Programación de componentes de accesos a datos

Tema 6. Acceso a datos

Objetivos

- Programación orientada a componentes
- Plataformas y modelos
- Spring Boot

¿Qué es un componente?

Un componente es un objeto escrito de acuerdo a unas especificaciones. No importa qué especificación sea ésta, siempre y cuando el objeto se adhiera a la especificación.

Solo cumpliendo correctamente con esa especificación es que el objeto se convierte en componente y adquiere características como reusabilidad.

Programación Orientada a Componentes

La programación orientada a componentes es una rama de la ingeniería del software, con énfasis en la descomposición de sistemas ya conformados en componentes

Los componentes dispone de interfaces bien definidas usadas para la comunicación entre ellos.

POO vs POC

- Las reutilizaciones de objetos son complicadas.
- No incorpora aspectos tales como distribución y empaquetado decomponentes.
- Imposibilidad de separar aspectos composicionales y computacionales.
- No define una unidad de composición de aplicaciones software.

Ventajas de la POC

- Reutilización de componentes.
- Al desarrollar cada componente se prueba por separado.
- Posibilidad de actualizar y/o agregar componentes a una aplicación en función de la necesidad.
- Un componente puede, una vez construido, mejorarse continuamente.
- El retorno sobre la inversión mejorará notablemente.

Desventajas POC

- En principio el desarrollador de componentes no sabe para quién está dirigido el componente ni como lo utilizará.
- Además, cuando finalmente el usuario del componente lo utilice, seguramente no pueda adaptarlo a sus necesidades, ya que no tiene acceso a su implementación.
- A esto, hay que añadir que aún no existe una solución estándar al problema de la convivencia entre versiones de un mismo componente.

Propiedades

Una propiedad de un componente es un atributo que afecta a su comportamiento o apariencia.

- Simples
- Indexadas
- Ligadas
- Restringidas

Propiedades ligadas

Son aquellas que al cambiar se informa a otros componentes permitiéndoles realizar alguna acción.

En este caso, la notificación del cambio se realiza mediante un PropertyChangeEvent.

Los componentes que quieran ser notificados del cambio en una propiedad deberán registrarse como auditores.

Características

Entre las principales características de la arquitectura basada en componentes destacan:

- Eventos.
- Persistencia.
- Introspección y reflexión.

Eventos

Un evento es un mecanismo utilizado por los componentes para comunicarse entre sí.

Son mensajes que se envían entre componentes notificando que algo ha ocurrido.

Eventos

- Un evento es un mecanismo utilizado por los componentes para comunicarse entre sí. Son mensajes que se envían entre componentes notificando que algo ha ocurrido.
- El componente fuente del evento invoca a un método del oyente pasándole como argumento un objeto que almacena toda la información sobre el evento.

Eventos

```
import java.util.*;
public class StockEvent extends EventObject {
    protected int anteStock, nuevoStock;

    public StockEvent (Object fuente , int anterior, int nuevo) {
        super(fuente);
        nuevoStock=nuevo;
        anteStock=anterior;
    }

    public int getNuevoStock (){ return nuevoStock;}

    public int getAnteStock (){ return anteStock;}
}
```

Persistencia

- Mecanismo que permite guardar y restaurar el estado de los componentes junto con los valores personalizados. Para que un componente sea persistente, es necesario que implemente la interfaz Serializable.

Introspección y Reflexión

Para conocer las propiedades, métodos y eventos de un componente, la herramienta de desarrollo tiene dos opciones:

- Introspección
- Reflexión

Introspección

Define un interfaz standard para poder acceder a las propiedades disponibles de un objeto.

- public List<String> getProperties() {...}
- public Object getProperty(String property) {...}
- public void setProperty(String property, Object value) {…}

Reflexión

El API de Java reflection nos permite leer los metadatos de nuestras clases y trabajar con ellos.

```
Method[] metodos=c.getClass().getMethods();
...
if (m.getName().equals("getId"){
                String cadena=(String) m.invoke(c, null);
                System.out.println(cadena);
}
```

Modelos de componentes

- Los modelos de componentes son los estándares encargados de definir la forma de las interfaces de los componentes y determinar los mecanismos de composición y comunicación entre ellos.
- Ejemplos de modelos son: COM/DCOM, JavaBeans y CORBA.

CORBA

- Estándar abierto creado por el OMG (Object Management Group) que se encarga de definir estándares para facilitar la interoperabilidad y portabilidad.
- Este estándar es el encargado de permitir que componentes software escritos en distintos lenguajes de programación y que ejecutan en distintas maquinas puedan trabajar juntos.

COM

- Modelo de componentes creado por Microsoft que permite, al igual que CORBA, la interoperabilidad entre componentes independientemente de los lenguajes de programación en los que se hayan escrito y de las plataformas sobre las que corran.

JAVABEANS

- Modelo de componentes creado por la compañía Sun Microsystems para la construcción de aplicaciones con el lenguaje de programación Java.
- Hoy en día es uno de los modelos de desarrollo de software basado en componentes con más aceptación.
- Permite la comunicación entre componentes a través de eventos.

Objetos JavaBean

Un JavaBean es un objeto Java que presenta ciertas características:

- Posee un constructor sin argumentos.
- Las propiedades son accesibles con los métodos get y set estudiados anteriormente.
- Es serializable, es decir, implementa la interfaz Serializable.

El principal objetivo de los JavaBeans es ser contenedores de código que puedan ser empaquetados e incorporados a aplicaciones que necesiten de esa funcionalidad.

Clase BeanInfo

- Clase que contiene la información sobre las propiedades, eventos y métodos del componente.

  Public class PersonaBeanInfo extends BeanInfo.

- Esta interfaz proporciona los métodos necesarios para obtener información sobre las propiedades, eventos y métodos.
- Sobreescribir los métodos necesarios.

