



Lenguaje de Programación II

Curso	Lenguaje de Programación II (1894)
Formato	Manual de curso
Autor Institucional	Cibertec
Páginas	171 p.
Elaborador	Ventura González, Jorge Enrique
Revisor de Contenidos	Atuncar Guzmán, José Augusto

Índice

Presentación	4
Red de contenidos	5
UNIDAD DE APRENDIZAJE 1: JAVA PERSISTENCE API, MAVEN, GIT	
1.1 Tema 1 : Introducción al API de Persistencia JPA	7
1.1.1 : Introducción a JPA y componentes	7
1.1.2 : Entidades y anotaciones	9
1.1.3 : JPA Query Language	54
1.1.4 : JPA y relaciones	60
1.2 Tema 2 : Maven	64
1.2.1 : Introducción y configuración	64
1.2.2 : Creación de un proyecto	66
1.2.3 : Ciclo de vida	76
1.2.4 : Arquetipos	78
1.2.5 : Gestión de dependencias	79
1.3 Tema 3 : Git	85
1.3.1 : Introducción a los sistemas de control de versiones	85
1.3.2 : Gestión de escenarios	94
1.3.3 : GitHub básico	96
UNIDAD DE APRENDIZAJE 2: FRAMEWORK DE PERSISTENCIA	
2.1 Tema 4 : Introducción al Framework Spring	105
2.1.1 : Instalación y Core	105
2.1.2 : Spring Boot	108
2.1.3 : Uso de plantillas	113
2.1.4 : Spring DATA (JPA - Hibernate)	121
2.1.5 : Web	126
2.1.6 : Transacciones	127
UNIDAD DE APRENDIZAJE 3: REPORTES CON JASPER Y DESPLIEGUE CON AZURE	
3.1 Tema 5 : JasperReports	136
3.1.1 : Diseño e implementación de reportes con Jasper Studio	136
3.1.2 : Integración JasperReports	145
3.1.3 : Reportes gráficos y resúmenes	147
3.2 Tema 6 : Azure	154
3.2.1 : Creación de cuenta educativa en Azure	154
3.2.2 : Configuración de entorno Azure	158
3.2.3 : Despliegue de aplicaciones web en Azure	165
Bibliografía	171

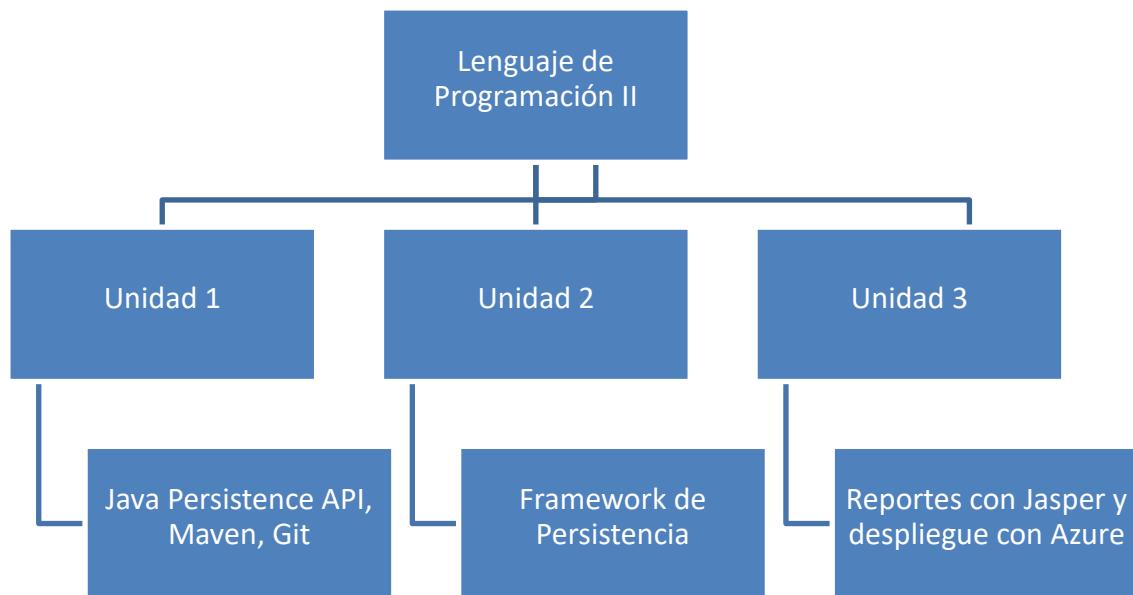
Presentación

El presente curso Lenguaje de Programación II, forma parte de la línea de programación de la carrera. En el presente curso lo que se busca es darle al estudiante las herramientas que le permitan implementar proyectos informáticos web, empleando hibernate, java, jpa, mysql, etc., y diversos frameworks de persistencia.

El manual está diseñado de acuerdo con el patrón de unidades de aprendizaje, que se desarrollan durante semanas específicas. En cada uno encontrarás los logros que deberás alcanzar al final de la unidad, los temas que se desarrollarán exhaustivamente, y lo que se desarrollará, los subtemas. Finalmente, encontrarás actividades para realizar en cada lección que te permitirán reforzar lo aprendido en clase.

En el curso se utilizarán herramientas que permitan cubrir todas las capacidades a lograr por parte de los estudiantes. La primera parte del curso usaremos ORM con la especificación de JPA donde abordaremos los diversos tipos de anotaciones. En la segunda parte del curso se cubrirá el uso del framework Spring.

