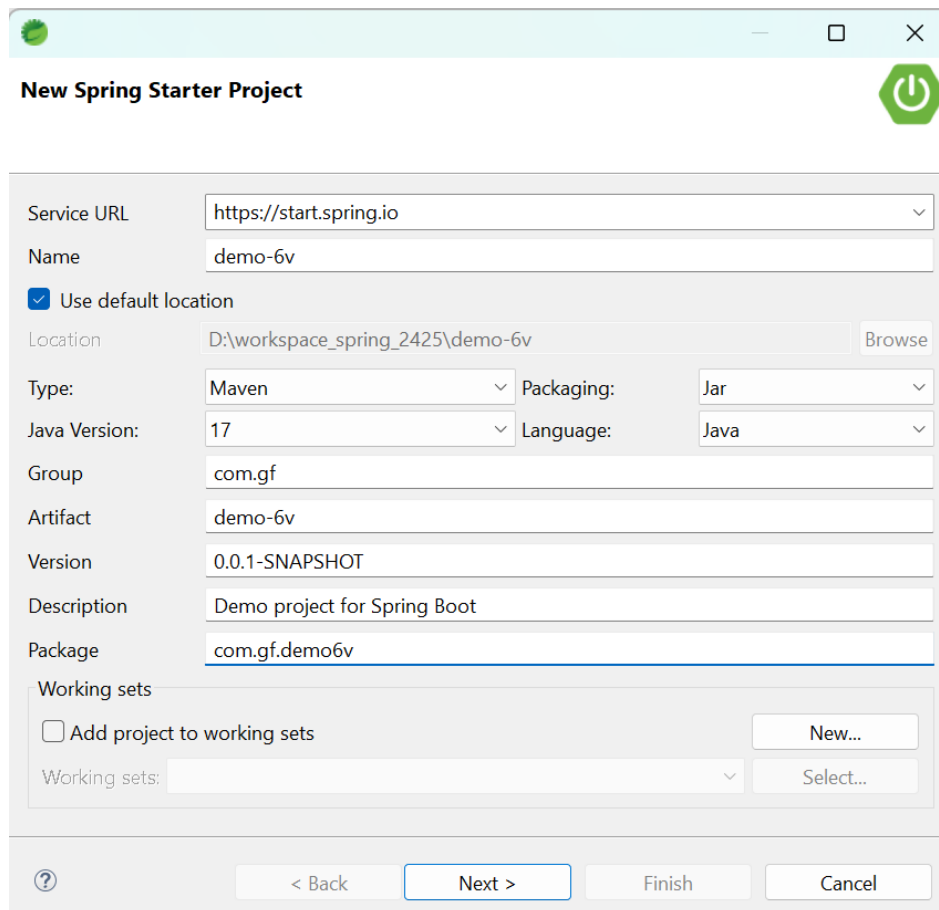




Crear un proyecto Spring Web+ MVC + JPA + Lombok

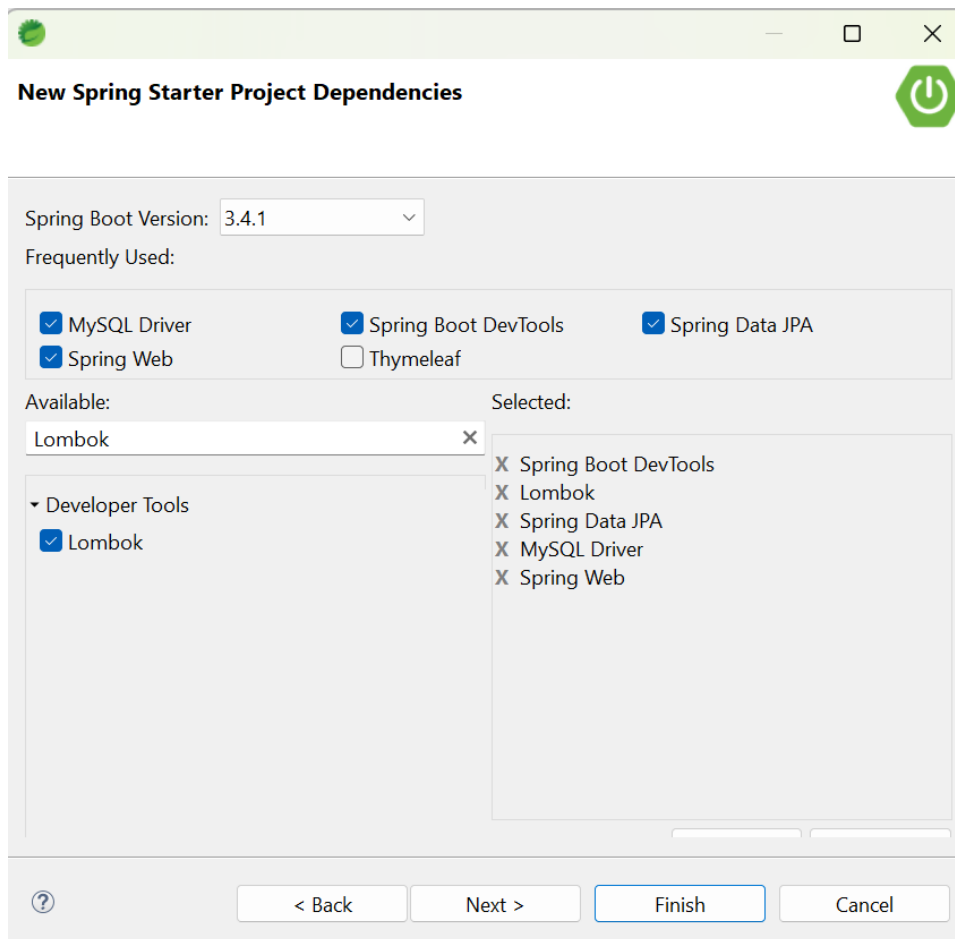
- ✓ *El primer paso es generar un proyecto de Spring Boot que soporte JPA en este caso solo necesitamos un starter adicional además de Web a la hora de crear nuestro proyecto:*