Spring

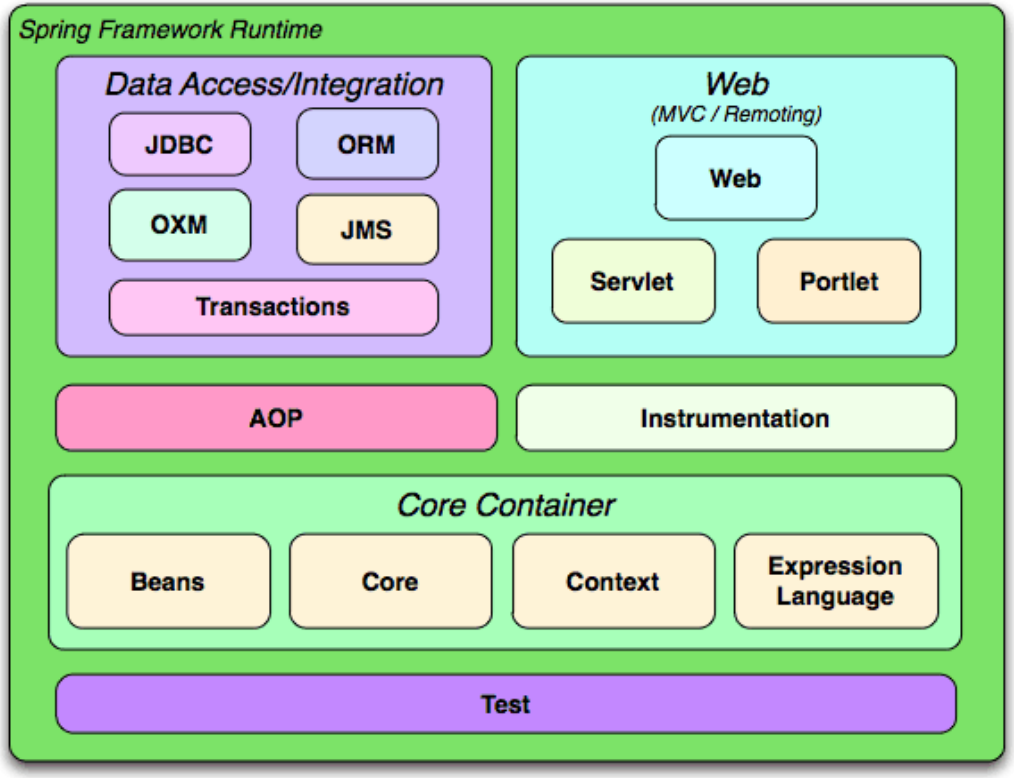
Spring es un framework Open Source que facilita la creación de aplicaciones de todo tipo en Java.

Spring

Spring Framework está dividido en diversos módulos que podemos utilizar:

- **Core container:** proporciona la funcionalidad básica.
- **Web:** nos permite crear controladores Web, tanto de vistas MVC como aplicaciones REST.
- **Acceso a datos:** abstracciones sobre JDBC, ORMs como Hibernate, etc...
- **Instrumentación:** proporciona soporte para la instrumentación y reutilización de clases.
- **Pruebas de código:** contiene un framework de testing con soporte para JUnit.

Spring


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
(MVC / Remoting)
Web
Servlet
Portlet
AOP
Instrumentation
Core Container
Beans
Core
Context
Expression Language
Test

Spring Boot

Spring Framework es muy potente pero la configuración inicial y la preparación de las aplicaciones para producción son tareas bastante tediosas.

Spring Boot simplifica el proceso al máximo gracias a sus dos principales mecanismos:

- **Contenedor de aplicaciones integrado**
- **Starters**

Spring Boot

Contenedor de aplicaciones integrado

- Permite compilar nuestras aplicaciones Web como un archivo .jar que podemos ejecutar como una aplicación Java normal (como alternativa a un archivo .war, que desplegaríamos en un servidor de aplicaciones como Tomcat).

Spring Boot

Starters

- Spring Boot nos proporciona una serie de dependencias, llamadas starters, que podemos añadir a nuestro proyecto dependiendo de lo que necesitemos.
- Una vez añadimos un starter, éste nos proporciona todas las dependencias que necesitamos, tanto de Spring como de terceros.
- Además, los starters vienen configurados con valores por defecto, que pretenden minimizar la necesidad de configuración

Spring Boot

Repositorios

- Es un interfaz que permite realizar operaciones de consulta y gestión sobre modelos de datos ya existentes.
- Las operaciones básicas ya suelen estar implementadas, por lo que solo hay que extenderlas.

Spring Boot

Repositorios

- **CrudRepository** provee las operaciones básicas
- **PagingAndSortingRepository** permite ordenar y hacer paginación
- **JpaRepository** permite realizar las operaciones de persistencias con el standard JPA (por ejemplo Hibernate)

Spring Boot

Servicios

- Son las clases donde se realiza la lógica de programación.
- Normalmente usan los repositorios que se hayan definido.
- Los métodos son usados por los controladores.
- No es obligatoria su implementación

Spring Boot

Controladores

- Es donde se procesan las solicitudes de los clientes.
- Se realiza el mapeado de la url y usan los métodos disponibles en los diferentes servicios.
- El resultado se encapsula normalmente en JSON o HTML y se devuelve al cliente.

Spring Boot

Plantillas

- Son archivos HTML que incluyen marcas y atributos especiales que se completan en tiempo de ejecución.
- Desde el controlador se pasan objetos a la plantilla para completarla.
- Se permiten estructuras complejas como condicionales y bucles para iterar sobre objetos que sean colecciones.