Red de contenidos





Java Persistence API, Maven, Git

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno comprende e implementa consultas y mantenimientos de base de datos usando JPA, utiliza Maven para configurar proyectos y conoce el uso de repositorios.

TEMARIO

1.1 Tema 1 : Introducción al API de Persistencia JPA

- 1.1.1 : Introducción a JPA y componentes
- 1.1.2 : Entidades y anotaciones
- 1.1.3 : JPA Query Language
- 1.1.4 : JPA y relaciones

1.2 Tema 2 : Maven

- 1.2.1 : Introducción y configuración
- 1.2.2 : Creación de un proyecto
- 1.2.3 : Ciclo de vida
- 1.2.4 : Arquetipos
- 1.2.5 : Gestión de dependencias

1.3 Tema 3 : Git

- 1.3.1 : Introducción a los sistemas de control de versiones
- 1.3.2 : Gestión de escenarios
- 1.3.3 : GitHub básico

ACTIVIDADES PROPUESTAS

- Los alumnos examinan las clases de la API Persistencia JPA.
- Los alumnos implementan clases Java en Ide Eclipse.
- Los alumnos desarrollan aplicaciones Java haciendo uso de JPA.
- Los alumnos implementan y ejecutan diversas sentencias a la base de datos.

1.1. INTRODUCCIÓN AL API DE PERSISTENCIA JPA

JDBC (Database Connectivity) es una herramienta tecnológica ampliamente conocida que se usa frecuentemente para establecer la conexión entre una aplicación con una base de datos. También debemos tener las observaciones que menciona Tejaswini Mandar (2016): Los problemas con JDBC son los siguientes:

- En JDBC, los desarrolladores asignan un miembro de datos a una columna de tabla. Esto no es centrado en objetos.
- No hay una asignación predeterminada disponible en JDBC con la tabla.
- JDBC funciona utilizando lo básico de SQL.
- JDBC utiliza código específico de la base de datos.
- No está disponible el control automático de versiones ni el sellado de tiempo.

Mandar (2016) plantea que para superar estos inconvenientes es recomendable usar ORM:

"Para superar estos inconvenientes, los desarrolladores tienen la técnica de **Mapeo Relacional de Objetos (ORM)**. ORM ayuda a administrar el desajuste de impedancia entre una aplicación orientada a objetos y una base de datos relacional. ORM ayuda a escribir aplicaciones menos complejas. Con la ayuda de los marcos ORM, podemos persistir los objetos en las tablas relacionales utilizando el mapeo entre las tablas y los objetos. iBatis, JPA e Hibernate son las tecnologías ORM que hay en el mercado"

La historia de JPA se origina en dos frameworks de persistencia: en el lado propietario, existía TopLink mientras que en el lado "open" estaba Hibernate. JPA es una especificación basada en el JSR 220 conocido como "Enterprise Java Bean 3.0".

La API de persistencia de Java es un marco ligero basado en POJO para la persistencia de Java. Aunque el mapeo relacional de objetos es un componente importante de la API, también ofrece soluciones a los desafíos arquitectónicos de integrar la persistencia en aplicaciones empresariales escalables. Mike Keith (2018)

JPA no es un producto, sino solo una especificación que no puede realizar la persistencia por sí misma. JPA, por supuesto, requiere una base de datos para persistir.

Como lo mencionamos anteriormente, estas son algunas implementaciones de JPA:

- Hibernate: <http://www.hibernate.org/>
- TopLink: <http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>
- OpenJPA: <http://openjpa.apache.org/>
- EclipseLink: <http://www.eclipse.org/eclipselink/>

1.1.1. Introducción a JPA y componentes

El concepto de **Entidad** fue introducido por Peter Chen en un documento llamado "The Entity-relationship model – Howard a Unified View of Data" publicado en "ACM Transactions on Database Systems" en el año de 1976.

Javier Ceballos (2015) define la entidad (entity) como un objeto que tiene una vida corta en memoria, pero persiste en la base de datos; en contraposición, un objeto convencional simplemente reside en memoria. Diseñar una entidad no es más que escribir una clase con sus

atributos y métodos **get** y **set**, y, después, añadir la anotación **@Entity** y declarar uno de sus atributos como clave primaria con la anotación **@Id**. Por ejemplo:

Figura 1: Ejemplo de entidad con anotaciones

```
package model;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Users {

    @Id
    private int id;

    private String name;
    private String lastname;
    private String user;
    private String password;
}
```

Nota: Elaboración Propia

En la actualidad, dicha definición es vigente dado que cualquier objeto dentro de una aplicación JPA puede ser una entidad, hay que definir las características que debe poseer una Entidad. Mike Keith (2018) define las siguientes características en su libro “Pro JPA 2 in Java EE 8: An In-Depth Guide to Java Persistence APIs”:

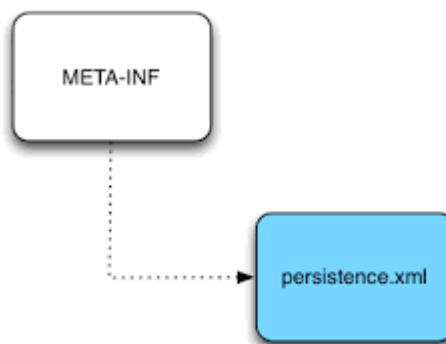
- **Persistencia:** La primera y más básica característica de las entidades es que son persistentes. Esto generalmente solo significa que se pueden hacer persistentes. Más específicamente, significa que su estado se puede representar en un almacén de datos y se puede acceder a él en un momento posterior, quizás mucho después del final del proceso que lo creó.
- **Identidad:** Como cualquier otro objeto de Java, una entidad tiene una identidad de objeto, pero cuando existe en la base de datos también tiene una identidad persistente. La identidad del objeto es simplemente la diferenciación entre los objetos que ocupan la memoria. La identidad persistente, o un identificador, es la clave que identifica de forma única una instancia de entidad y la distingue de todas las demás instancias del mismo tipo de entidad.
- **Transaccionalidad:** Las entidades podrían llamarse quasi-transaccionales. Aunque se pueden crear, actualizar y eliminar en cualquier contexto, estas operaciones normalmente se realizan dentro del contexto de una transacción porque se requiere una transacción para que los cambios se confirmen en la base de datos. Los cambios realizados en la base de datos tienen éxito o fallan atómicamente, por lo que la vista persistente de una entidad debería ser transaccional.

- **Granularidad:** Una buena manera de mostrar lo que son las entidades es describir lo que no son. No son primitivos, envoltorios primitivos ni objetos integrados con estado unidimensional. Estos no son más que escalares y no tienen ningún significado semántico inherente a una aplicación. Una cadena, por ejemplo, es un objeto demasiado detallado para ser una entidad porque no tiene ninguna connotación específica de dominio. Más bien, una cadena es adecuada y se usa muy a menudo como un tipo para un atributo de entidad y se le otorga un significado de acuerdo con el atributo de entidad que está escribiendo.
- **Metadata:** cada entidad tiene asociado una “metadata” que la describe. Dicha información puede estar almacenada dentro de la entidad Java o puede existir en un archivo externo: en ambos casos, esa información no se almacena en la base de datos.

Existen dos maneras de especificar la metadata:

- Usando Anotaciones.
- Usando XML.

Figura 2: Estructura de ubicación del archivo persistence.xml



Nota: Adaptado de Arquitectura Java, s.f., <https://www.arquitecturajava.com/>

Para aquellos que prefieren usar XML tradicional, esta opción todavía está disponible. Debería ser bastante sencillo cambiar al uso de descriptores XML después de haber aprendido y comprendido las anotaciones, porque el XML se ha modelado principalmente a partir de las anotaciones.

1.1.2. Entidades y anotaciones

Entorno de trabajo

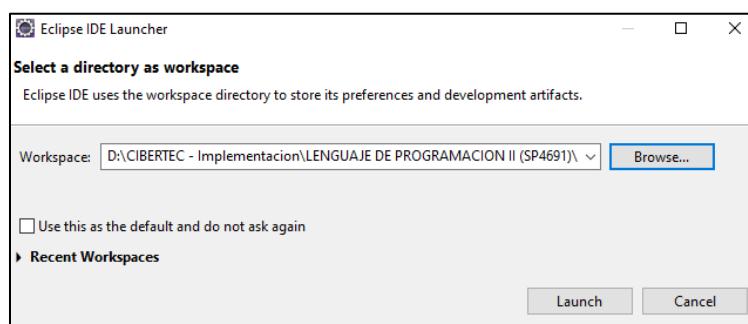
Realizaremos la creación de proyecto JPA usando el IDE Eclipse en la versión estable 2022:

Figura 3: Abrimos el aplicativo Eclipse 2022.12



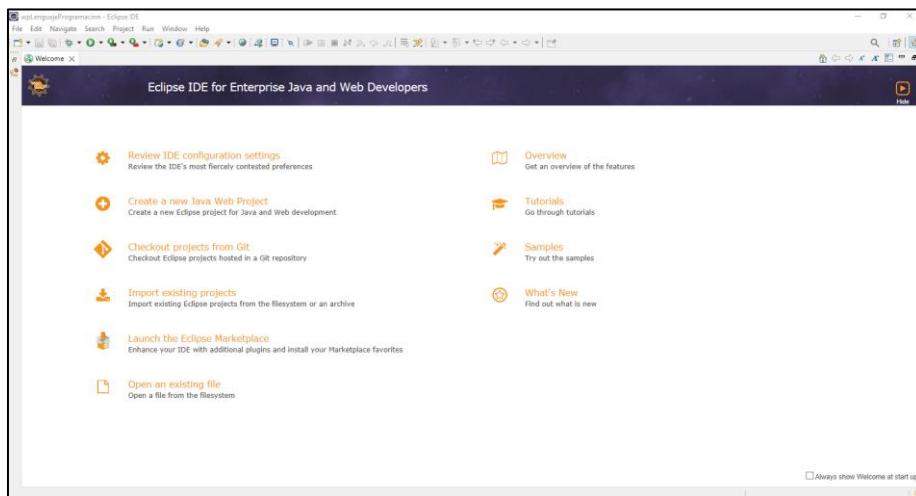
Nota: Elaboración Propia

Figura 4: Seleccionamos nuestro workspace



Nota: Elaboración Propia

Figura 5: Abrimos eclipse luego de seleccionar el workspace



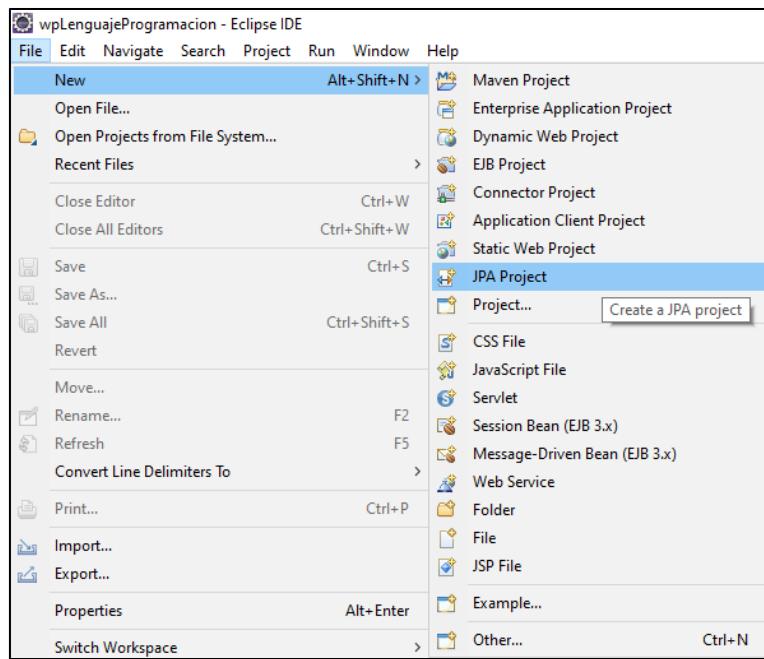
Nota: Elaboración Propia

Ejecuta el programa Eclipse, seleccionando la carpeta de trabajo a usar y en la pantalla principal, haz clic en **Create a JPA Project** (o usando el Menú File).

Creando un proyecto JPA

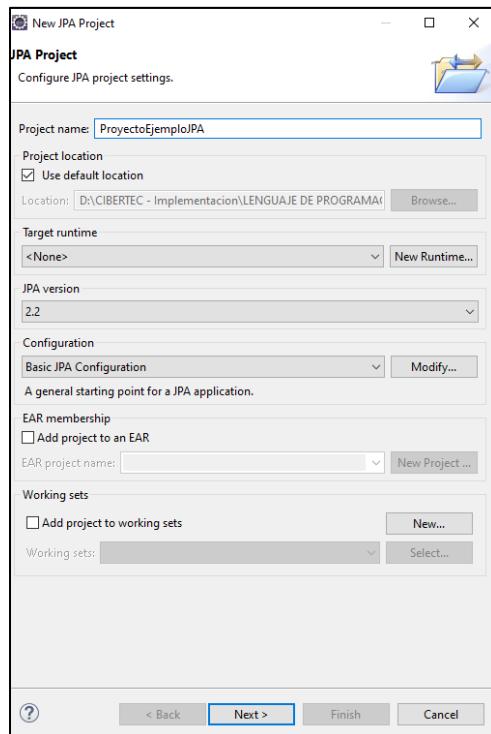
Paso 1. Selecciona **nuevo proyecto JPA**, escribiendo el nombre y configurando la versión (en este caso, usa la versión 2.2)