The screenshot shows the 'New Spring Starter Project' dialog box in the Eclipse IDE. The dialog has a title bar with a green power icon on the right. The main content area contains the following fields and options:

- Service URL:** A dropdown menu showing 'https://start.spring.io'.
- Name:** A text field containing 'demo-6v'.
- Use default location:** A checked checkbox.
- Location:** A text field showing 'D:\workspace_spring_2425\demo-6v' with a 'Browse' button to its right.
- Type:** A dropdown menu showing 'Maven'.
- Packaging:** A dropdown menu showing 'Jar'.
- Java Version:** A dropdown menu showing '17'.
- Language:** A dropdown menu showing 'Java'.
- Group:** A text field containing 'com.gf'.
- Artifact:** A text field containing 'demo-6v'.
- Version:** A text field containing '0.0.1-SNAPSHOT'.
- Description:** A text field containing 'Demo project for Spring Boot'.
- Package:** A text field containing 'com.gf.demo6v'.
- Working sets:** A section with an unchecked checkbox 'Add project to working sets', a 'New...' button, and a 'Working sets:' dropdown menu with a 'Select...' button.

At the bottom of the dialog, there is a row of buttons: a help icon (?), '< Back', 'Next >', 'Finish', and 'Cancel'.



También se pueden agregar las dependencias a posterior modificando el archivo **pom.xml** del nuestro proyecto.

- ✓ El siguiente paso es configurar los parámetros contra la base de datos usando el fichero de **application.properties** que tenemos en la estructura de nuestro proyecto:

```
spring.datasource.url=jdbc:mysql://localhost:3306/prueba
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

- ✓ A continuación, se genera una entidad que pueda ser persistida o seleccionada por JPA: se almacenará en el package entidades (entities) donde se incluirán todos los objetos de la aplicación.

```
@Entity
@Table (name="usuario")
public class Usuario {

    @Id
    private Long id;
    @Column(name="nombre")
    private String nombre;
    @Column(name="apellidos")
    private String apellidos;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
}
```

- ✓ Si hemos implementado la herramienta de Lombok la clase Usuario quedará más simplificada.
- ✓ A continuación, creamos una interfaz de repositorio apoyados en Spring Data JPA que **nos automatiza las operaciones básicas, ya sea de buscar, borrar, actualizar o crear un registro cuando se conecten a la base de datos**: se almacenarán en un package llamado repositorio (repository).
- ✓ Spring creará implementaciones de repositorio automáticamente, en tiempo de ejecución, desde la interfaz del repositorio.

```
public interface UsuarioRepository extends JpaRepository < Usuario, Long > {
}
```

Que Spring obligue al uso de interfaces en vez de implementaciones es una buena práctica para el servicio:

- **Abstracción:** El servicio que implementa una interfaz está obligado a ofrecer una serie de métodos bien definidos. Cualquier servicio que quiera cumplir con la interfaz tendrá que tener obligatoriamente un conjunto determinado de elementos. Esto te permite conectarte a diferentes servicios usando el mismo cliente.
 - **Usabilidad:** Si tu servicio implementa una interfaz, podrías reescribirlo completamente (o sustituirlo por otro distinto) y los clientes podrían seguir consumiendo tu servicio sin problemas. Ellos a lo sumo notarán diferencias en el rendimiento, pero poco más.
 - **Seguridad:** Una interfaz representa únicamente una declaración. Al exponer una interfaz no estás comprometiendo el código fuente del servicio.
 - **Mantenimiento:** Si un servicio expone una interfaz tu entorno de desarrollo puede verificar fácilmente que una nueva versión del servicio cumple (al menos) con los requisitos expuestos en la interfaz. Es una forma sencilla de automatizar un proceso que evita que se te olvide implementar parte de la funcionalidad.
- ✓ Ahora se debe **implementar una capa de servicio/negocio**. Primero se crea una interfaz de servicio llamada `UsuarioService`, y después la implementación en la clase `UsuarioServiceImpl`.