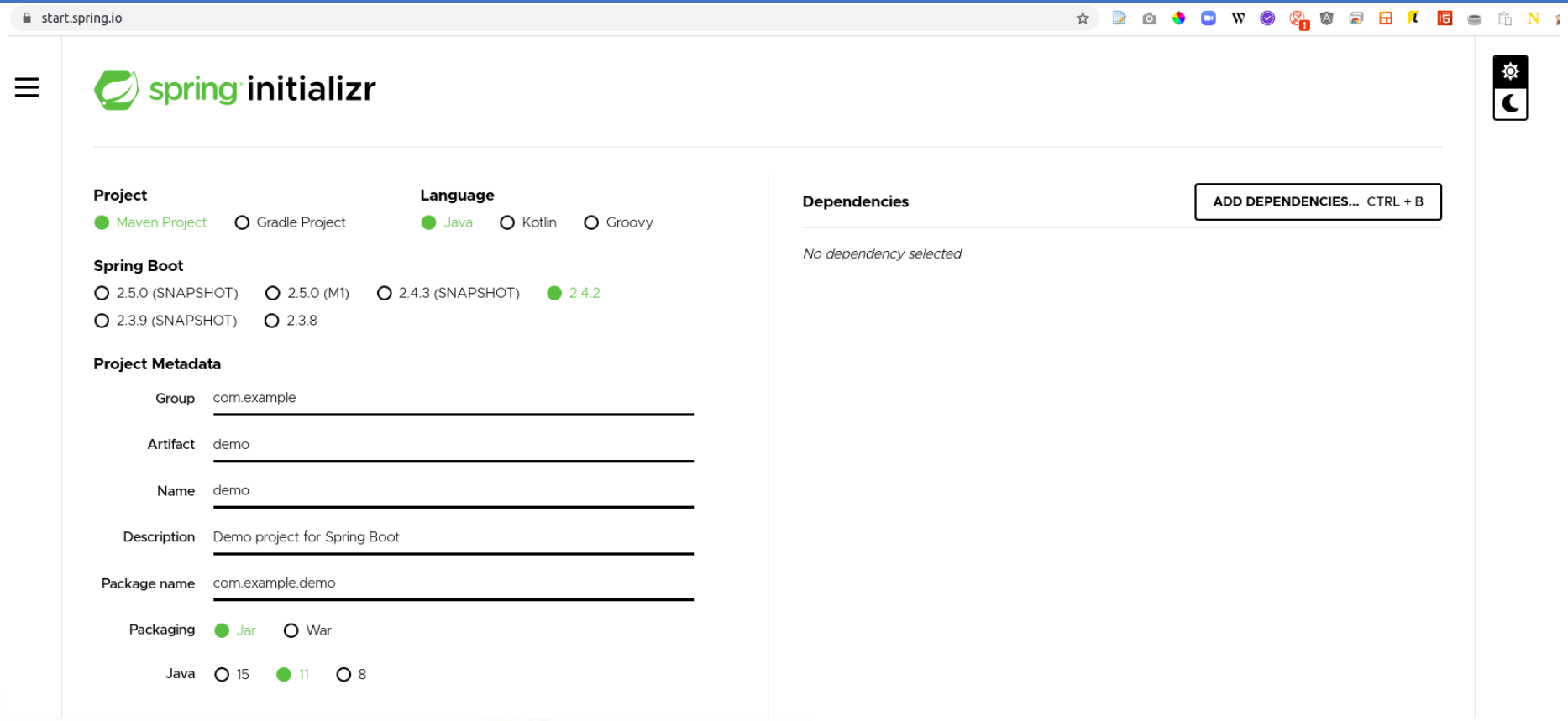
Spring Boot

```
com
+- tutorialspoint
     +- myproject
          +- Application.java
          |
          +- model
          | +- Product.java
          +- dao
          | +- ProductRepository.java
          +- controller
          | +- ProductController.java
          +- service
          | +- ProductService.java
```