Figura 6: Seleccionando nuevo proyecto JPA



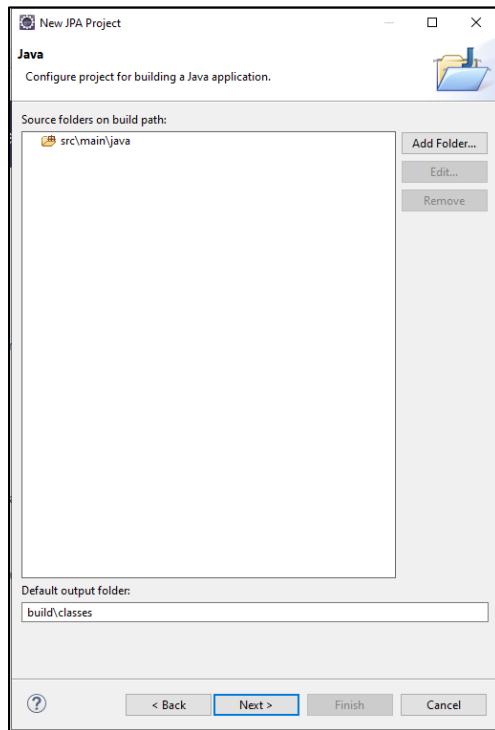
Nota: Elaboración Propia

Figura 7: Configurando el proyecto JPA



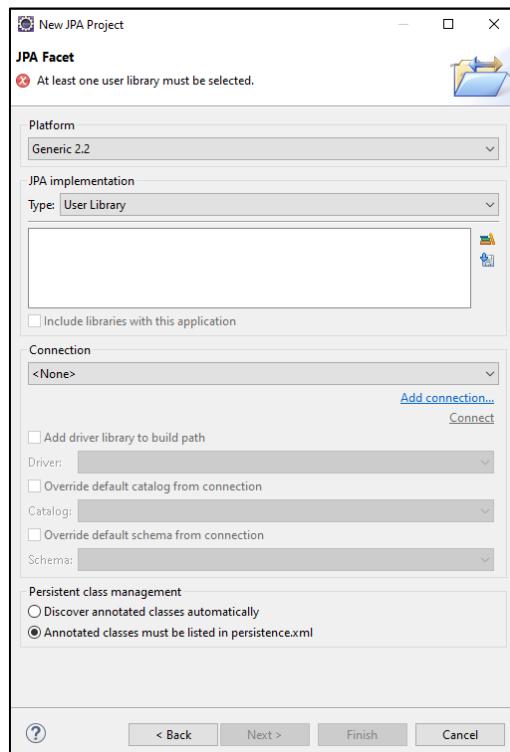
Nota: Elaboración Propia

Figura 8: Verificamos el source folders



Nota: Elaboración Propia

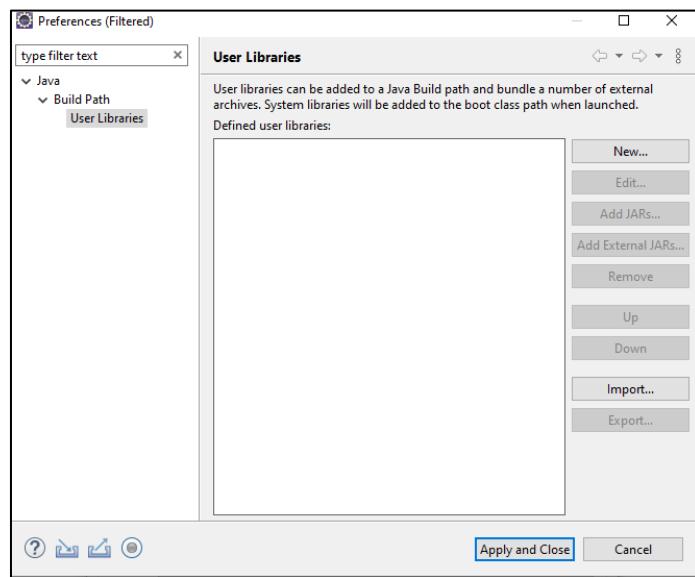
Figura 9: Seleccionamos la plataforma y la librería



Nota: Elaboración Propia

Paso 2. Creación de librería.

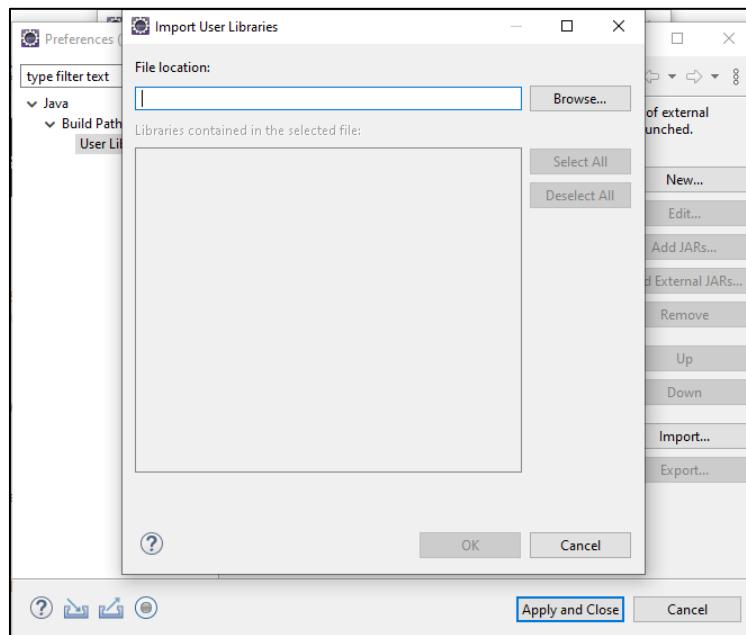
Figura 10: Mostrar pantalla de preferencias para importar librería



Nota: Elaboración Propia

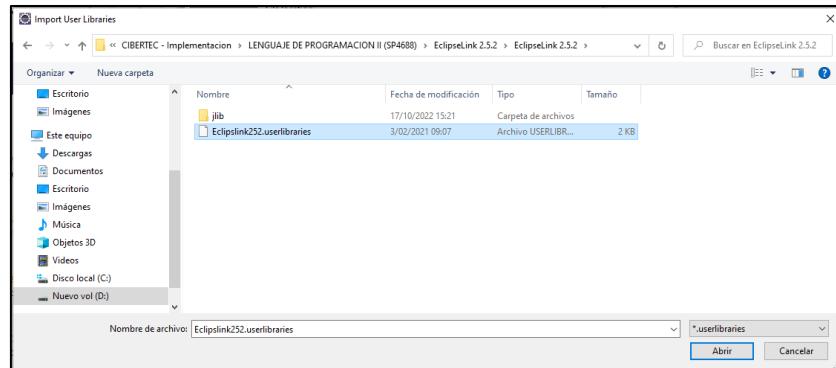
Importamos la librería y seleccionamos la ruta de la librería:

Figura 11: Pantalla para importar Mostrar pantalla de preferencias para librería



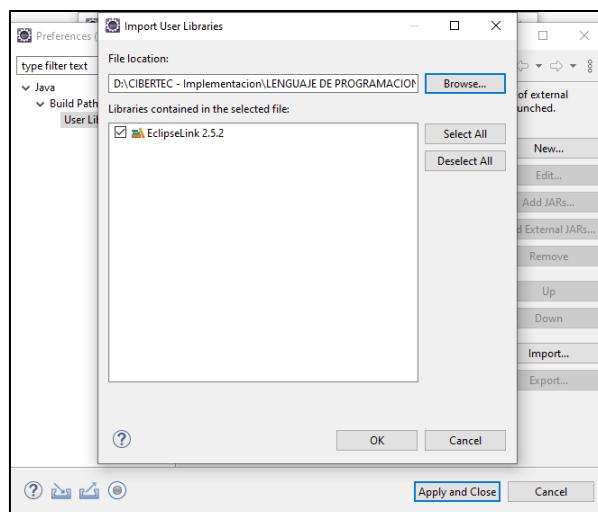
Nota: Elaboración Propia

Figura 12: En el directorio del pc seleccionamos la librería



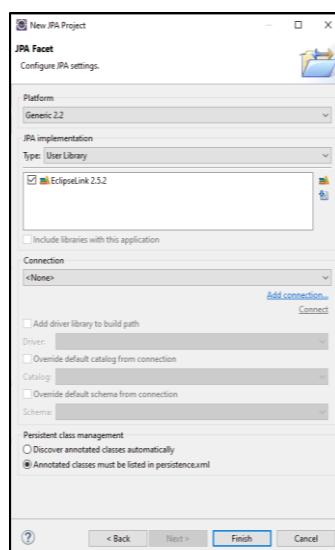
Nota: Elaboración Propia

Figura 13: En el directorio del pc seleccionamos la librería



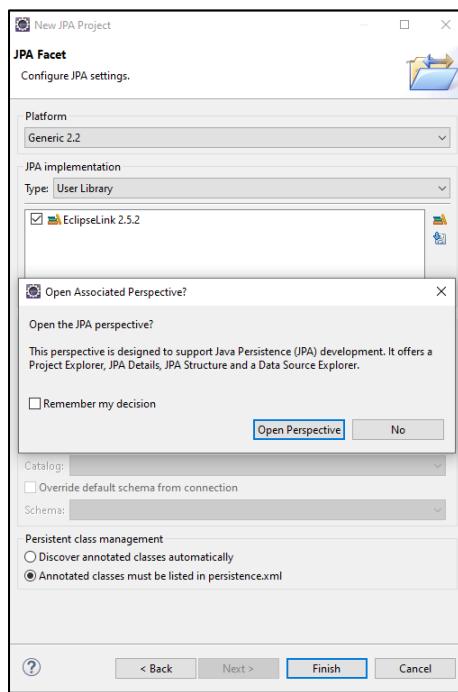
Fuente. - Elaboración Propia

Figura 14: Configuración completada para crear el proyecto



Nota: Elaboración Propia

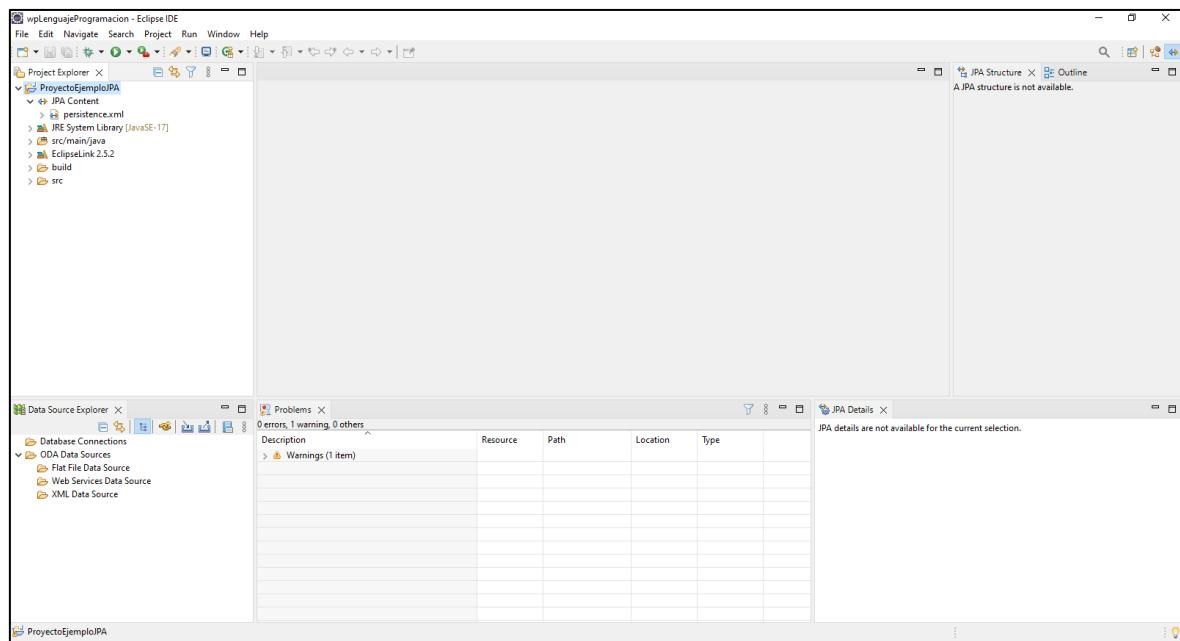
Figura 15: Abrimos en perspectiva JPA



Nota: Elaboración Propia

Paso 3. Se muestra la pantalla principal con el proyecto, donde podrás observar su estructura.

