

En esta sección, aprenderemos cómo crear una aplicación web usando Spring MVC, JDBC y Hibernate ORM framework sin configuración xml que no provenga del POM de Maven.

JAVA SPRING SIN XML

I

Creación de un CRUD con Spring



Temario



Crear y
listar



Borrar y
actualizar

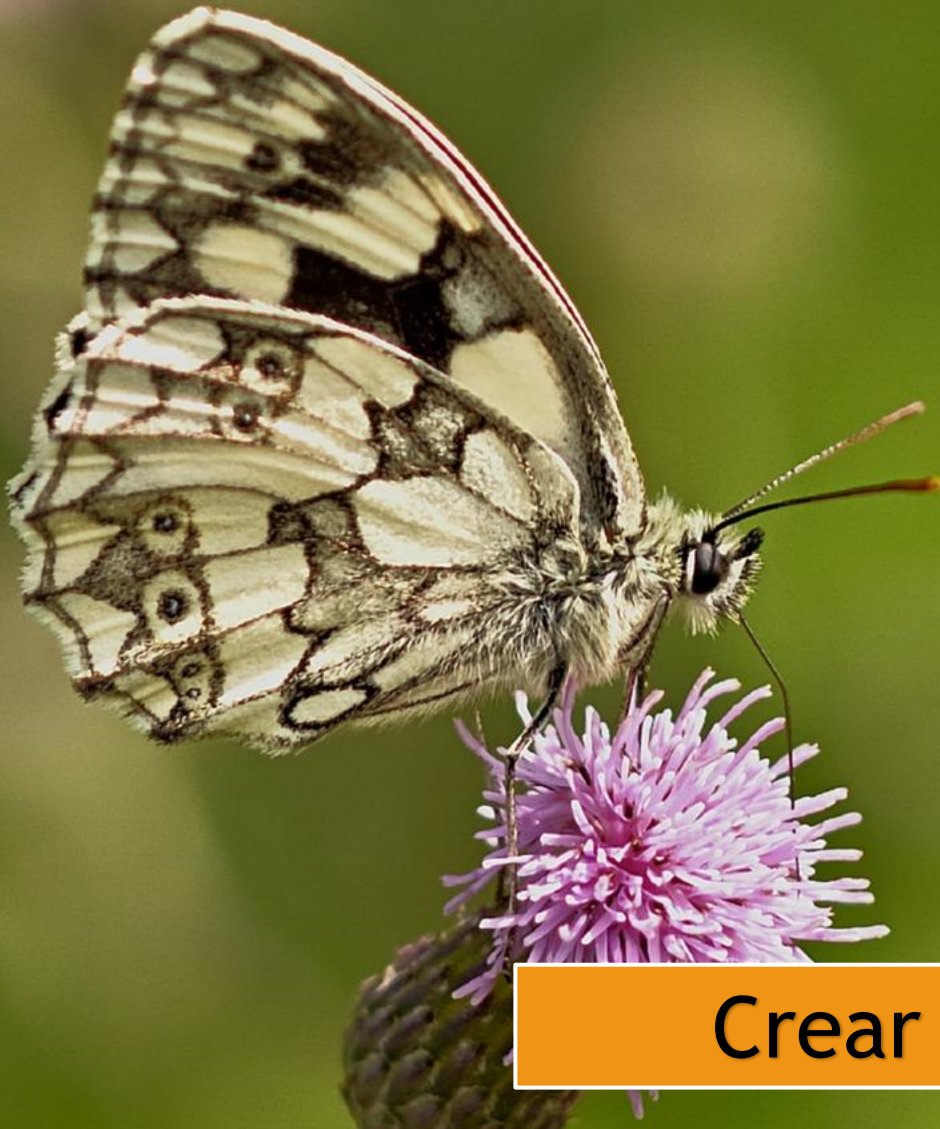


Mejorando
aspectos



Crear y listar





Crear y listar



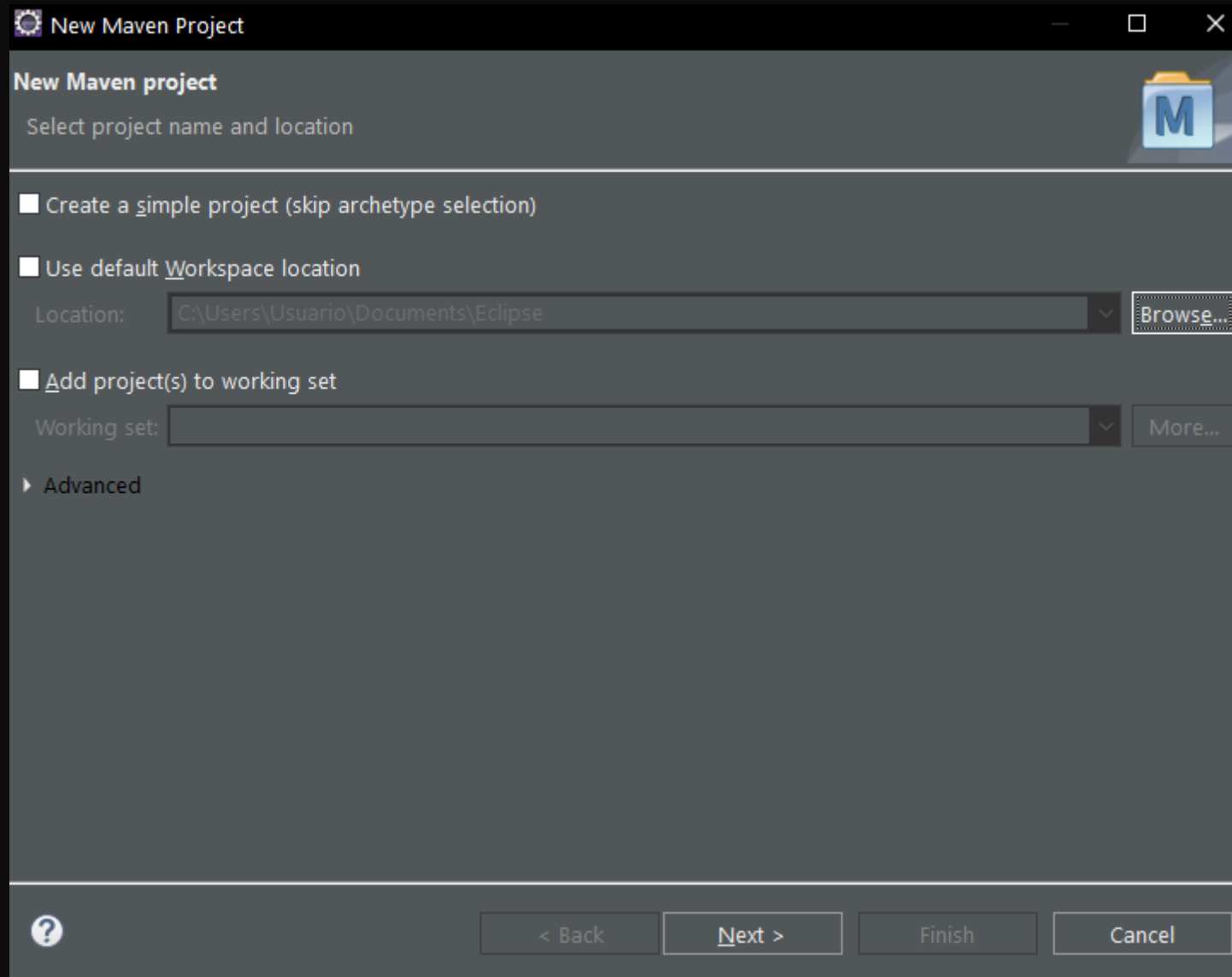
Creación del proyecto

1

En esta parte, crearemos el proyecto con las tecnologías Java, Spring y Maven



Selección del workspace de Eclipse



The screenshot shows the 'New Maven Project' dialog box in the Eclipse IDE. The dialog has a title bar with the Eclipse logo and the text 'New Maven Project'. Below the title bar, there is a section titled 'New Maven project' with a subtitle 'Select project name and location'. To the right of this section is a folder icon with a blue 'M'. The main area of the dialog contains three checkboxes: 'Create a simple project (skip archetype selection)', 'Use default Workspace location', and 'Add project(s) to working set'. The 'Use default Workspace location' checkbox is selected. Below this checkbox, there is a text field labeled 'Location:' containing the path 'C:\Users\Usuario\Documents\Eclipse'. To the right of the text field is a 'Browse...' button. Below the 'Add project(s) to working set' checkbox, there is a text field labeled 'Working set:' which is empty. To the right of the text field is a 'More...' button. At the bottom of the dialog, there is a section titled 'Advanced' with a right-pointing arrow. The bottom of the dialog features a footer with a question mark icon on the left and four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

New Maven Project

New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location: C:\Users\Usuario\Documents\Eclipse Browse...

☐ Add project(s) to working set

Working set: More...

▶ Advanced

? < Back Next > Finish Cancel



Selección del arquetipo

New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs Configure...

Filter: webapp

Group Id	Artifact Id	Version
net.wasdev.wlp.maven	liberty-archetype-webapp	3.0.M1
org.apache.axis2.archetype	quickstart-webapp	1.8.0
org.apache.cocoon	cocoon-22-archetype-webapp	1.0.0
org.apache.cocoon.archetype-webapp	cocoon-archetype-webapp	3.0.0-alpha-3
org.apache.marmotta	marmotta-archetype-webapp	3.4.0
org.apache.maven.archetypes	maven-archetype-webapp	1.4
org.apache.openejb.maven	tomee-webapp-archetype	1.7.5

An archetype which contains a sample Maven Webapp project.
<https://repo1.maven.org/maven2>


☒ Show the last version of Archetype only ☐ Include snapshot archetypes Add Archetype...