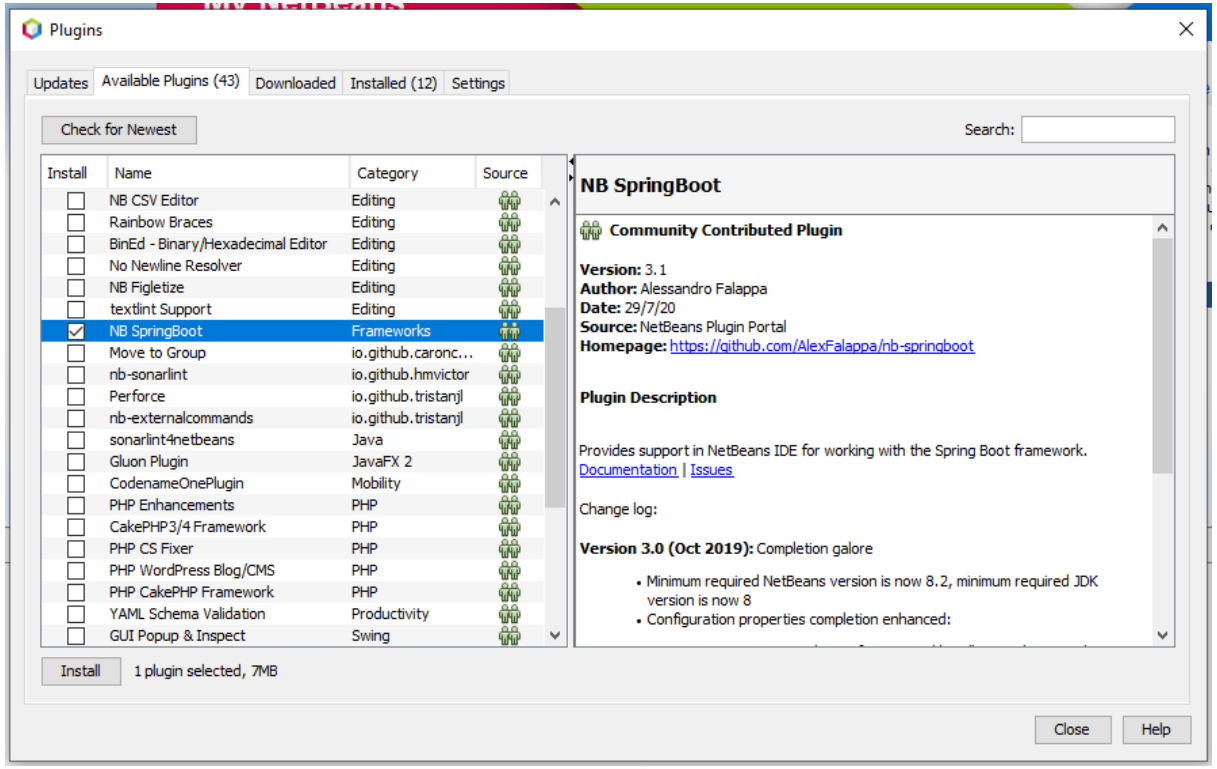
Spring Boot

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-configuration-processor</artifactId>
        <optional>true</optional>
</dependency>
```

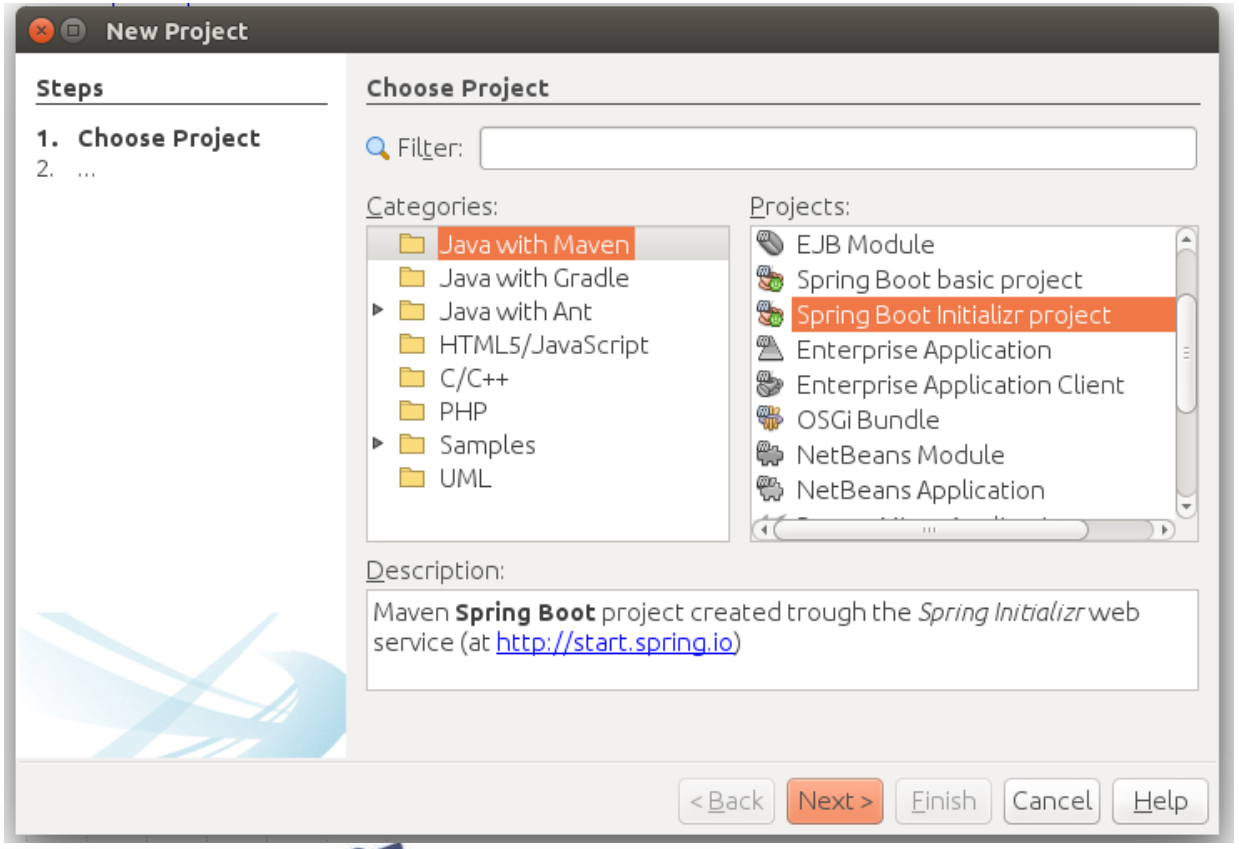
Spring Boot


start.spring.io
spring initializr
Project
Maven Project
Gradle Project
Language
Java
Kotlin
Groovy
Spring Boot
2.5.0 (SNAPSHOT)
2.5.0 (M1)
2.4.3 (SNAPSHOT)
2.4.2
2.3.9 (SNAPSHOT)
2.3.8
Project Metadata
Group
com.example
Artifact
demo
Name
demo
Description
Demo project for Spring Boot
Package name
com.example.demo
Packaging
Jar
War
Java
15
11
8
Dependencies
ADD DEPENDENCIES... CTRL + B
No dependency selected

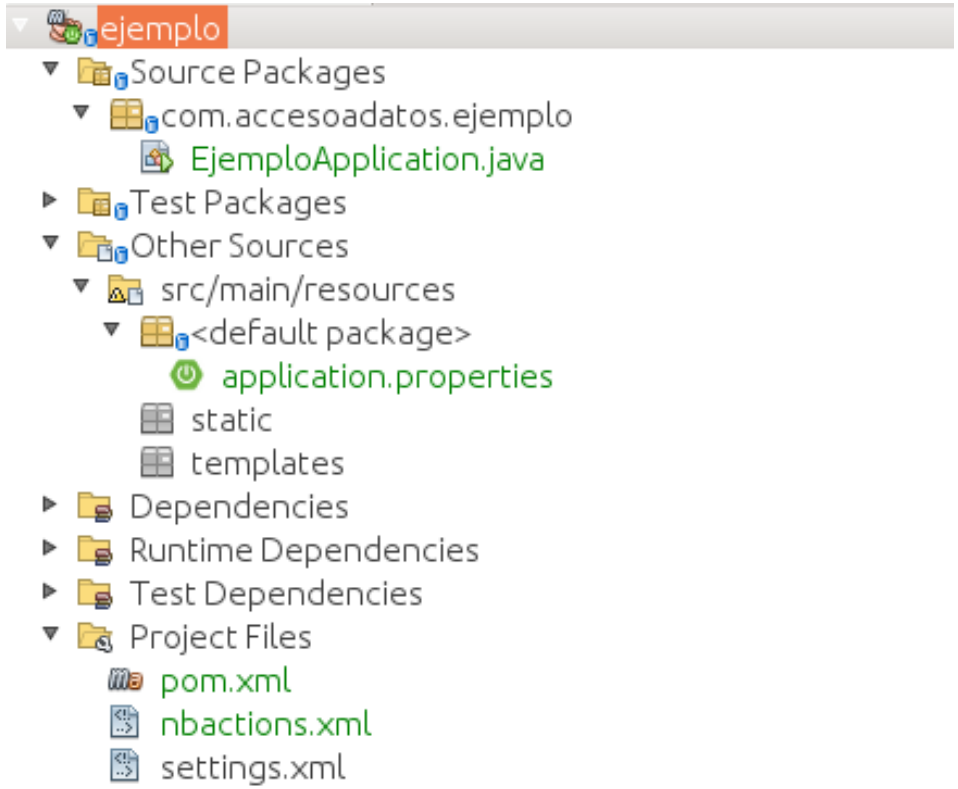
Plugin para NetBeans



Spring Boot


New Project
Steps
1. Choose Project
2. ...
Choose Project
Filter:
Categories:
Java with Maven
Java with Gradle
Java with Ant
HTML5/JavaScript
C/C++
PHP
Samples
UML
Projects:
EJB Module
Spring Boot basic project
Spring Boot Initializr project
Enterprise Application
Enterprise Application Client
OSGi Bundle
NetBeans Module
NetBeans Application
Description:
Maven Spring Boot project created trough the Spring Initializr web service (at http://start.spring.io)
< Back
Next >
Finish
Cancel
Help

Spring Boot


ejemplo
Source Packages
com.accesoadatos.ejemplo
EjemploApplication.java
Test Packages
Other Sources
src/main/resources
<default package>
application.properties
static
templates
Dependencies
Runtime Dependencies
Test Dependencies
Project Files
pom.xml
nbactions.xml
settings.xml

Spring Boot

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.5.0-SNAPSHOT)

2021-02-14 22:31:51.913  INFO 1212322 --- [  restartedMain] c.a.ejemplo.EjemploApplication           : Starting EjemploApplic
2021-02-14 22:31:51.915  INFO 1212322 --- [  restartedMain] c.a.ejemplo.EjemploApplication           : No active profile set,
2021-02-14 22:31:51.952  INFO 1212322 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defa
2021-02-14 22:31:51.953  INFO 1212322 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web re
2021-02-14 22:31:52.706  INFO 1212322 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized wit
2021-02-14 22:31:52.716  INFO 1212322 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomc
2021-02-14 22:31:52.717  INFO 1212322 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engin
2021-02-14 22:31:52.759  INFO 1212322 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring em
2021-02-14 22:31:52.759  INFO 1212322 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationCon
2021-02-14 22:31:52.897  INFO 1212322 --- [  restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorS
2021-02-14 22:31:53.007  WARN 1212322 --- [  restartedMain] ion$DefaultTemplateResolverConfiguration : Cannot find template
2021-02-14 22:31:53.047  INFO 1212322 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is r
2021-02-14 22:31:53.073  INFO 1212322 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port
2021-02-14 22:31:53.083  INFO 1212322 --- [  restartedMain] c.a.ejemplo.EjemploApplication           : Started EjemploApplica
```