```
public interface UsuarioService {  
  
    List<Usuario> findAll();  
    void save(Usuario user);  
    Usuario findById(Long id);  
    void delete(Long id);  
  
}
```

En la clase `UsuarioServiceImpl`, se inyecta una instancia de `UsuarioRepository` a través de un campo privado mediante la anotación

```

@Service
public class UsuarioServiceImpl implements UsuarioService {

    @Autowired
    private UsuarioRepository usuarioRepository;

    @Override
    public List<Usuario> findAll() {
        // TODO Auto-generated method stub
        return usuarioRepository.findAll();
    }

    @Override
    public void save(Usuario user) {
        usuarioRepository.save(user);
    }

    @Override
    public Usuario findById(Long id) {
        Optional<Usuario> userResult = usuarioRepository.findById(id);
        Usuario u = null;
        if (userResult.isPresent()) {
            u = userResult.get();
        }
        return u;
    }

    @Override
    public void delete(Long id) {
        // TODO Auto-generated method stub
        usuarioRepository.deleteById(id);
    }
}

```

@Autowired. En tiempo de ejecución, Spring Data JPA generará una instancia de proxy de **UsuarioRepository** e inyectarla en la instancia de clase **UsuarioServiceImpl**.

Puede parecer que esta clase de servicio es redundante, ya que delega todas las llamadas a **UsuarioRepository**. Hay que tener en cuenta que, la lógica de negocios va a ser más compleja con el tiempo, por ejemplo, llamando a dos o más instancias de repositorio, así que creamos esta clase con el propósito de extensibilidad en el futuro.

Por último, inyectamos una instancia de la clase **UsuarioService** a este controlador: Spring creará automáticamente una en tiempo de ejecución. Escribiremos código para los métodos de controlador al implementar cada operación CRUD.

```

@RestController
public class UsuarioController {

    @Autowired
    UsuarioService usuario;

    @GetMapping("/{}/usuarios")
    public List<Usuario> getAll(){
        //retornará todos las preinscripciones
        return (List<Usuario>) usuario.findAll();
    }

    @PostMapping("/insert")
    public void createTitulo(@RequestBody Usuario user) {
        usuario.save(user);
    }

    @PutMapping("/update")
    public void updateTitulo(@RequestBody Usuario pre) {
        usuario.save(pre);
    }

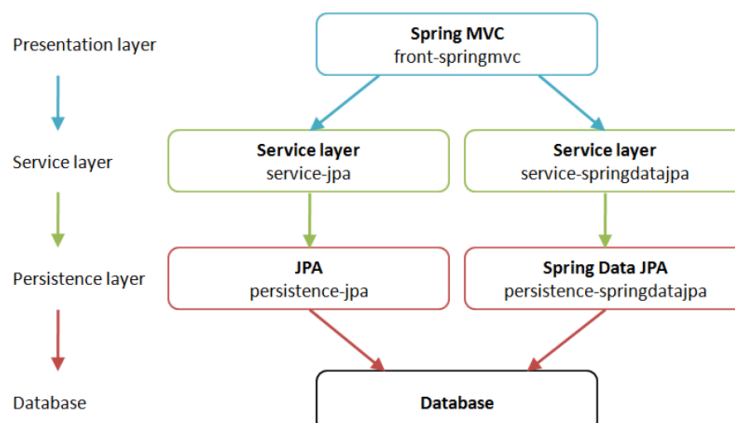
    @DeleteMapping("/delete/{id}")
    public void deleteTitulo(@PathVariable() Long id) {
        usuario.delete(id);
    }
}

```

Como parte de la gran familia de Spring Data, Spring Data JPA se construye como una capa de abstracción sobre el JPA. Por lo tanto, tenemos todas las características de JPA más la facilidad de desarrollo de Spring.

Durante años, los desarrolladores han escrito código repetitivo para crear una DAO de JPA para funcionalidades básicas. Spring ayuda a reducir significativamente esta cantidad de código al proporcionar interfaces mínimas e implementaciones reales.

Spring MVC JPA/Spring Data JPA



- ✓ Ahora se implementa la clase **UsuarioController.java**. @GetMapping asigna la solicitud `"/usuarios"` al método `getAll()`. Se crea *un servicio RES que devuelve la lista de usuarios en formato JSON directamente*.
- ✓ *Previamente queda invocar al repositorio o servicio desde el contenido del controller y nos selecciona los datos de la base de datos.*

```
@GetMapping(value="/usuarios", produces=MediaType.APPLICATION_JSON_VALUE)
public List<Usuario> getAll(){

    return (List<Usuario>) usuario.findAll();
}
```

- ✓ *La estructura final del proyecto Spring Web + MVC + JPA debe de quedar con el siguiente empaquetado de clases.*

```
demo-6 [boot] [devtools]
├── src/main/java
│   ├── com.gf.demo6
│   │   ├── Demo6Application.java
│   │   └── com.gf.demo6.controllers
│   │       ├── UsuarioController.java
│   │       └── com.gf.demo6.entities
│   │           ├── Usuario.java
│   │           └── com.gf.demo6.repositories
│   │               ├── UsuarioRepository.java
│   │               └── com.gf.demo6.services
│   │                   ├── UsuarioService.java
│   │                   └── UsuarioServiceImpl.java
└── src/main/resources
    ├── static
    ├── templates
    └── application.properties
```