Advanced

? < Back Next > Finish Cancel



Dar un nombre al proyecto

 New Maven Project

New Maven project

Specify Archetype parameters

Group Id:

Artifact Id:


Version:

Package:

Properties available from archetype:

Name	Value

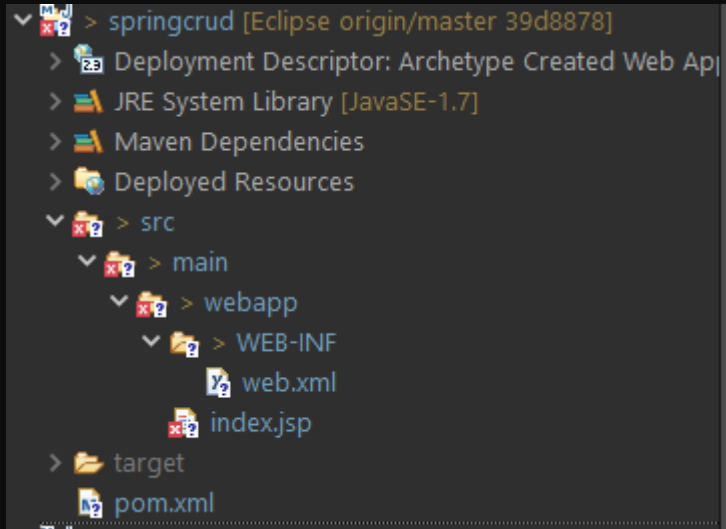
► Advanced





Limpiar estructura inicial

Borrar archivos no útiles



Indicar que no queremos utilizar XML

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <version>3.2.2</version>
  <configuration>
    <failOnMissingWebXml>>false</failOnMissingWebXml>
  </configuration>
</plugin>
```

Cambiar la propiedad inicial de JAVA

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```



Añadir dependencias “clásicas”

```
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>javax.servlet.jsp-api</artifactId>
  <version>2.3.1</version>
  <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.s
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>4.0.0</version>
  <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl --
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```



Añadir dependencias clásicas p2

```
<!-- https://mvnrepository.com/artifact/org.springframework
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.3.17</version>
</dependency>
<!-- PARA AÑADIR NUEVAS TAGS Y REALIZAR ACCIONES ESPECIALES
EN LOS FICHEROS JSP COMO BUCLES, CONDICIONALES...
https://mvnrepository.com/artifact/org.apache.taglibs/tagl
<dependency>
  <groupId>org.apache.taglibs</groupId>
  <artifactId>taglibs-standard-impl</artifactId>
  <version>1.2.5</version>
  <scope>runtime</scope>
</dependency>
```



Añadir nuevas dependencias

ORM de Spring

```
<!-- Spring ORM -->  
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-orm</artifactId>  
  <version>4.3.7.RELEASE</version>  
</dependency>
```

Para el mapeo de los objetos de nuestra APP

MYSQL Conector

```
<!-- Mysql Connector -->  
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <version>6.0.6</version>  
</dependency>
```

Para la conexión a la base de datos

Hibernate Core

```
<!-- Hibernate ORM -->  
<dependency>  
  <groupId>org.hibernate</groupId>  
  <artifactId>hibernate-core</artifactId>  
  <version>5.2.10.Final</version>  
</dependency>
```

Para relacionar nuestras clases con las tablas



Añadiendo nuevas dependencias p2

Hibernate C3P0

```
<!-- Hibernate-C3P0 Integration -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-c3p0</artifactId>
  <version>5.2.10.Final</version>
</dependency>
```

Para relacionar hibernate con las herramientas de C3P0

C3P0

```
<!-- c3p0 -->
<dependency>
  <groupId>com.mchange</groupId>
  <artifactId>c3p0</artifactId>
  <version>0.9.5.2</version>
</dependency>
```

Para controlar la forma que tiene Spring de realizar conexiones con la BDD y ahorrar recursos

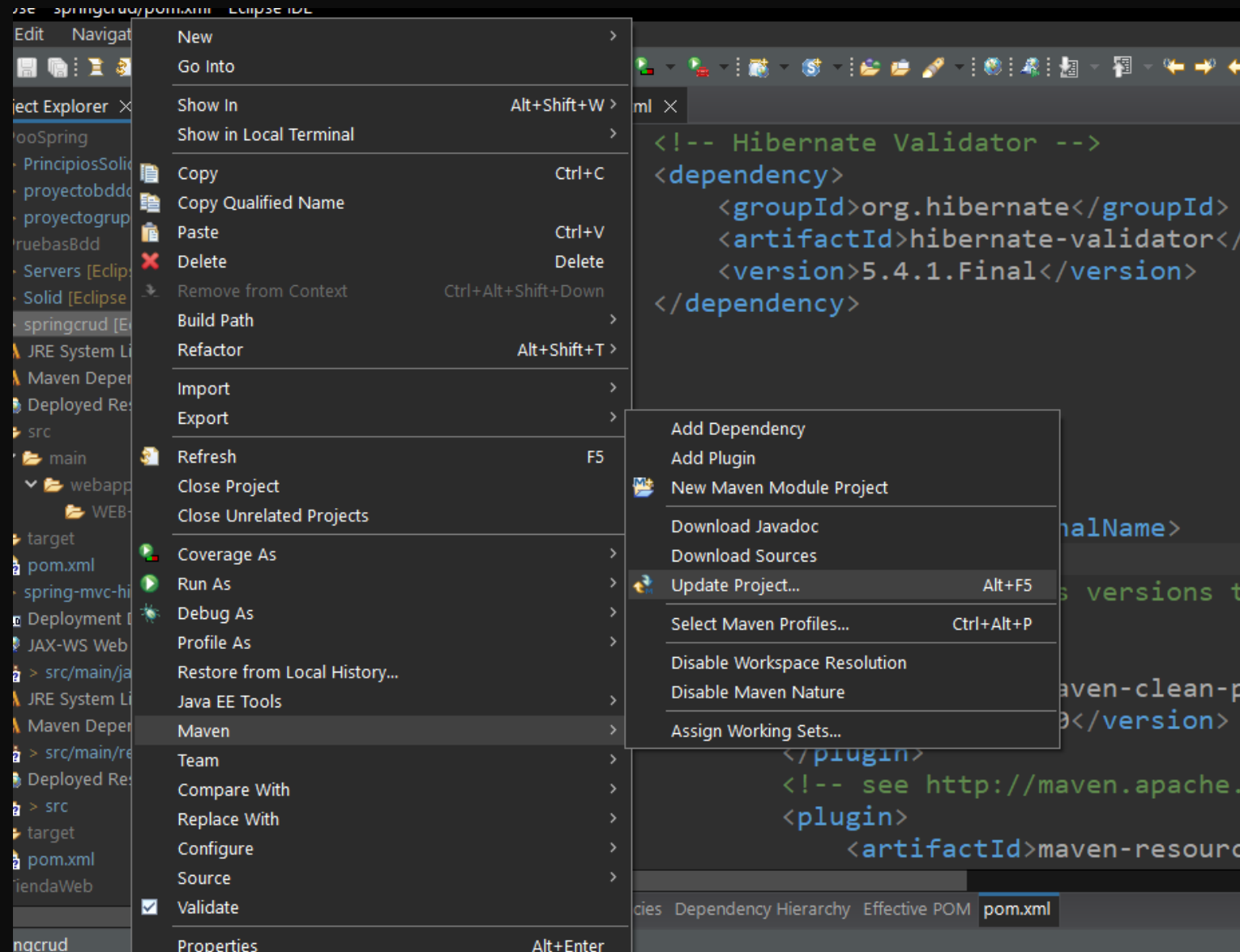
Hibernate validator

```
<!-- Hibernate Validator -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>5.4.1.Final</version>
</dependency>
```

Para validar entradas de usuario según reglas específicas



Actualizando el proyecto maven



Configuración del proyecto

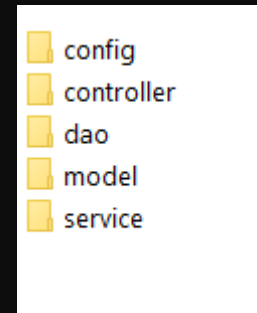
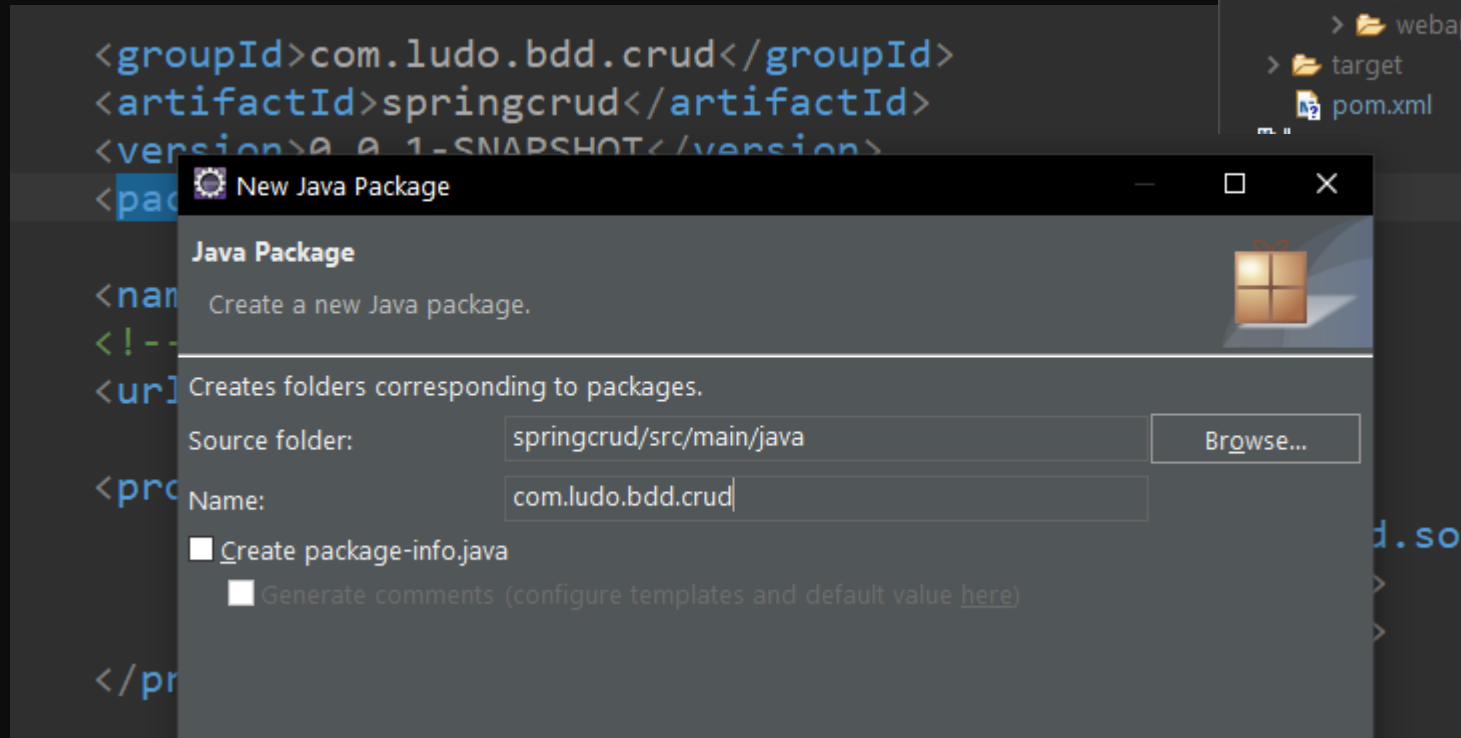
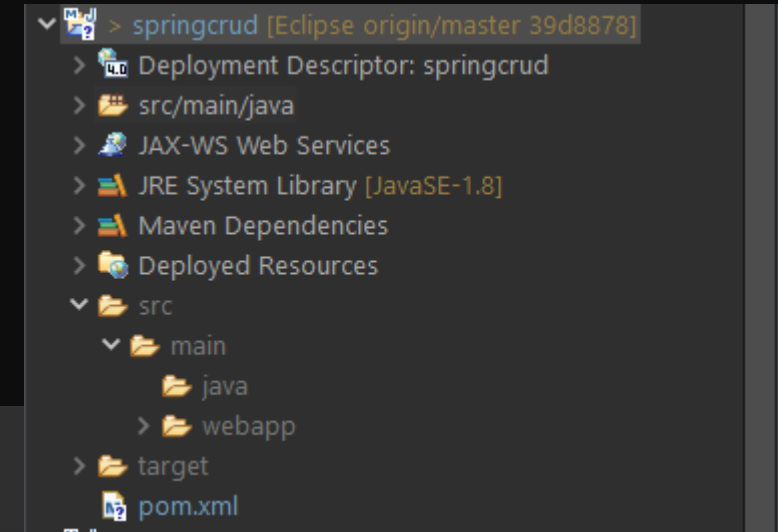
En esta parte:

- Inicializaremos la configuración del proyecto usando las clases `AbstractAnnotationConfigDispatcherServletInitializer` y `WebMvcConfigurerAdapter` de `springframework`.
- Crearemos la `sessionFactory` necesaria para realizar conexión a bases de datos.
- Configuraremos las variables de entorno ENV
- Crearemos la base de datos

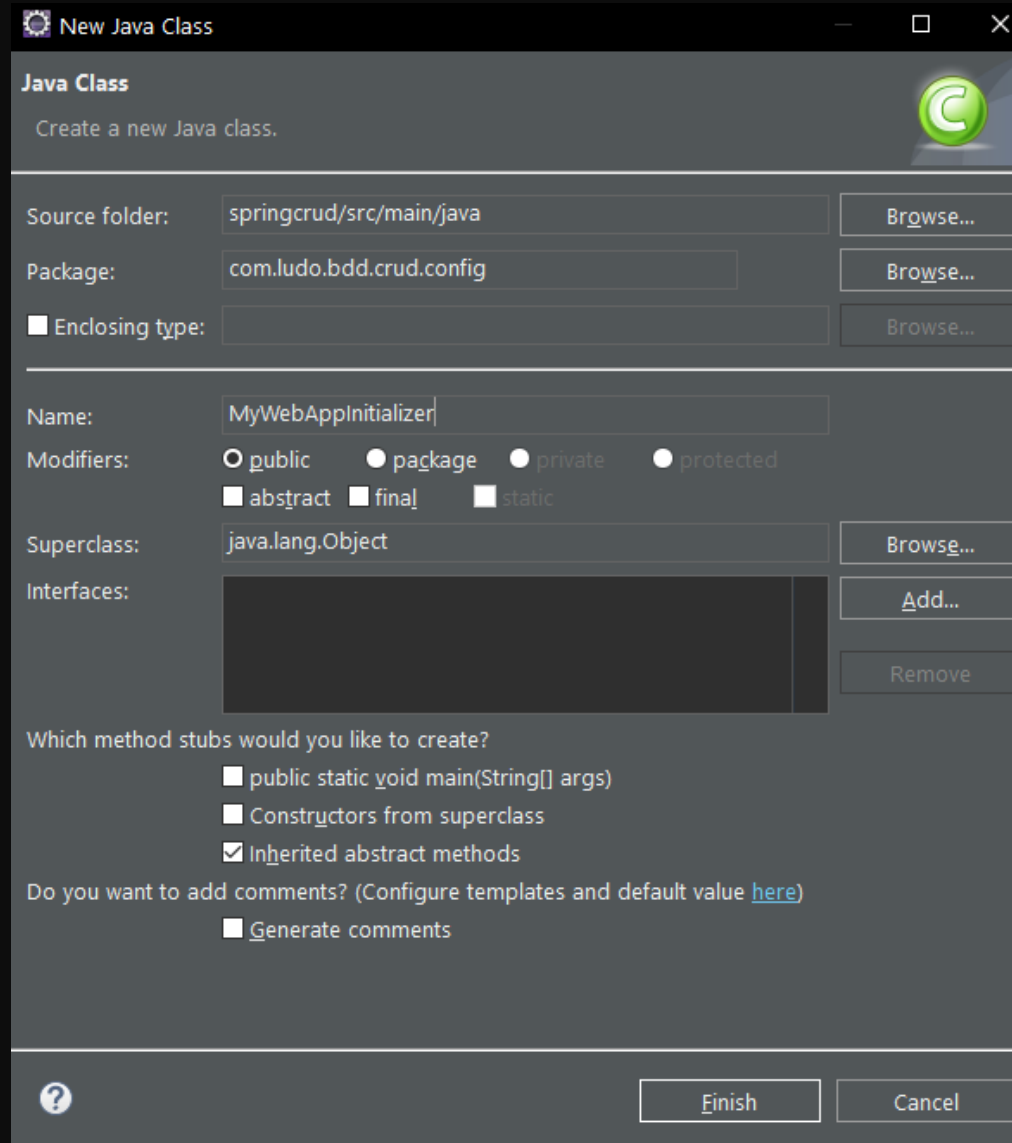


Creación de los paquetes parte JAVA

```
<groupId>com.ludo.bdd.crud</groupId>
<artifactId>springcrud</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
```



Creación de la clase inicializadora

A screenshot of the 'New Java Class' dialog box in an IDE. The dialog has a title bar with a gear icon and the text 'New Java Class'. Below the title bar is a section labeled 'Java Class' with a green 'C' icon and the text 'Create a new Java class.'.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☐ public ☒ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

--

Which method stubs would you like to create?

- ☐ public static void main(String[] args)
- ☐ Constructors from superclass
- ☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

- ☐ Generate comments



Aspecto inicial

```
springcrud/pom.xml  *MyWebAppInitializer.java ×
1 package com.ludo.bdd.crud.config;
2
3 import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
4
5 public class MyWebAppInitializer
6 extends AbstractAnnotationConfigDispatcherServletInitializer{
7
8 }
9 |
```



Sobrescribiendo métodos heredados

```
springcrud/pom.xml  *MyWebAppInitializer.java ×
1 package com.ludo.bdd.crud.config;
2
3 import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
4
5 public class MyWebAppInitializer
6 extends AbstractAnnotationConfigDispatcherServletInitializer{
7
8     @Override
9     protected Class<?>[] getRootConfigClasses() {
10         // TODO Auto-generated method stub
11         return null;
12     }
13
14     @Override
15     protected Class<?>[] getServletConfigClasses() {
16         // TODO Auto-generated method stub
17         return null;
18     }
19
20     @Override
21     protected String[] getServletMappings() {
22         // TODO Auto-generated method stub
23         return null;
24     }
25 }
```



Sobrescribiendo métodos heredados p2

```
springcrud/pom.xml  MyWebAppInitializer.java X
1 import org.springframework.web.servlet.support.AbstractAnnotationMethodDispatcher;
2
3
4
5 public class MyWebAppInitializer
6 extends AbstractAnnotationConfigDispatcherServletInitializer{
7
8     @Override
9     protected Class<?>[] getRootConfigClasses() {
10         // TODO Auto-generated method stub
11         return new Class[] { AppConfig.class };
12     }
13
14     @Override
15     protected Class<?>[] getServletConfigClasses() {
16         // TODO Auto-generated method stub
17         return new Class[] { WebConfig.class };
18     }
19
20     @Override
21     protected String[] getServletMappings() {
22         // TODO Auto-generated method stub
23         return new String[] { "/" };
24     }
25
26 }
```



Creando clase AppConfig

New

Java Class
Create a new Java class.

Source folder:

Package:

☒ Enclosing type:

Name:

Modifiers: ☐ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

- ☐ public static void main(String[] args)
- ☐ Constructors from superclass
- ☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

- ☐ Generate comments



Creando classe WebConfig

New

Java Class
Create a new Java class.

Source folder:

Package:

☒ Enclosing type:

Name:

Modifiers: ☐ public ☒ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

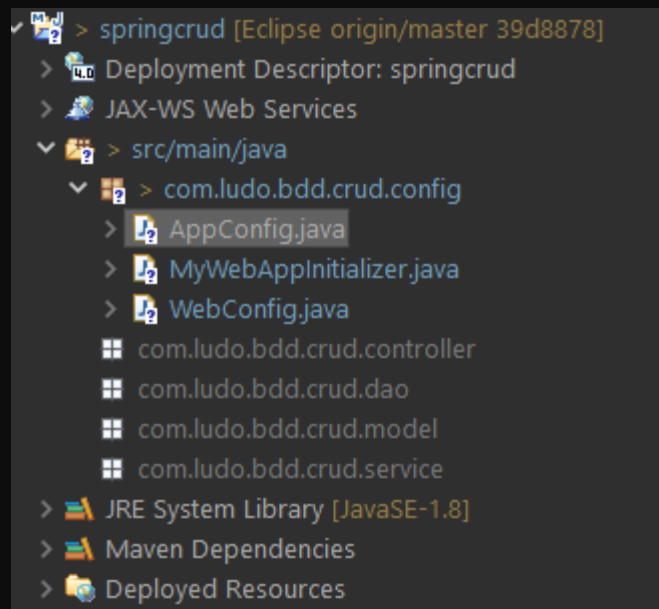
☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments



Aspecto inicial clase AppConfig



```
1 package com.ludo.bdd.crud.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.transaction.annotation.EnableTransactionManagement;
5
6 @Configuration
7 @EnableTransactionManagement
8 public class AppConfig {
9
10 }
11
```



Creación de la variable de entorno

```
import org.springframework.transaction.annotation.EnableTransactionManagement;
import org.springframework.core.env.Environment;

@Configuration
@EnableTransactionManagement
public class AppConfig {

    @Autowired
    private Environment env;

}
```



Añadiendo métodos para la sessionFactory

```
@Bean
public HibernateTransactionManager getTransactionManager() {
    HibernateTransactionManager transactionManager = new HibernateTransactionManager();
    transactionManager.setSessionFactory(getSessionFactory().getObject());
    return transactionManager;
}
```

```
@Bean
public LocalSessionFactoryBean getSessionFactory() {
    LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();
    return factoryBean;
}
```

Para integrar Hibernate con la aplicación Spring MVC, puedes usar la clase LocalSessionFactoryBean. Configura el objeto SessionFactory dentro de un contexto de aplicación Spring. Este objeto SessionFactory se puede pasar a clases DAO mediante la inyección de dependencias.