Conectar la base de datos

Lo primero que hay que hacer es indicar los parámetros de conexión con la base de datos que vamos a usar.

En el caso de usar Hibernate como orm JPA...

NO ES NECESARIO EL ARCHIVO DE CONFIGURACION.

Persistencia de datos JPA y repositorios

La información de la base de datos se introduce en el archivo **aplication.propierties** dentro de src/main/resources

```
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3307/ejemplo?useUnicode=true&useJDBCCompliantTime
spring.datasource.username = root
spring.datasource.password = pacoromero
## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto = update
```

Persistencia de datos JPA y repositorios

org.springframework.data.repository

Interface CrudRepository<T,ID>

All Superinterfaces:

`Repository<T,ID>`

All Known Subinterfaces:

`PagingAndSortingRepository<T,ID>`

```
@NoRepositoryBean
public interface CrudRepository<T,ID>
extends Repository<T,ID>
```

Interface for generic CRUD operations on a repository for a specific type.

Author:

Oliver Gierke, Eberhard Wolff

Method Summary

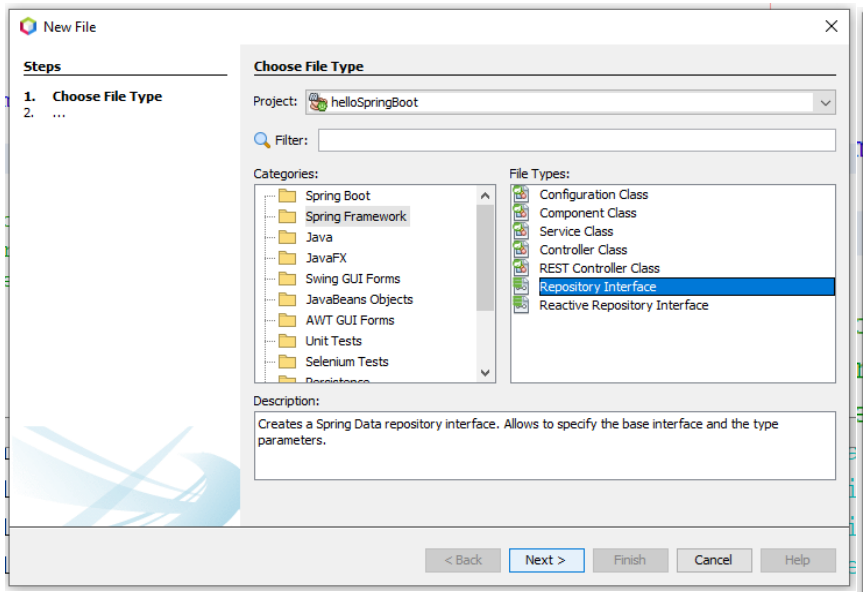
All Methods	Instance Methods	Abstract Methods

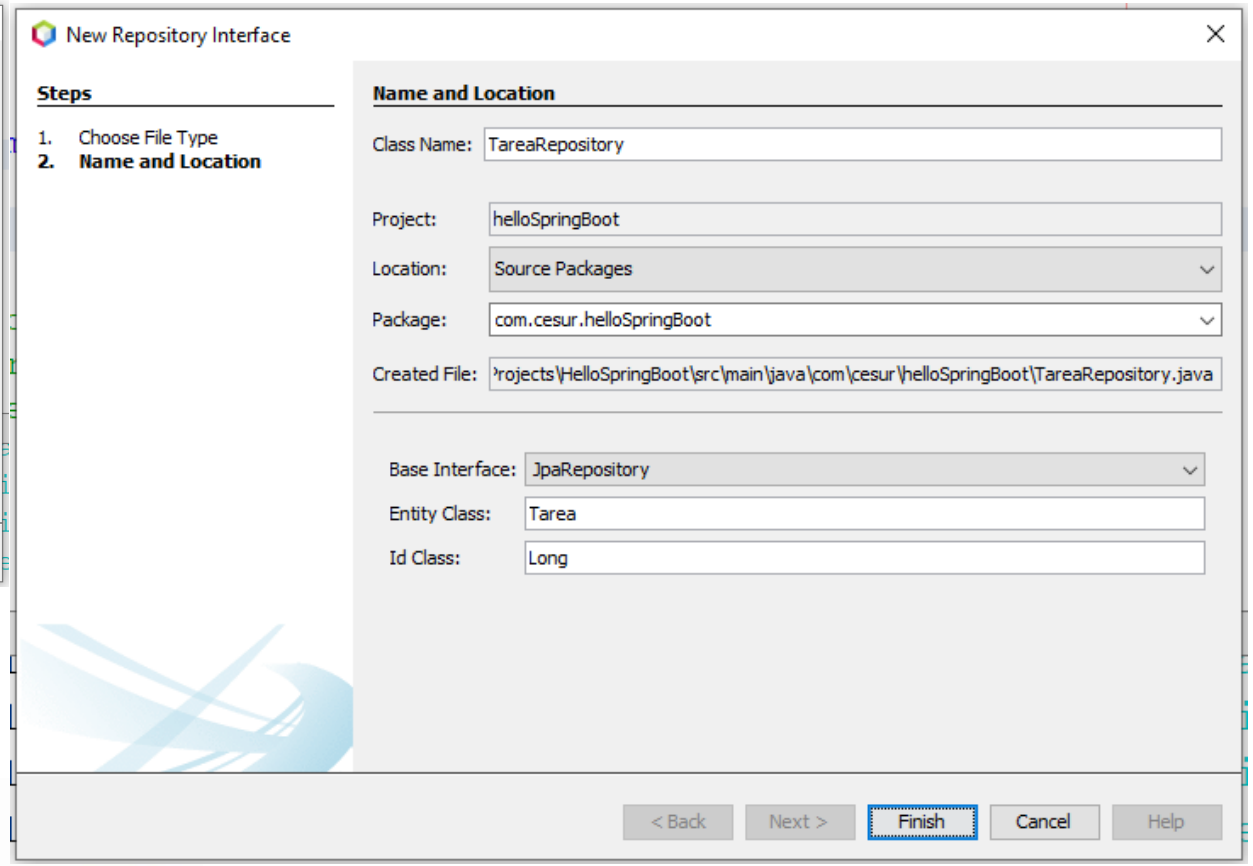
Modifier and Type	Method and Description
long	count()