Figura 16: Pantalla principal de la IDE Eclipse con el nuevo proyecto



Nota: Elaboración Propia

Entidades

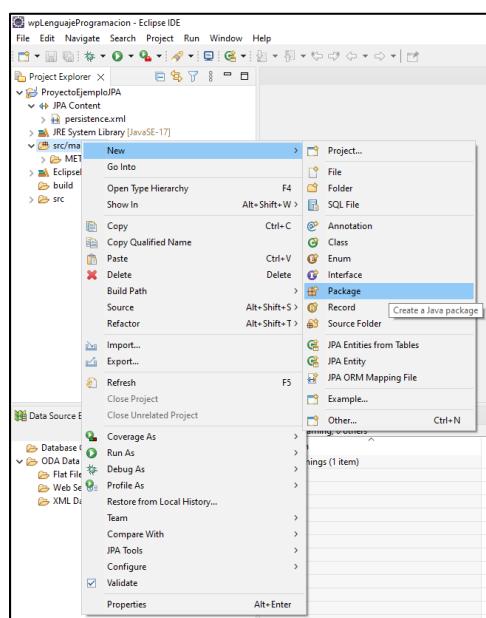
Las entidades son clases que permiten construir los objetos de transferencia de datos, es decir, de la base de datos al proyecto y viceversa.

IBM (2021) en la documentación oficial nos define una entidad: Una Entidad es un objeto de dominio persistente. Una entidad puede ser:

- **Clases abstractas o concretas.** Las entidades pueden ampliar tanto clases de entidad como no de entidad y las clases no de entidad pueden ampliar clases de entidad.

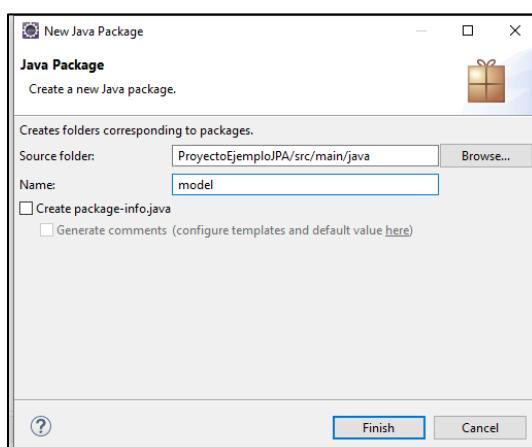
Ejemplo: Clase usuario (es importante crear los métodos de acceso get y set).