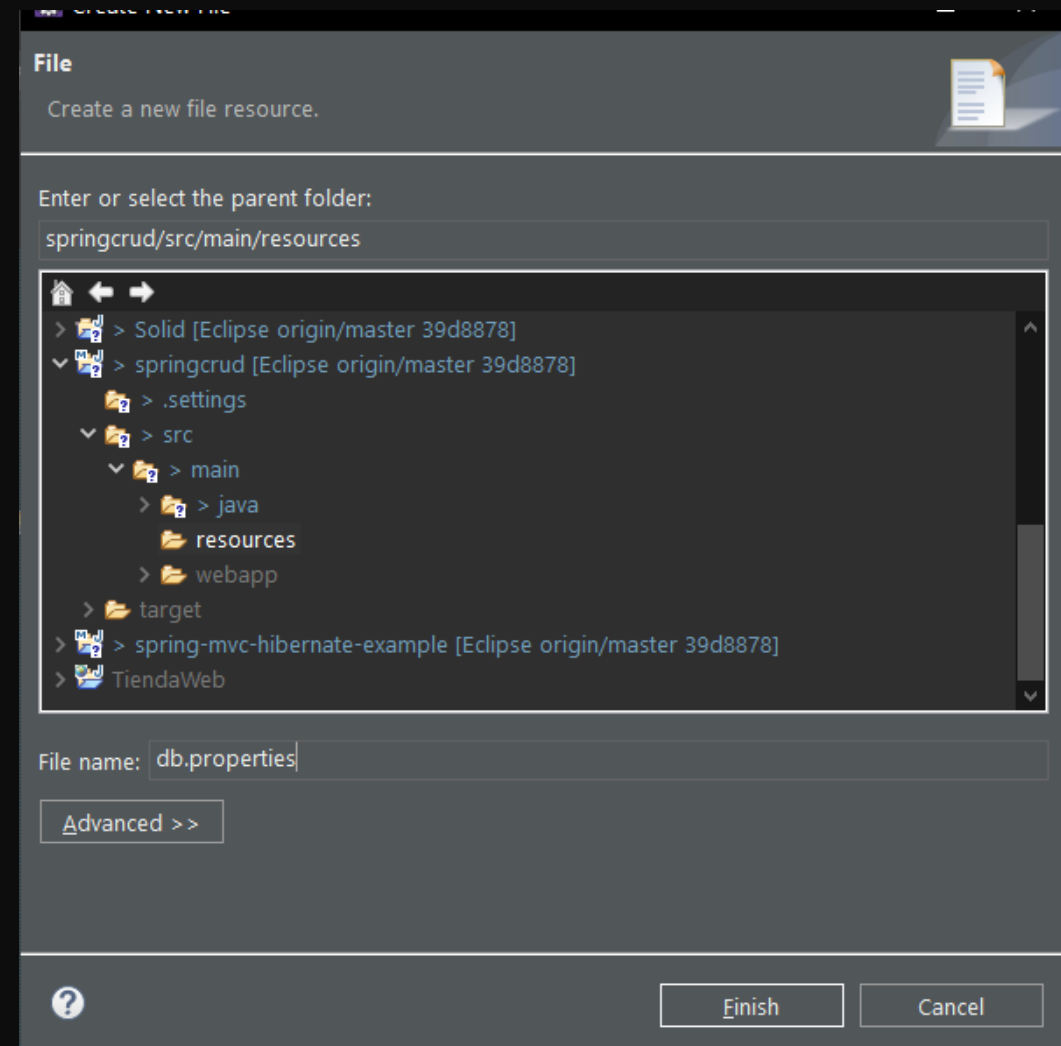
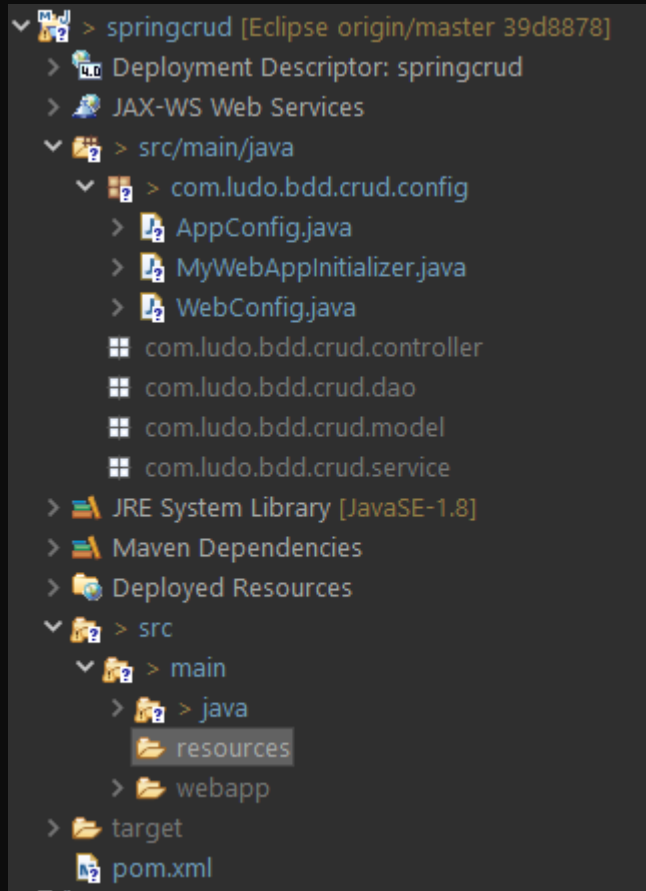
Asignando propiedades para la variable de entorno

```
Properties props = new Properties();  
factoryBean.setHibernateProperties(props);
```

```
@Configuration  
@PropertySource("classpath:db.properties")  
@EnableTransactionManagement  
public class AppConfig {
```



Creando la carpeta resources



Componiendo las variables de MySQL



```
1# MySQL properties
2mysql.driver=com.mysql.jdbc.Driver
3mysql.url=jdbc:mysql://localhost:3306/springbdd?useUnicode=true
4mysql.user=root
5mysql.password=
6
```

```
7# Hibernate properties
8hibernate.show_sql=true
9hibernate.hbm2ddl.auto=update
10
11#C3P0 properties
12hibernate.c3p0.min_size=5
13hibernate.c3p0.max_size=20
14hibernate.c3p0.acquire_increment=1
15hibernate.c3p0.timeout=1800
16hibernate.c3p0.max_statements=150
```




Creando la BDD

Bases de datos

 Crear base de datos 

Filtros

Que contengan la palabra:

Base de datos	Cotejamiento	Acción
<input type="checkbox"/> springbdd	utf8_bin	 Seleccionar privilegios



Importando nuevos paquetes de entorno

```
ud/pom.xml  MyWebAppInitializer.java  *AppConfig.java X
class AppConfig {

    @Autowired
    private Environment env;

    @Bean
    public LocalSessionFactoryBean getSessionFactory() {
        LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();

        Properties props = new Properties();

        // Setting JDBC properties
        props.put(DRIVER, env.getProperty("mysql.driver"));

        factoryBean.setHibernateProperties(props);

        return factoryBean;
    }
}
```



Importando nuevos paquetes de entorno p2

```
import org.springframework.core.env.Environment;
import static org.hibernate.cfg.Environment.*;

@Configuration
@PropertySource("classpath:db.properties")
@EnableTransactionManagement
public class AppConfig {

    @Autowired
    private Environment env;

    @Bean
    public LocalSessionFactoryBean getSessionFactory() {
        LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();

        Properties props = new Properties();

        // Setting JDBC properties
        props.put(DRIVER, env.getProperty("mysql.driver"));
    }
}
```



Equivalente al fichero hibernate.cfg.xml

```
AppConfig.java db.properties hibernate.cfg.xml ×
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN
3 <hibernate-configuration>
4   <session-factory>
5     <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
6     <property name="connection.url">jdbc:mysql://localhost:3306/biblioteca_2022?useSSL
7     <property name="connection.username">root</property>
8     <property name="connection.password"> </property>
9     <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
10    <property name="hibernate.show_sql">true</property>
11
12  </session-factory>
13 </hibernate-configuration>
```



Metiendo el conjunto de propiedades

```
// Setting JDBC properties
props.put(DRIVER, env.getProperty("mysql.driver"));
props.put(URL, env.getProperty("mysql.url"));
props.put(USER, env.getProperty("mysql.user"));
props.put(PASS, env.getProperty("mysql.password"));

// Setting Hibernate properties
props.put(SHOW_SQL, env.getProperty("hibernate.show_sql"));
props.put(HBM2DDL_AUTO, env.getProperty("hibernate.hbm2ddl.auto"));

// Setting C3P0 properties
props.put(C3P0_MIN_SIZE,
    env.getProperty("hibernate.c3p0.min_size"));
props.put(C3P0_MAX_SIZE,
    env.getProperty("hibernate.c3p0.max_size"));
props.put(C3P0_ACQUIRE_INCREMENT,
    env.getProperty("hibernate.c3p0.acquire_increment"));
props.put(C3P0_TIMEOUT,
    env.getProperty("hibernate.c3p0.timeout"));
props.put(C3P0_MAX_STATEMENTS,
    env.getProperty("hibernate.c3p0.max_statements"));|
```



Correlación variables MySQL con sus valores

```
// Setting JDBC properties
props.put(DRIVER, env.getProperty("mysql.driver"));
props.put(URL, env.getProperty("mysql.url"));
props.put(USER, env.getProperty("mysql.user"));
props.put(PASS, env.getProperty("mysql.password"));
```

```
1# MySQL properties
2mysql.driver=com.mysql.jdbc.Driver
3mysql.url=jdbc:mysql://localhost:3306/springbdd?useUnicode=true
4mysql.user=root
5mysql.password=
6
```



Correlación variables Hibernate con sus valores

```
// Setting Hibernate properties  
props.put(SHOW_SQL, env.getProperty("hibernate.show_sql"));  
props.put(HBM2DDL_AUTO, env.getProperty("hibernate.hbm2ddl.auto"));
```

```
7 # Hibernate properties  
8 hibernate.show_sql=true  
9 hibernate.hbm2ddl.auto=update  
0
```



Correlación variables C3P0 con sus valores

```
// Setting C3P0 properties
props.put(C3P0_MIN_SIZE, env.getProperty("hibernate.c3p0.min_size"));
props.put(C3P0_MAX_SIZE, env.getProperty("hibernate.c3p0.max_size"));
props.put(C3P0_ACQUIRE_INCREMENT, env.getProperty("hibernate.c3p0.acquire_increment"));
props.put(C3P0_TIMEOUT, env.getProperty("hibernate.c3p0.timeout"));
props.put(C3P0_MAX_STATEMENTS, env.getProperty("hibernate.c3p0.max_statements"));
```

```
#C3P0 properties
hibernate.c3p0.min_size=5
hibernate.c3p0.max_size=20
hibernate.c3p0.acquire_increment=1
hibernate.c3p0.timeout=1800
hibernate.c3p0.max_statements=150
```



Acabando de configurar AppConfig

```
factoryBean.setHibernateProperties(props);  
factoryBean.setAnnotatedClasses(User.class);  
return factoryBean;  
}
```



Creación del modelo User

En pocos pasos, crearemos el modelo User y los mensajes de validación.