Creamos el modelo anotado con JPA

```java
@Entity
@Table(name="tarea")
@Data
public class Tarea implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    private String nombre;
    private String prioridad;
    private String responsable;
}
```

Creamos el repositorio asociado





Persistencia de datos JPA y repositorios

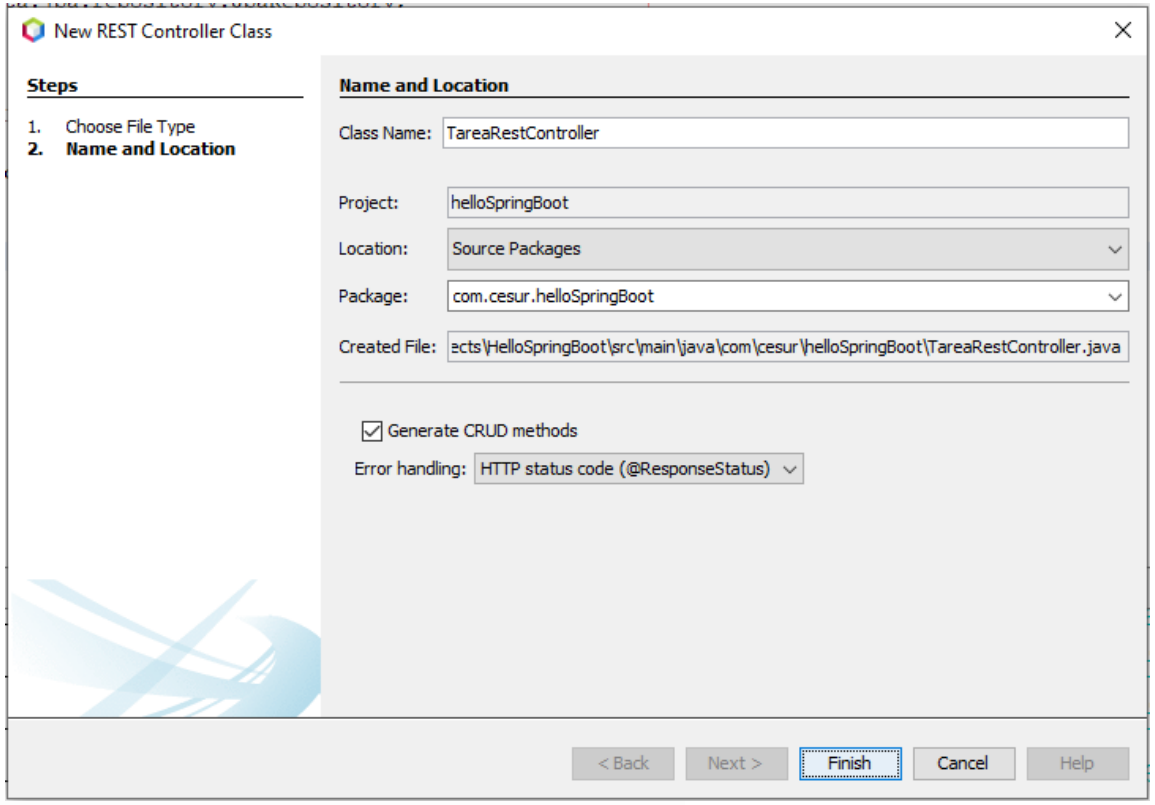
```java
import org.springframework.data.jpa.repository.JpaRepository;

public interface LibroRepository extends JpaRepository<models.Libro, Integer> {

}
```

```java
@Controller
public class MainController {
    @Autowired
    private LibroRepository repositorio;
```

Creación de Controladores REST



Errores cuando el controlador no encuentra el modelo

```
Caused by: java.lang.IllegalArgumentException: Not a managed type: class models.Tarea
        at org.hibernate.metamodel.internal.MetamodelImpl.managedType(MetamodelImpl.java:582) ~[hibernate-c
        at org.hibernate.metamodel.internal.MetamodelImpl.managedType(MetamodelImpl.java:85) ~[hibernate-co
        at org.springframework.data.jpa.repository.support.JpaMetamodelEntityInformation.<init>(JpaMetamodel
        at org.springframework.data.jpa.repository.support.JpaEntityInformationSupport.getEntityInformation
        at org.springframework.data.jpa.repository.support.JpaRepositoryFactory.getEntityInformation(JpaRepo
        at org.springframework.data.jpa.repository.support.JpaRepositoryFactory.getTargetRepository(JpaRepos
        at org.springframework.data.jpa.repository.support.JpaRepositoryFactory.getTargetRepository(JpaRepos
```

```java
@SpringBootApplication
@EntityScan(basePackages="models")

public class HelloSpringBootApplication {

        public static void main(String[] args) {
                SpringApplication.run(HelloSpringBootApplication.class, args);
        }

}
```

