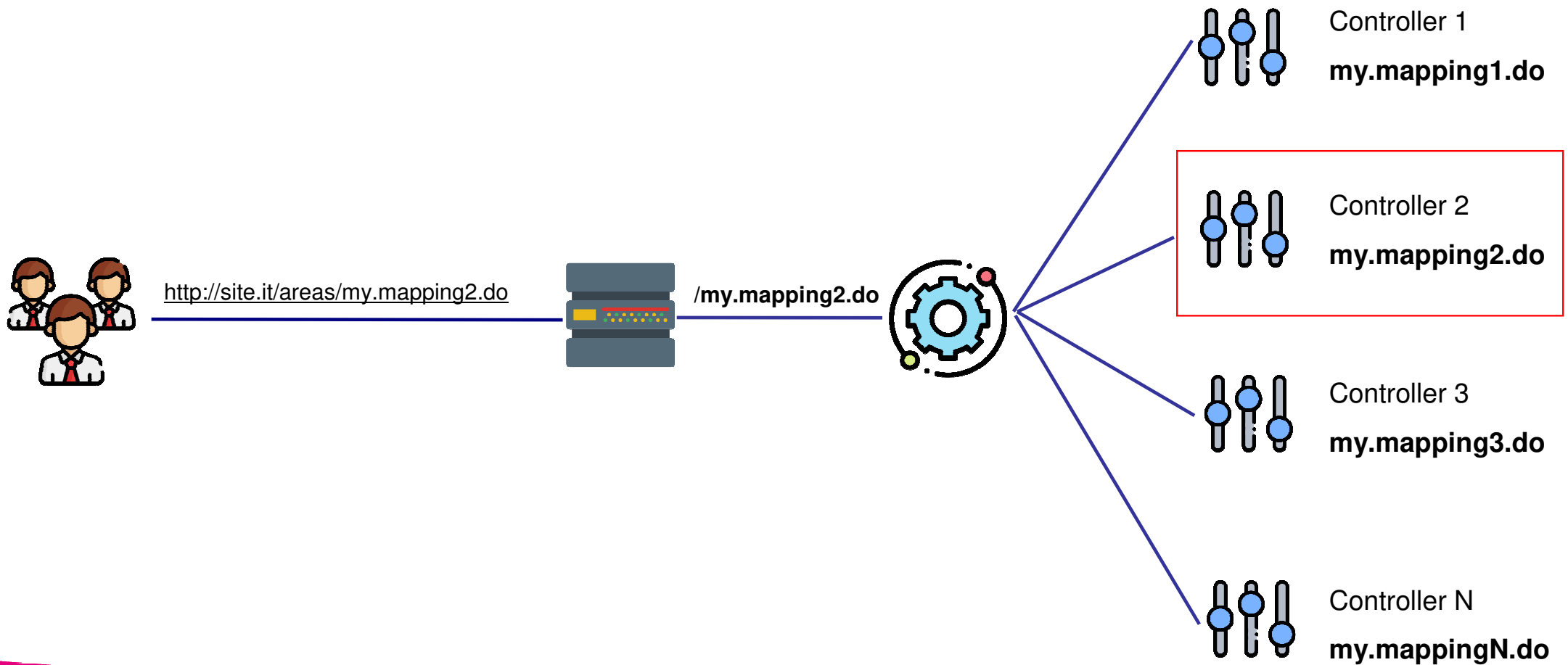
The **mainLogin.do** is the mapping of the Controller responsible for the login page. The association between the mapping and the corresponding Java Class, it's made inside a properties file located in every project of AREAS: /**jsp/WEB-INF/controllers** (es: controllers.properties in PSGLib).

mapping.mainLogin = PSGExt.profile.w3.Login2W3

The **ControllerDispatcher** is configured (inside the web.xml) to catch all the URL that ends with the string "**.do**". In our case the ControllerDispatcher:

1. takes the mapping part of the URL: **mainLogin**
2. looks inside all the controllers.properties if it exists a declared mapping
3. instantiates the Java class **Login2W3** and it calls its init method.

MVC Pattern in AREAS / The Controller Dispatcher Flow



Exercises

Keep up following all the other exercises.

We can find all exercises in the "***Allegato 2 - AREAS_AP_01_JBF - XMPI-JBF-JAUF - Exercises.pdf***"

And we can help ourself with the code snippets in the "***Allegato 1 - AREAS_AP_01_JBF - XMPI-JBF-JAUF - Code Snippets.pdf***"