Creación de la clase User

New

Java Class
Create a new Java class.

Source folder:

Package:

Enclosing type:

Name:

Modifiers: ☐ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

```
springcrud/pom.xml  MyWebAppInitializer.java  AppConfig.java  *User.java X
1 package com.ludo.bdd.crud.model;
2
3 public class User {
4
5     private Long id;
6     private String name;
7     private String email;
8
9     public User() {
10    }
11
12
13    public User(Long id, String name, String email) {
14        this.setId(id);
15        this.setName(name);
16        this.setEmail(email);
17    }
18
19    //GETTERS + SETTERS
20
```



Creación de la clase User

Crea una clase `@Entity`, cuyos nombres de campo estén anotados con la restricción del validador de hibernación y las anotaciones JPA.

Utilizaremos esta clase de entidad para asignar la tabla de la base de datos con `User`. También se usará para vincular los datos al modelo usando la anotación `@ModelAttribute` en el método del controlador.

```
springcrud/pom.xml  MyWebAppInitializer.java  AppConfig.java  *User.java X
1 package com.ludo.bdd.crud.model;
2
3 public class User {
4
5     private Long id;
6     private String name;
7     private String email;
8
9     public User() {
10    }
11
12
13    public User(Long id, String name, String email) {
14        this.setId(id);
15        this.setName(name);
16        this.setEmail(email);
17    }
18
19    //GETTERS + SETTERS
20
```



Configurar ORM y reglas de validación

Añadimos la anotación `@Entity` al nivel del nombre de la clase y el nombre de cada campo de la tabla al nivel de la propiedad de la clase.

Podemos añadir restricciones del Validator de Hibernate y de la configuración de las tablas.

Utilizaremos esta clase de entidad para asignar la tabla de la base de datos con `User`. También se usará para vincular los datos al modelo usando la anotación `@ModelAttribute` en el método del controlador adecuado.

```
@Entity
@Table(name = "USER_TBL")
public class User {

    @Id
    @GeneratedValue
    @Column(name = "id")
    private Long id;

    @Column(name = "name")
    @Size(min = 3, max = 20, message = "{user.name.invalid}")
    private String name;

    @Column(name = "email", unique = true)
    @Email(message = "{user.email.invalid}")
    private String email;

    public User() {
    }
}
```



Añadiendo mensajes de validación adecuados

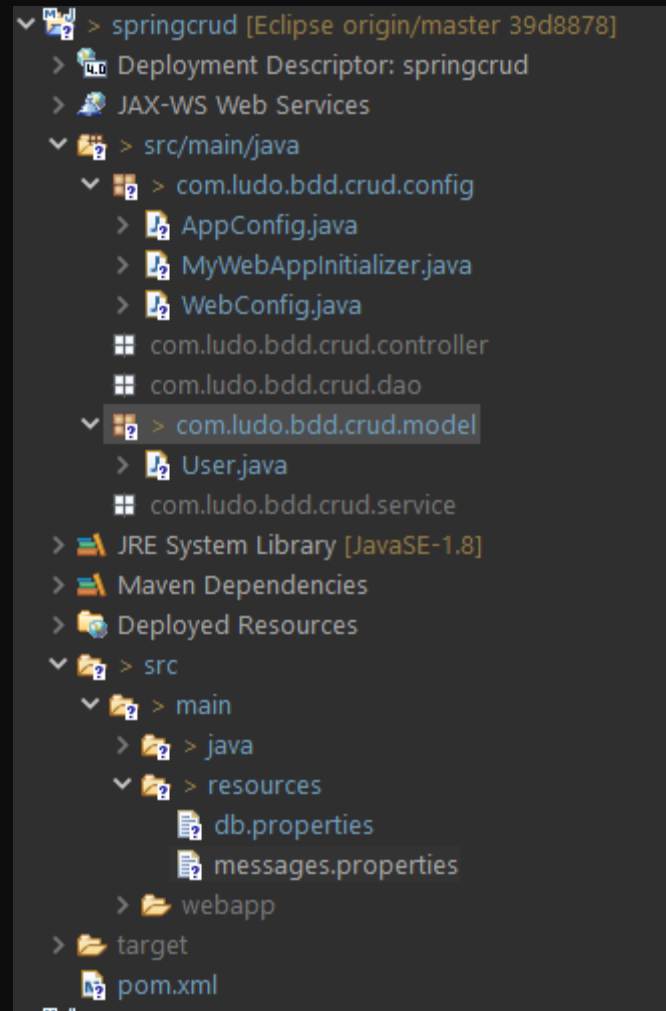
En este ejemplo, estamos utilizando las anotaciones de restricciones del validador de hibernate para la validación del formulario de usuario.

Para anular los mensajes de error predeterminados, proporcionados por Hibernate Validator framework, cree un archivo `messages.properties` en la carpeta `src/main/resources`.

```
1user.name.invalid = Nombre ingresado invalido. Debe de contener entre {min} and {max} caracteres.  
2user.email.invalid = ¡Email invalido! Por favor, ingreso un email valido.
```



Localización del fichero de mensajes en el proyecto



Configuración del proyecto sin XML

En esta parte vemos como configurar el proyecto sin XML.



Estado inicial del fichero WebConfig

```
MyWebAppInitializer.java  AppConfig.java  WebConfig.java X
1 package com.ludo.bdd.crud.config;
2
3 public class WebConfig {
4
5 }
6
```



Añadiendo anotaciones y estableciendo herencia

```
MyWebAppInitializer.java  AppConfig.java  *WebConfig.java X
1 package com.ludo.bdd.crud.config;
2
3 import org.springframework.context.annotation.ComponentScan;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.servlet.config.annotation.EnableWebMvc;
6 import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
7
8 @Configuration
9 @EnableWebMvc
10 @ComponentScan("com.ludo.bdd.crud.controller")
11 public class WebConfig extends WebMvcConfigurerAdapter {
12
13 }
14
```



Creación del método para la validación

```
@Configuration
@EnableWebMvc
@ComponentScan("com.ludo.bdd.crud.controller")
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public Validator getValidator() {
        LocalValidatorFactoryBean validator = new LocalValidatorFactoryBean();
        validator.setValidationMessageSource(messageSource());
        return validator;
    }
}
```



Creación de los métodos para la validación

```
@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource source = new ResourceBundleMessageSource();
    source.setBasename("messages");
    return source;
}

@Override
public Validator getValidator() {
    LocalValidatorFactoryBean validator = new LocalValidatorFactoryBean();
    validator.setValidationMessageSource(messageSource());
    return validator;
}
```

Equivalente a

```
/*
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="messages"/>
</bean>
*/
```



Creación de los métodos para la validación

```
@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource source = new ResourceBundleMessageSource();
    source.setBasename("messages");
    return source;
}

@Override
public Validator getValidator() {
    LocalValidatorFactoryBean validator = new LocalValidatorFactoryBean();
    validator.setValidationMessageSource(messageSource());
    return validator;
}
```

Sobrescribimos el método `getValidator()` de `WebMvcConfigurerAdapter` para establecer los mensajes de error personalizados para el formulario de usuario.



Creando el método resolver()

```
@Bean
public InternalResourceViewResolver resolver() {
    InternalResourceViewResolver resolver = new InternalResourceViewResolver();
    resolver.setViewClass(JstlView.class);
    resolver.setPrefix("/WEB-INF/views/");
    resolver.setSuffix(".jsp");
    return resolver;
}
```

```
/*
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
</bean>
*/
```

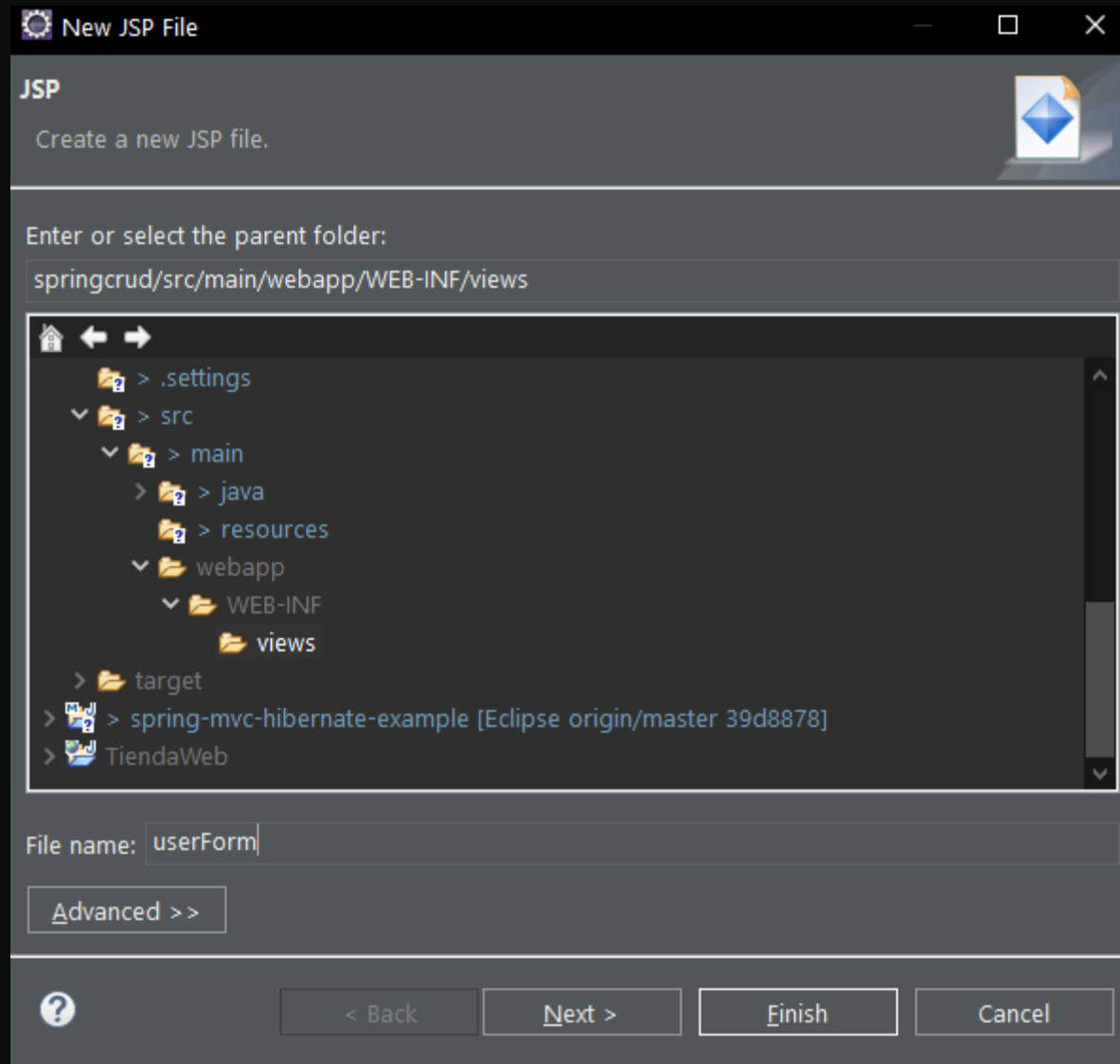


Creación del formulario User

En pocos pasos, crearemos el formulario de creación de usuario en JSP.