Controladores REST

```java
@RestController
@RequestMapping("/biblioteca")
public class TareaRestController {

    @Autowired
    private TareaRepository repositorio;

    @GetMapping()
    public List<models.Tarea> list() {
        return repositorio.findAll();
    }

    @GetMapping("/{id}")
    public Object get(@PathVariable Long id) {
        return repositorio.getById(id);
    }
```

Paso de parámetros

@PathVariable --> a través de la url

@RequestBody --> a través del cuerpo de la petición (el parámetro va serializado en un json y se mapea sobre un objeto pojo)

@ModelAttribute --> serializado en la petición como un formulario POST

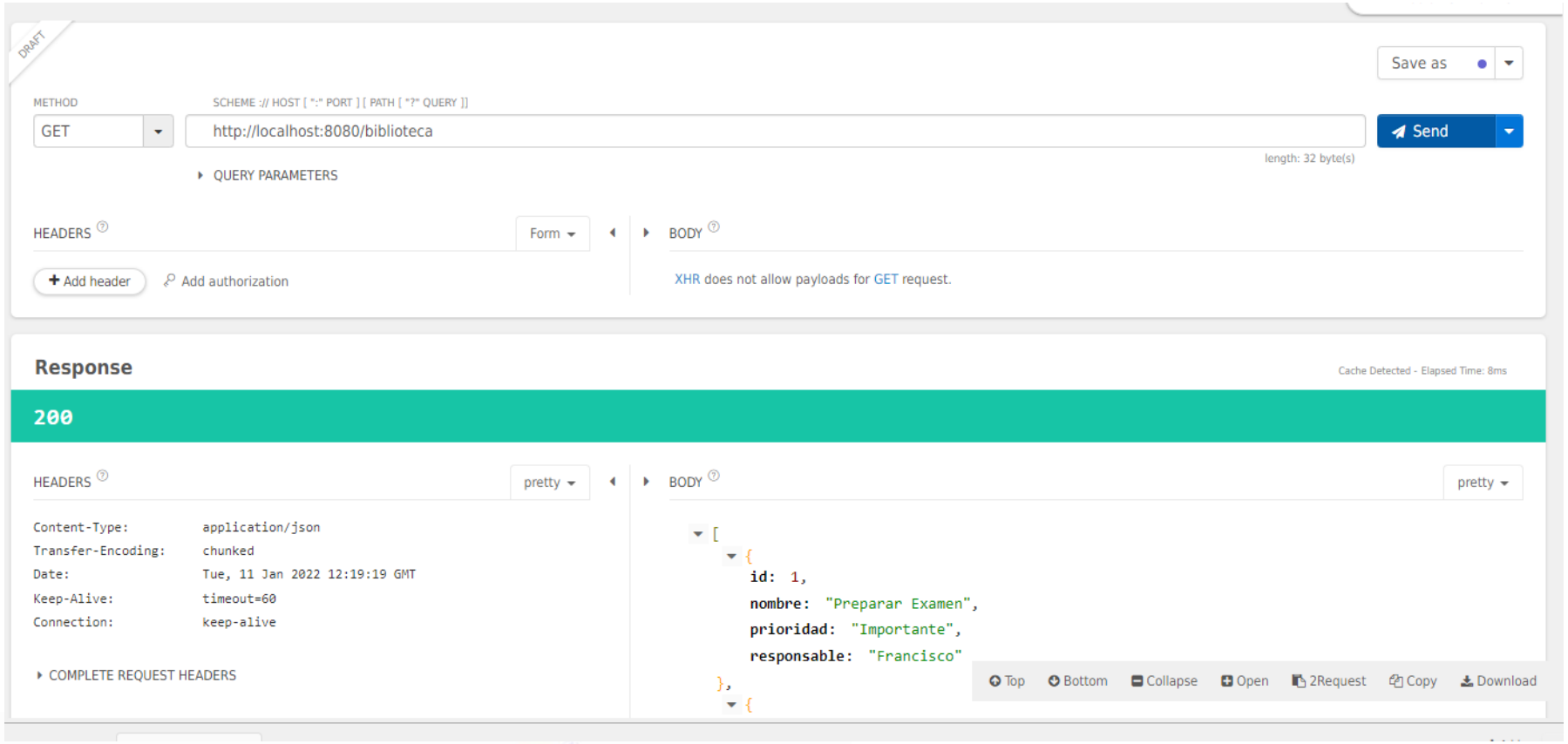
Esquema de rutas en la url

A la hora de crear una aplicación web, la ruta de la url se usa para acceder a los diferentes objetos a los que el usuario puede acceder.

Se suelen usar con los verbos de HTTP.

- `[GET]  /biblioteca`
- `[GET]  /libro/{id}`
- `[POST] /libro/{id}`
- `[GET]  /biblioteca/{categoria_id}`

Probando la API



Uso de parámetros en la plantilla

```java
@RequestMapping("/web/{id}")
public String one(@PathVariable Long id, Model model) {
    model.addAttribute("tarea", repositorio.getById(id) );
    model.addAttribute("fecha", new Date() );
    return "tarea";
}
```

tarea.html

```html
<html>
    <head>
        <title>Tarea <span th:text="${tarea.id}"/> </title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h1 th:text="${tarea.nombre}"></h1>
        <p th:text="${tarea.responsable}"></p>
        <p th:text="${tarea.prioridad}"></p>
    </body>
</html>
```

Uso de parámetros en la plantilla

```
@RequestMapping(path="/biblioteca", method = RequestMethod.GET)
public String listado(Model pagina) {
    pagina.addAttribute("biblioteca",repositorio.findAll());
    return "vista-biblioteca";
}
```

```
<tr th:each="libro : ${biblioteca}">
    <td width="20px" th:text="${libro.id}"></td>
    <td th:text="${libro.nombre}"></td>
    <td th:text="${libro.autor}"></td>
    <td th:text="${libro.estado}"></td>
    <td><a th:href="@{'/biblioteca/'+${libro.id}}">Ver detalles</a></td>
</tr>
```

Themeleaf

- Añadiendo "atributos" (elementos) al modelo de la plantilla:

```
model.addAttribute("fecha",
dateFormat.format(new Date()));
```

- Mostrando la información del modelo en la plantilla:

```
Hoy es <span th:text="${fecha}"/>
```