Creación y configuración del fichero JSP



Parte Form

```
<fieldset>
  <legend>Formulario para ingresar usuarios</legend>
  <form:form action="saveUser" method="post" modelAttribute="user">
    <table>
      <tr>
        <th>Nombre</th>
        <td>
          <form:input path="name" />
          <form:errors path="name" cssClass="error" />
        </td>
        <th>Email</th>
        <td>
          <form:input path="email" />
          <form:errors path="email" cssClass="error" />
        </td>
        <td><button type="submit">Validar</button></td>
      </tr>
    </table>
  </form:form>
</fieldset>
```



Parte lista

```
<fieldset>
  <legend>Lista de usuarios</legend>
  <table class="resltTable">
    <tr>
      <th>Nombre</th>
      <th>Email</th>
    </tr>
    <c:forEach items="${users}" var="user">
      <tr>
        <td>${user.name}</td>
        <td>${user.email}</td>
      </tr>
    </c:forEach>
  </table>
</fieldset>
```



Configuración del controlador

En esta parte, crearemos y configuremos el controlador que nos va a servir para conectar el JSP del userForm a los servicios de validación y al objeto de acceso a datos (DAO)



Creación del controlador

```
|
@Controller
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/")
    public String userForm(Locale locale, Model model) {

        model.addAttribute("user", new User());
        model.addAttribute("users", userService.list());

        return "userForm";
    }
}
```



Qué es @Service?

Spring @Service es una de las anotaciones más habituales de Spring Framework . Se usa para construir una clase de Servicio que habitualmente se conecta a varios repositorios y agrupa su funcionalidad. Es decir por ejemplo si disponemos de dos Repositorios uno con Profesores y otro con Alumnos es muy común disponer de una clase Fachada de tipo ServicioCurso que aglutine la funcionalidad de las dos capas de Repositorio.



Acerca del patrón Servicio

El patrón Servicio nos sirve para añadir una lista de objetos a una vista. Este patrón de diseño hace las funciones de pivot y aglutina un conjunto de métodos que ya se encuentran en las clases de Repositorio. Es decir muchas veces simplemente realiza unas tareas de delegación. Seguramente algunas persona me dirán que no ven porqué hacerlo así. Preferirán no usar el patrón de Servicio y apoyarse directamente en el Repositorio por temas de simplicidad. Es verdad: es posible y factible. Y eso hará que el Controlador y el Repositorio trabajen de forma directa entre ellos. Pero ¿es correcto?



Creando un servicio para poder alterar propiedades

```
WebConfig.java  userForm.jsp  *UserController.java  UserService.java X
1 package com.ludo.bdd.crud.service;
2
3 import java.util.List;
4
5 import com.ludo.bdd.crud.model.User;
6
7 public interface UserService {
8     void save(User user);
9
10    List<User> list();
11 }
12
```



Implementando la interfaz

```
public class UserServiceImpl implements UserService {  
  
    @Autowired  
    private UserDao userDao;  
  
    @Override  
    @Transactional  
    public void save(User user) {  
        userDao.save(user);  
    }  
  
    @Override  
    @Transactional  
    public List<User> list() {  
        return userDao.list();  
    }  
}
```



Realizando cambios según nuestra lógica de negocio

```
@Override
@Transactional
public void save(User user) {
    User copiaUser = new User();
    copiaUser.setEmail(user.getEmail().toLowerCase());
    copiaUser.setName(user.getName().toUpperCase());
    copiaUser.setId(user.getId());
    userDao.save(copiaUser);
}
```



Mi respuesta:

La anotación @Service

En una aplicación, la lógica de negocio reside dentro de la capa de servicio, por lo que utilizamos @Service para indicar que una clase pertenece a esa capa. También es una especialización de @Component al igual que @Repository.

Una de las cosas más importantes de la anotación @Service es que solo se puede aplicar a clases. No a métodos, ni a propiedades...

Se usa para marcar la clase como proveedor de servicios. Por lo tanto, la usamos para realizar funciones específicas: Pasar a minúsculas un email, a mayúsculas un apellido, sacar IVAs, circunferencias de una forma con la clase Math, convertir fechas al formato local...

Hacer esto en las capas del controlador o del repositorio iría entonces en contra de los principios SOLID



Creando la relación entre la BDD y la clase

```
1 package com.ludo.bdd.crud.dao;
2
3 import java.util.List;
4
5 import com.ludo.bdd.crud.model.User;
6
7 public interface UserDao {
8     void save(User user);
9
10    List<User> list();
11 }
12 |
```



Implementando la interfaz

```
import com.ludo.bdd.crud.model.User;|

public class UserDaoImpl implements UserDao {

    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public void save(User user) {
        sessionFactory.getCurrentSession().save(user);
    }

    @Override
    public List<User> list() {
        @SuppressWarnings("unchecked")
        TypedQuery<User> query = sessionFactory.getCurrentSession().createQuery("from User");
        return query.getResultList();
    }
}
```



Se están quitando los errores

```
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;

    @Override
    @Transactional
    public void save(User user) {
        userDao.save(user);
    }

    @Override
    @Transactional
    public List<User> list() {
        return userDao.list();
    }
}
```



Se están quitando los errores p2

```
@Controller
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/")
    public String userForm(Locale locale, Model model) {

        model.addAttribute("user", new User());
        model.addAttribute("users", userService.list());

        return "userForm";
    }
}
```



Acabando de configurar AppConfig

```
@Configuration
@PropertySource("classpath:db.properties")
@EnableTransactionManagement
@ComponentScans(value = { @ComponentScan("com.ludo.bdd.crud.dao"),
                          @ComponentScan("com.ludo.bdd.crud.service") })
public class AppConfig {
```



Añadiendo método que controla el FORM

En el Controlador

```
@PostMapping("/saveUser")
public String saveUser(@ModelAttribute("user") @Valid User user,
    BindingResult result, Model model) {

    if (result.hasErrors()) {

        model.addAttribute("users", userService.list());
        return "userForm";
    }

    userService.save(user);

    return "redirect:/";
}
```



Solucionar el estado obsoleto de la clase WebMvcConfigurerAdapter

```
@Configuration  
@EnableWebMvc  
@ComponentScan("com.ludo.bdd.crud.controller")  
public class WebConfig implements WebMvcConfigurer {
```



Probando

← → ↻ 🏠 localhost:8080/springcrud/

📁 D 📁 Jeu 📁 LL 📁 Foot C 📁 W N 📁 YouTube G L1 G FF 📁 20 FS G

FORMULARIO PARA INGRESAR USUARIOS

Nombre **Email**

LISTA DE USUARIOS

Nombre	Email
Ludovic	ludoviclaisnez@gmail.com



Borrar y actualizar





Borrar y actualizar



Borrar registro

1

En esta parte, configuraremos los ficheros para poder borrar registros.



Actualizar el JSP añadiendo un botón Borrar

```
<c:forEach items="${users}" var="user">
  <c:url var="delete" value="/deleteUser">
    <c:param name="id" value="${user.id}" />
  </c:url>
  <tr>
    <td>${user.name}</td>
    <td>${user.email}</td>
    <td><a href="${delete}"
      title="Borrar ${user.name} con id ${user.id}"
      <button type="submit" class="btn btn-danger"
onclick="if(!(confirm('¿Seguro que quieres eliminar el registro?'))) return false">Borrar</button>
    </a></td>
  </tr>
</c:forEach>
```



Actualizando el controlador y las interfaces

```
@GetMapping("/deleteUser")
public String deleteUser(@RequestParam("id") long id) {
    userService.delete(id);
    return "redirect:/";
}
```

```
UserController.java  UserService.java  UserServiceImpl.java  UserDao.java X
1 package com.ludo.bdd.crud.dao;
2
3 import java.util.List;
4
5
6
7 public interface UserDao {
8     void save(User user);
9
10    List<User> list();
11
12    void delete(long id);
13
14
15 }
16
```

```
UserController.java  UserService.java X
1 package com.ludo.bdd.crud.service;
2
3 import java.util.List;
4
5
6
7 public interface UserService {
8     void save(User user);
9
10    List<User> list();
11
12    void delete(long id);
13 }
14
```



Actualización del userServiceImpl

```
UserController.java  UserService.java  UserServiceImpl.java x
18  @Override
19  @Transactional
20  public void save(User user) {
21      userDao.save(user);
22  }
23
24  @Override
25  @Transactional
26  public List<User> list() {
27      return userDao.list();
28  }
29
30  @Override
31  @Transactional
32  public void delete(long id) {
33      userDao.delete(id);
34  }
35
36 }
```



Actualización del userDaoImpl

```
@Override
public void delete(long id) {
    System.out.println(id);
    Query query = sessionFactory.getCurrentSession().createQuery("delete from User where id =:id");
    query.setParameter("id", id);
    query.executeUpdate();
}
```



Prueba

FORMULARIO PARA INGRESAR USUARIOS

Nombre Email

LISTA DE USUARIOS

Nombre	Email	Borrar
ludovic	ludoviclaisnez@gmail.com	<input type="button" value="Borrar"/>
Marcel	marcel@gmail.com	<input type="button" value="Borrar"/>
Gorka	gorka@gmail.com	<input type="button" value="Borrar"/>



Prueba p2

FORMULARIO PARA INGRESAR USUARIOS

Nombre Email

LISTA DE USUARIOS

Nombre	Email	Borrar
ludovic	ludoviclaisnez@gmail.com	<input type="button" value="Borrar"/>
Marcel	marcel@gmail.com	<input type="button" value="Borrar"/>
Gorka	gorka@gmail.com	<input type="button" value="Borrar"/>

🌐 localhost:8080

¿Seguro que quieres eliminar el registro?



Prueba p3

FORMULARIO PARA INGRESAR USUARIOS

Nombre Email

LISTA DE USUARIOS

Nombre	Email	Borrar
ludovic	ludoviclaisnez@gmail.com	<input type="button" value="Borrar"/>
Gorka	gorka@gmail.com	<input type="button" value="Borrar"/>



Borrar registro

En esta parte, configuraremos los ficheros para poder actualizar registros. Usaremos campos Hidden en el JSP y modificaremos métodos ya escritos.



Actualizando el JSP parte form

```
<fieldset>
  <legend>Formulario para ingresar usuarios</legend>
  <form:form action="saveUser" method="post" modelAttribute="user">
    <table>
      <tr>
        <th>Nombre</th>
        <td><form:input path="name" /> <form:errors path="name"
          cssClass="error" /></td> <form:hidden path="id"/>
        <th>Email</th>
        <td><form:input path="email" /> <form:errors path="email"
          cssClass="error" /></td>
        <td><button type="submit">Validar</button></td>
      </tr>
    </table>
  </form:form>
</fieldset>
```



Añadiendo el botón Modificar al listado

```
<c:forEach items="${users}" var="user">
  <c:url var="delete" value="/deleteUser">
    <c:param name="id" value="${user.id}" />
  </c:url>
  <c:url var="edit" value="/editUser">
    <c:param name="id" value="${user.id}" />
  </c:url>
  <tr>
    <td>${user.name}</td>
    <td>${user.email}</td>
    <td><a href="${edit}"
      title="Actualizar ${user.name} con id ${user.id}">
      <button type="submit" class="btn btn-success">Modificar</button>
    </a></td>
    <td><a href="${delete}"
      title="Borrar ${user.name} con id ${user.id}">
      <button type="submit" class="btn btn-danger"
        onclick="if(!confirm('¿Seguro que quieres eliminar el registro?
      </a></td>
  </tr>
```



Actualizando el controlador y las interfaces

```
@GetMapping("/editUser")
public String editUser(@RequestParam("id") int id, Model model) {
    User user = userService.get(id);
    model.addAttribute(user);
    model.addAttribute("users", userService.list());
    return "userForm";
}
```

```
userForm.jsp × UserController.java UserService.java × Us
1 package com.ludo.bdd.crud.service;
2
3 import java.util.List;
4
5
6
7 public interface UserService {
8     void save(User user);
9
10    List<User> list();
11
12    void delete(long id);
13
14    User get(long id);
15 }
16
```

```
1 package com.ludo.bdd.crud.dao;
2
3 import java.util.List;
4
5
6
7 public interface UserDao {
8     void save(User user);
9
10    List<User> list();
11
12    void delete(long id);
13
14    User get(long id);
15
16
17 }
18
```



Modificando el UserServiceImpl

```
userForm.jsp  UserController.java  UserService.java  UserServiceImpl.java
20 public void save(User user) {
21     userDao.save(user);
22 }
23
24 @Override
25 @Transactional
26 public List<User> list() {
27     return userDao.list();
28 }
29
30 @Override
31 @Transactional
32 public void delete(long id) {
33     userDao.delete(id);
34 }
35
36 @Override
37 @Transactional
38 public User get(long id) {
39     User user = userDao.get(id);
40     return user;
41 }
42
43 }
```



Modificando el UserDaoImpl

```
@Override
public User get(long id) {
    // TODO Auto-generated method stub
    Session session = sessionFactory.getCurrentSession();
    User user = session.get(User.class, id);
    //System.out.println("GET" + user);
    return user;
}
```

```
@Override
public void save(User user) {
    System.out.println("SaveOrUpdate" + user);
    sessionFactory.getCurrentSession().saveOrUpdate(user);
}
```



Probando

FORMULARIO PARA INGRESAR USUARIOS

Nombre **Email**

LISTA DE USUARIOS

Nombre	Email	Actualizar	Borrar
ludovick	ludoviclaisnez@gmail.com	<input type="button" value="Modificar"/>	<input type="button" value="Borrar"/>
Marcel	marcel@gmail.com	<input type="button" value="Modificar"/>	<input type="button" value="Borrar"/>
Gorka	gorka@gmail.com	<input type="button" value="Modificar"/>	<input type="button" value="Borrar"/>

FORMULARIO PARA INGRESAR USUARIOS

Nombre **Email**



Probando p2

FORMULARIO PARA INGRESAR USUARIOS

Nombre Email

LISTA DE USUARIOS

Nombre	Email	Actualizar	Borrar
ludovic	ludoviclaisnez@gmail.com	<input type="button" value="Modificar"/>	<input type="button" value="Borrar"/>
Marcel	marcel@gmail.com	<input type="button" value="Modificar"/>	<input type="button" value="Borrar"/>
Gorka	gorka@gmail.com	<input type="button" value="Modificar"/>	<input type="button" value="Borrar"/>



Mejorando aspectos





Mejorando aspectos

Contabilizar los registros

En esta parte, crearemos un método para contar los registros y avisar al usuario de sí existen registros o no al cargar el JSP



Modificamos al JSP userForm

```
<c:if test="${how_many > 0}" >
<c:forEach items="${users}" var="user">
  <c:url var="delete" value="/deleteUser">
    <c:param name="id" value="${user.id}" />
  </c:url>
  <c:url var="edit" value="/editUser">
    <c:param name="id" value="${user.id}" />
  </c:url>
  <tr>
    <td>${user.name}</td>
    <td>${user.email}</td>
    <td><a href="${edit}"
      title="Actualizar ${user.name} con id ${user.id}">
      <button type="submit" class="btn btn-success">Modificar</button>
    </a></td>
    <td><a href="${delete}"
      title="Borrar ${user.name} con id ${user.id}">
      <button type="submit" class="btn btn-danger"
        onclick="if(!(confirm('¿Seguro que quieres eliminar el regi:
    </a></td>
  </tr>
</c:forEach>
</c:if>
```

Si hay registros, los imprimimos en pantalla



Modificamos el JSP userForm p2

Añadimos un condicional IF para avisar el visitante en caso de no tener registros

```
<c:if test="${how_many == 0}" >  
<tr><td colspan="4">La tabla no contiene registros</td></tr>  
</c:if>
```



Modificando el controlador

```
@GetMapping("/")
public String userForm(Locale locale, Model model) {

    model.addAttribute("user", new User());
    addAttributes(model);
    return "userForm";
}

@GetMapping("/editUser")
public String editUser(@RequestParam("id") int id, Model model) {
    User user = userService.get(id);
    model.addAttribute(user);
    addAttributes(model);
    return "userForm";
}

private void addAttributes(Model model) {
    model.addAttribute("users", userService.list());
    model.addAttribute("how_many", userService.num());
}
```

Necesitamos modificar el controlador para añadir el atributo "how_many"



Modificando las interfaces

```
package com.ludo.bdd.crud.service;

import java.util.List;

public interface UserService {
    void save(User user);

    List<User> list();

    void delete(long id);

    User get(long id);

    long num();
}
```

```
1 package com.ludo.bdd.crud.dao;
2
3 import java.util.List;
4
5
6
7 public interface UserDao {
8     void save(User user);
9
10    List<User> list();
11
12    void delete(long id);
13
14    User get(long id);
15
16    long num();
17
18 }
```

Este atributo se consigue calculando el número de registros en la tabla de la base de datos.



Modificando userDaoImpl y userServiceImpl

```
@Override
@Transactional
public long num() {
    return userDao.num();
}
```

Modificamos entonces los ficheros para obtener el número de registros en una tabla en concreto. Aquí usamos HQL. Por ese motivo escribimos User y no USER_TBL. Ya que hacemos referencia a la clase User. No a la tabla de la base de datos.

```
@Override
public long num() {
    long count = ((long) sessionFactory.getCurrentSession()
        .createQuery("SELECT COUNT(*) FROM User")
        .uniqueResult());
    return count;
}
```



En caso de no tener usuarios

FORMULARIO PARA INGRESAR USUARIOS

Nombre Email

LISTA DE USUARIOS

Nombre	Email	Actualizar	Borrar
La tabla no contiene registros			



En caso de tener por lo menos 1

FORMULARIO PARA INGRESAR USUARIOS

Nombre Email

LISTA DE USUARIOS

Nombre	Email	Actualizar	Borrar
Ludovic	ludoviclaisnez@gmail.com	<input type="button" value="Modificar"/>	<input type="button" value="Borrar"/>



Añadir hojas de estilo y recursos externos

En esta parte, haremos pequeños cambios para mejorar el aspecto básico de nuestro proyecto.



Desarrollando el proyecto

Aquí lo que pretendo es añadir más formularios y una parte estática a la web. Para conseguirlo empiezo por añadir los JSP y las plantillas.

springcrud > src > main > webapp > WEB-INF > view	
Nombre	Fecha de modificación
templates	08/06/2022 18:57
bookForm	08/06/2022 20:13
contacto	07/06/2022 17:58
index	08/06/2022 18:58
userForm	08/06/2022 19:47

src > main > webapp > WEB-INF > view > templates	
Nombre	Fecha de modificación
footer	01/06/2022
head	08/06/2022
nav	08/06/2022



Añadiendo Bootstrap

Después de haber añadido las rutas hacia los ficheros Bootstrap.css y Bootstrap.js, puedo utilizar las reglas de estilo que viene definido con esta biblioteca para diseño.

```
<head>|
<jsp:include page="/WEB-INF/view/templates/head.jsp"></jsp:include>
</head>
<body class="d-flex flex-column h-100 text-center">
  <jsp:include page="/WEB-INF/view/templates/nav.jsp"></jsp:include>
  <div class="flex-shrink-0">
    <div class="container">
      <div class="row mt-4">
        <div class="col">
          <fieldset>
```



Utilizando Bootstrap

Los formularios, los botones, los carruseles... Bootstrap nos ofrece muchas ideas de diseño.

```
<tr>
  <td>${user.name}</td>
  <td>${user.email}</td>
  <td><a href="${edit}"
    title="Actualizar ${user.name} con id ${user.id}">
    <button type="submit" class="btn btn-success">Modificar</button>
  </a></td>
  <td><a href="${delete}"
    title="Borrar ${user.name} con id ${user.id}">
    <button type="submit" class="btn btn-danger"
      onclick="if(!(confirm('¿Seguro que quieres eliminar el registro?')) r
    </a></td>
</tr>
```



Recursos, imágenes, estilos...

Lo único que no debo olvidar hacer es dar de alta la localización de la carpeta de recursos o de soportes para que Spring sepa dónde se encuentran mis ficheros de estilo (CSS), JavaScript (JS) y mis imágenes. Esto se indica en el fichero WebConfig.java

```
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
    registry.addResourceHandler("/soportes/**").addResourceLocations("/soportes/");
}
```



Añadir plantillas

En esta parte, añadiremos plantillas a nuestro código JSP.