Themeleaf

- Añadiendo listas al modelo:

```
List<Student> students = new ArrayList<Student>();
….
model.addAttribute("students", students);
```

- Mostrando la información del modelo en la plantilla:

```
<tr th:each="student: ${students}">
    <td th:text="${student.id}" />
    <td th:text="${student.name}" />
</tr>
```

Themeleaf

- Condicionales en la plantilla

```
<span th:if="${person.gender} == 'M'" th:text="Male" />
<span th:unless="${person.gender} == 'M'" th:text="Female" />
```

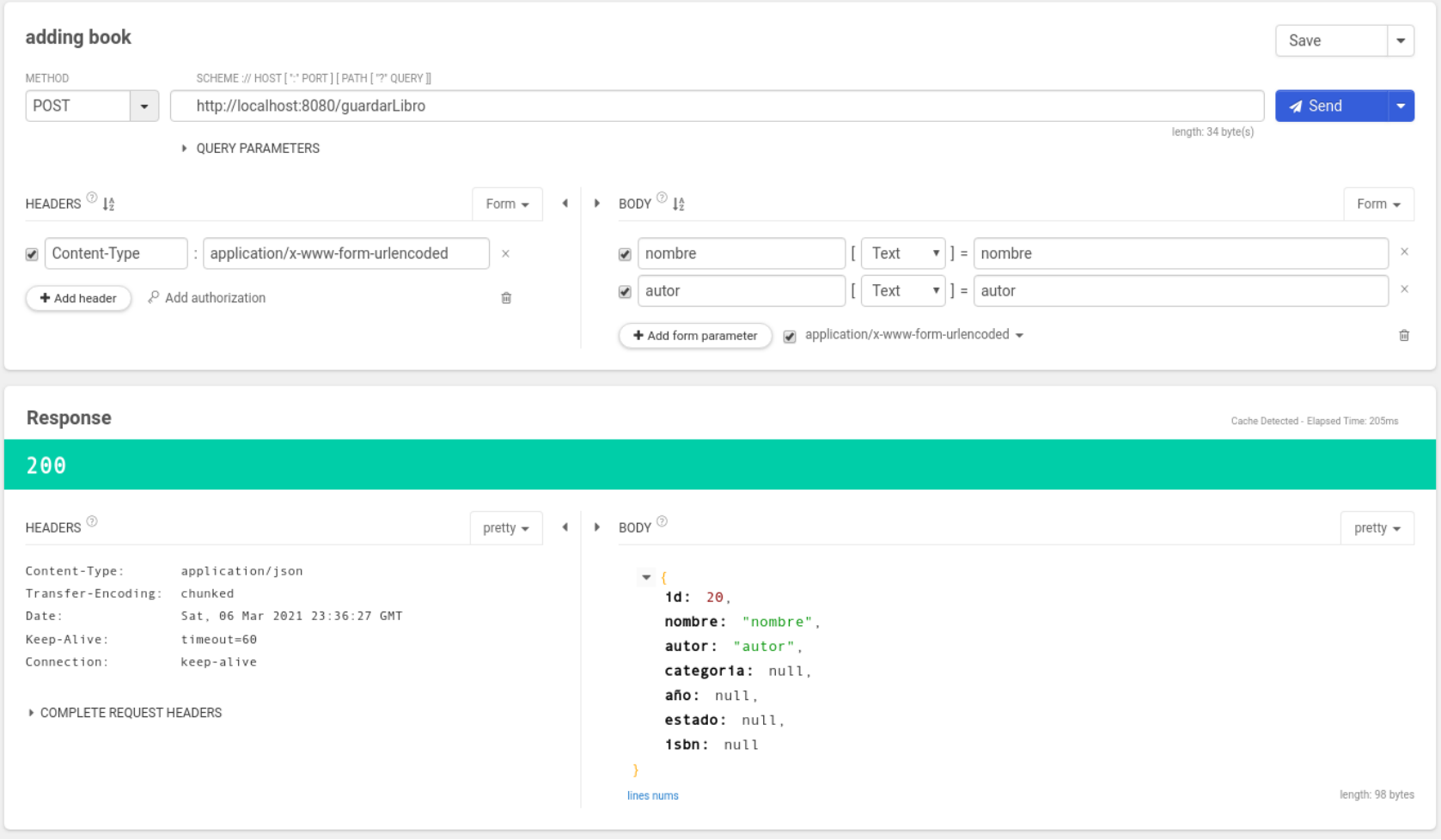
- Añadiendo enlaces

```
<a th:href="@{'/person/'+${person.id}}">Edit info</a>
```

Añadiendo objetos al repositorio desde un formulario

```
@ResponseBody
@RequestMapping(value = "/guardarLibro", method = RequestMethod.POST)
public Libro guardarLibro(@ModelAttribute Libro libro, Model model) {
    repositorio.save(libro);
    return libro;
}
```

Añadiendo objetos al repositorio



Variables de sesión

- El Interfaz **HttpSession** permite identificar a un usuario y almacenar información asociada a él a lo largo de todas las peticiones que se hagan a un mismo controlador.
- Las sesiones tienen un tiempo determinado.
- Métodos usuales para gestionarlas:
  - Cookies
  - Parámetros de url

Escribir en la sesión actual

```
@RequestMapping("/web/{id}")
public String one(@PathVariable Long id, Model model,HttpServletRequest request) {
    request.getSession().setAttribute("id", id);
    model.addAttribute("tarea", repositorio.getById(id) );
    model.addAttribute("fecha", new Date() );
    return "tarea";
}
```

Leer de la sesión actual

```
@RequestMapping("/web/actual")
public String recap(Model model,HttpServletRequest request) {
    Long id = (Long) request.getSession().getAttribute("id");
    if( id!=null){
        System.out.println("fetching " + id);
        model.addAttribute("tarea", repositorio.getById(id) );
        model.addAttribute("fecha", new Date() );
    }
    return "tarea";
}
```

Crear una API Rest con Spring para acceder a una biblioteca de libros.

Los libros están almacenados en una base de datos MySQL y para cada uno de ellos necesitamos: id, titulo, autor, categoria, ISBN, edicion

Queremos que se pueda hacer:

- Listar todos los libros (solo el id y titulo de cada uno)
- Listar el detalle de un libro concreto
- Listar todos los libros de una categoría concreta

Hay que:

- Diseñar el esquema de url necesario
- Realizar la programación.
- Realizar pruebas para testear el correcto funcionamiento.