Utilizando jsp:include

Usando JSP:INCLUDE, reduzco drásticamente los contenidos de mis ficheros JSP y los haga más fáciles de mantener.

```
<html>
<head>
<jsp:include page="/WEB-INF/view/templates/head.jsp"></jsp:include>
</head>
<body class="d-flex flex-column h-100 text-center">
  <jsp:include page="/WEB-INF/view/templates/nav.jsp"></jsp:include>
  <div class="flex-shrink-0">
    <div class="container">
      <div class="row">
        <div class="col-12">
          <h1 class="text-center">${titulo}</h1>
          <p class="text-center">${descripcion}</p>
        </div>
      </div>
    </div>
  </div>
  <jsp:include page="/WEB-INF/view/templates/footer.jsp">
    <jsp:param name="web" value="campus2b.com" />
  </jsp:include>
</body>
```



Funcionamiento de @ComponentScan

En esta parte, vemos como dar de alta los fichero JSP y las rutas URI



@ComponentScan

Puedo añadir más rutas a mis controladores y más controladores a mi proyecto. Pues mi fichero pide a Spring que los indexe con @ComponentScan

```
@RequestMapping("/")
public ModelAndView prepareView() {
    ModelAndView mv = new ModelAndView();
    mv.addObject("titulo", "Campus C2B");
    mv.addObject("descripcion", "Pruebas de acceso a BDD en C2B.");
    mv.setViewName("index");
    return mv;
}

@RequestMapping("/contacto")
public String contacto() {
    return "contacto";
}
```

```
@Configuration
@EnableWebMvc
@ComponentScan({"com.ludo.bdd.crud.controller", "com.ludo.bdd.crud.validator"})
public class WebConfig implements WebMvcConfigurer {
```



@Bean

Otra forma de hacerlo es crear Beans:

El hecho de añadir @Configuration a los ficheros de configuración provoca que Spring se ponga a escanear los métodos de este fichero en busca de anotaciones @Bean.

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {
    /**
        @Bean
        public MainController myMvcController() {
            return new MainController();
        }
    }
```

Si encuentra uno o varios, asocia el nombre del método a una clase y esperará encontrar una anotación de tipo @Component. También le valdrá @Controller, @Repository, @Service. Al fin y al cabo, son componentes especializados en algún tipo de función:

Ofrecer un servicio, Conectar a bases de datos o registrar un Controlador.

```
@Controller
public class MainController {
```

```
@Service
public class BookServiceImpl implements BookService {
```

```
@Component
public class Libro implements Categoria {
    /**
```

```
@Repository
public class UserDaoImpl implements UserDao {
```



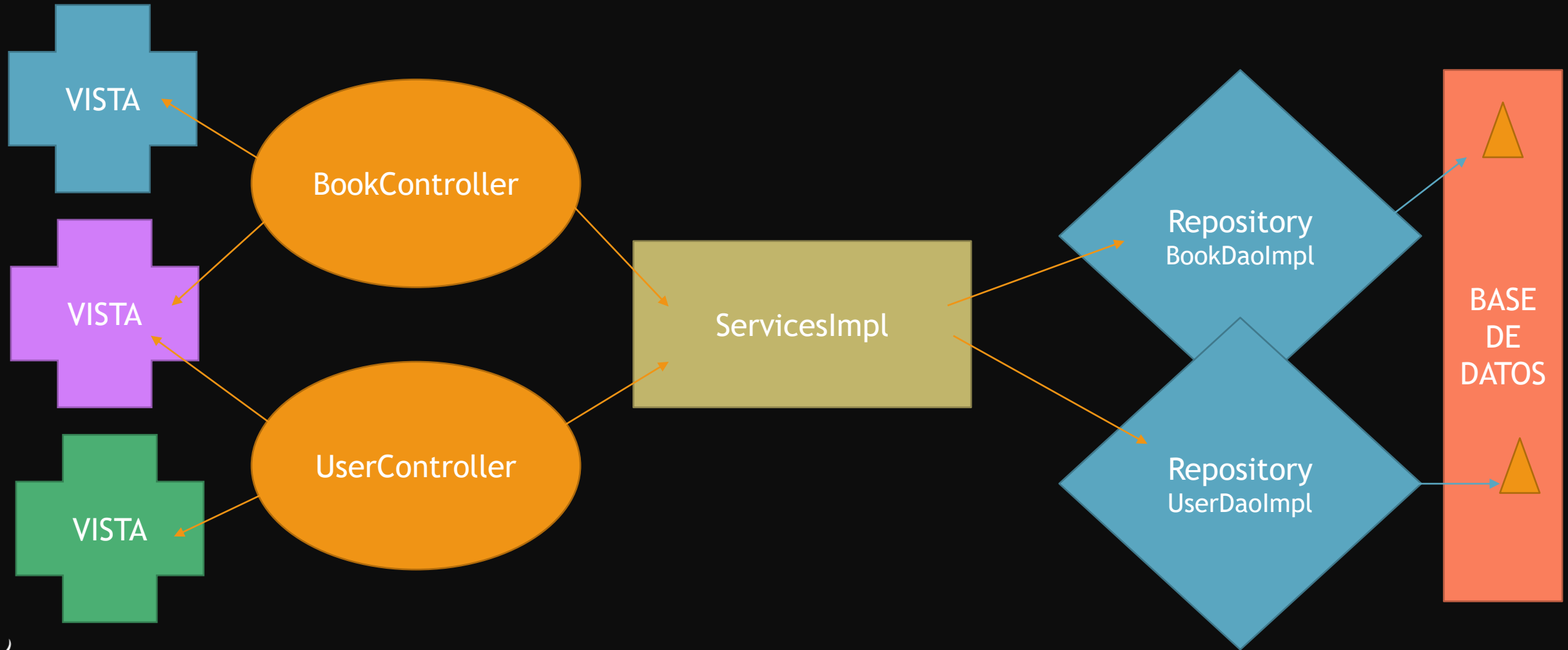
Añadiendo una tabla con libros

En esta parte, vemos como modificar la aplicación para poder rellenar la página con datos de otra clase.



Esquema de la configuración de las clases

Puedo añadir más rutas a mis controladores y más controladores a mi proyecto. Pues mi fichero pide a Spring que los indexe con `@ComponentScan`



Clase Libro

Estructura básica de la clase Libro

```
@Entity
@Table(name = "book")
public class Book {

    • @Id
      @GeneratedValue
      @Column(name = "id")
      private Long id;

    • @Column(name = "title")
      @Size(min = 1, max = 225, message = "{book.title.invalid}")
      private String title;

    • @Column(name = "author")
      private String author;

    • public Book() {
      }
}
```



Añadiendo la clase Libro y sus dependencias

Para empezar, necesitamos indicar a Spring qué clases van a tener relación con la base de datos. Esto se consigue modificando el método `setAnnotatedClasses` de `AppConfig.class`

```
factoryBean.setHibernateProperties(props);  
factoryBean.setAnnotatedClasses(new Class[] {User.class, Book.class});
```

Luego, necesitamos crear una nueva interfaz y conectarla desde el controlador. Así podremos gestionar los servicios (Service) así como el objeto de acceso a datos (DAO) de nuestros libros.

```
public interface BookService {  
    void save(Book book);  
  
    List<Book> listBooks();  
  
    void deleteBook(long id);  
  
    Book getBook(long id);  
  
    long numBooks();  
}
```



Configurando el acceso a datos

Seguiremos cambiando el nombre de la clase UserServiceImpl. La renombraremos ServicesImpl.java. Haremos que implemente los métodos de las interfaces servicios adaptando los nombres según el caso.

```
@Service
public class ServicesImpl implements UserService, BookService {

    @Autowired
    private UserDao userDao;

    @Autowired
    private BookDao bookDao;
```

Implementaremos los métodos necesarios y los conectaremos a la interfaz DAO adecuada.

```
@Override
@Transactional
public long numBooks() {
    return bookDao.num();
}

@Override
@Transactional
public Book getBook(long id) {
    Book book = bookDao.get(id);
    return book;
}
```

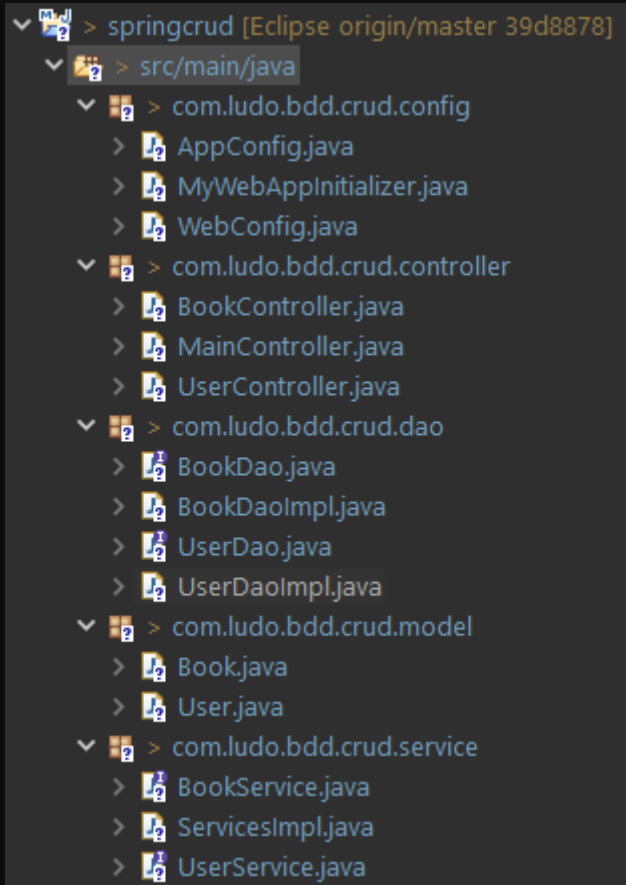


Finalizando el proceso

No olvidaremos actualizar los ficheros de propiedades

```
book.title.invalid = Título inválido. Ingresar un título que tenga entre {min} y {max} caracteres.
```

Seguiremos con las interfaces de DAO y al final tendríamos la estructura de archivos siguiente



Resultado final

[Portada](#)[Contacto](#)[Gestionar usuarios](#)[Gestionar libros](#)

FORMULARIO PARA INGRESAR USUARIOS

Nombre

Email

Validar

LISTA DE USUARIOS

Nombre	Email	Actualizar	Borrar
Gorka	gorka@gmail.com	Modificar	Borrar
Ludovick	ludoviclaisnez@gmail.com	Modificar	Borrar
Ignacio	ignacio@gmail.com	Modificar	Borrar
rafa	rafa@erfeffe.com	Modificar	Borrar
Reshad	reshad@gmail.com	Modificar	Borrar
Iñigo	inigo@gmail.com	Modificar	Borrar



Resultado final p2

[Portada](#)[Contacto](#)[Gestionar usuarios](#)[Gestionar libros](#)

FORMULARIO PARA INGRESAR LIBROS

Título

Autor

Validar

LISTA DE LIBROS

Título	Autor	Actualizar	Borrar
20000 lieues sous les mers	Jules Vernes	Modificar	Borrar
Les misérables	Victor Hugo	Modificar	Borrar
Les Fables	La Fontaine	Modificar	Borrar
Cujo	Stephen King	Modificar	Borrar

© 2022 [campus2b.com](#). Todos derechos reservados.