Figura 17: Pasos para crear un nuevo package para entidades



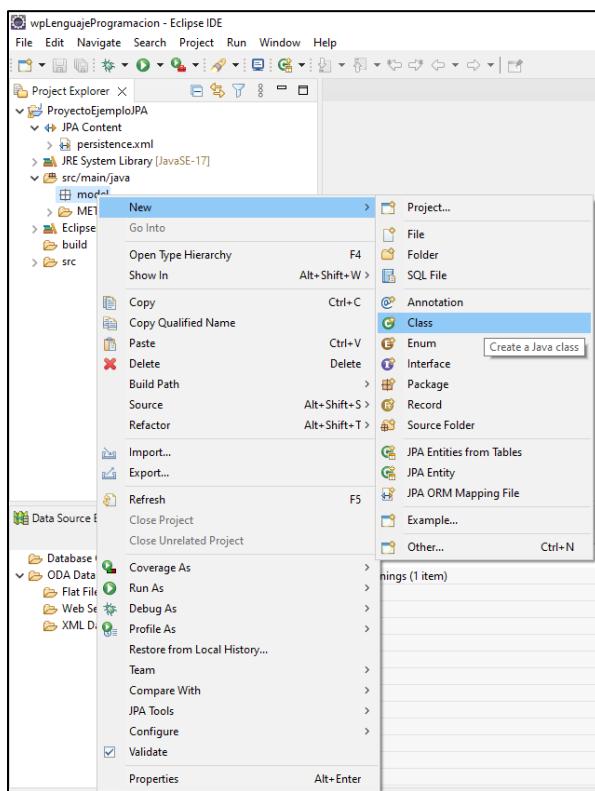
Nota: Elaboración Propia

Figura 18: Formulario para ingresar nombre del package



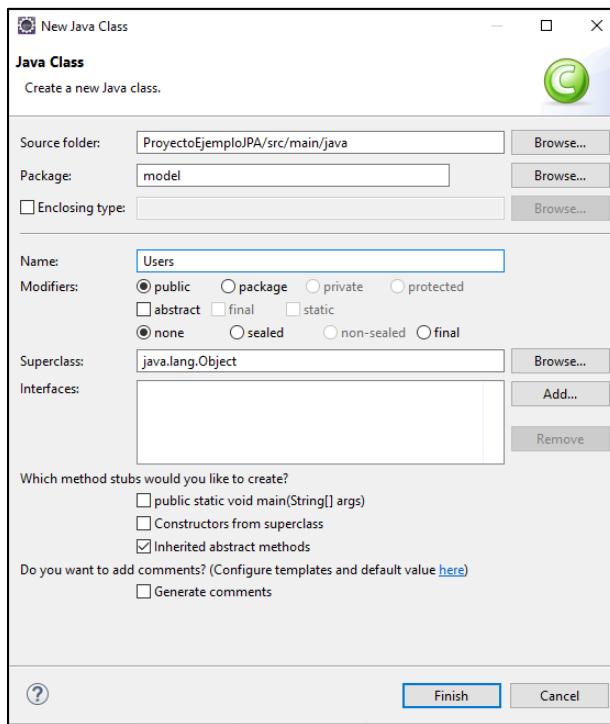
Nota: Elaboración Propia

Figura 19: Pasos para crear una nueva clase en el package entidades



Nota: Elaboración Propia

Figura 20: Formulario para colocar nombre de clases



Nota: Elaboración Propia

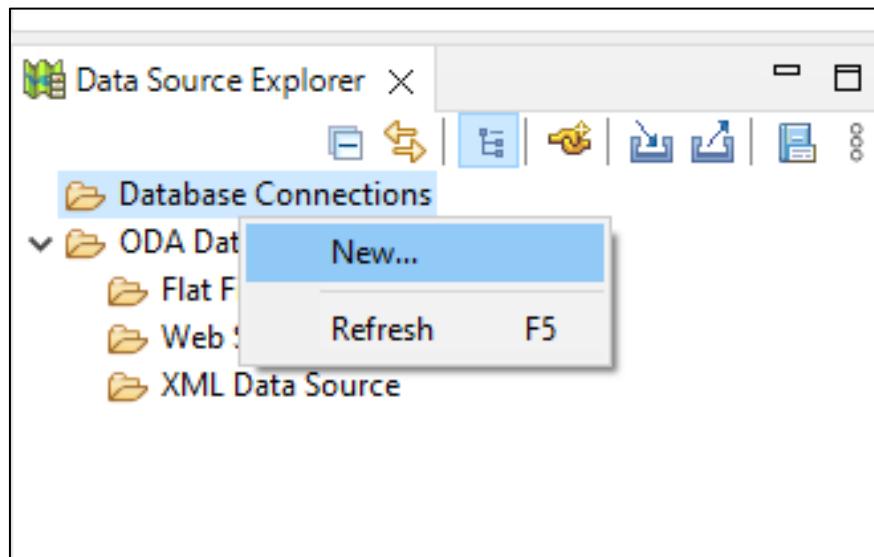
Figura 21: Creamos la entidad



Nota: Elaboración Propia

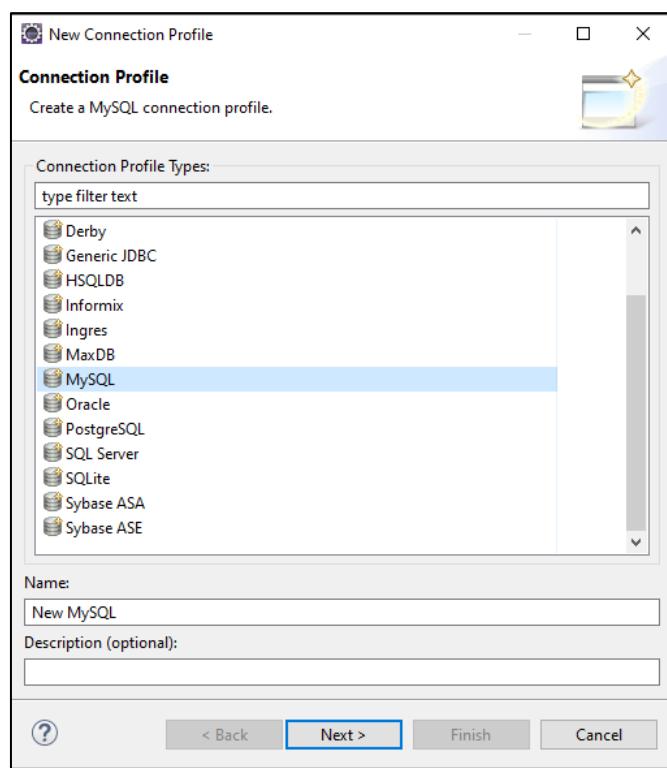
También, puedes crear la entidad utilizando una conexión con la base de datos, en ese caso, será necesario indicar los datos de conexión.

Figura 22: Creación de fuente de datos



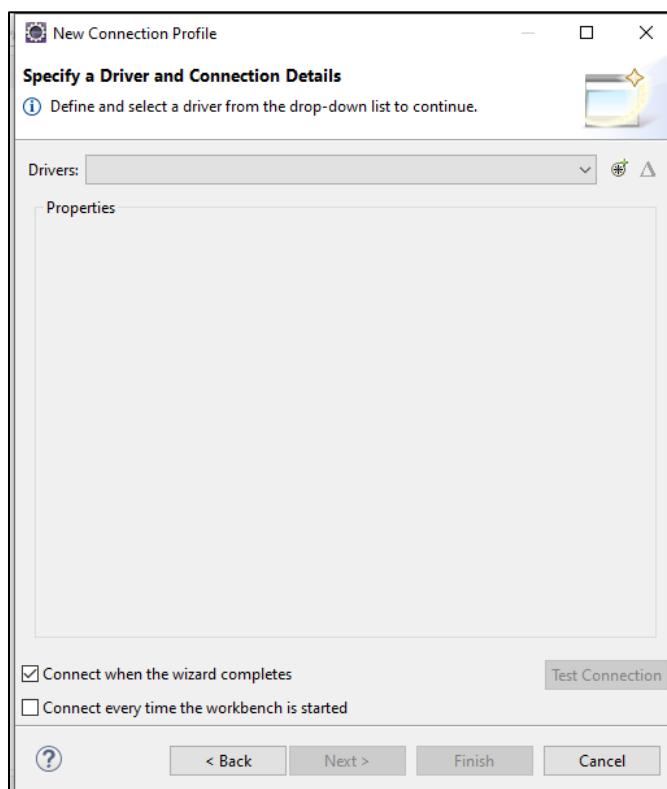
Nota: Elaboración Propia

Figura 23: Seleccionamos el Connection Profile



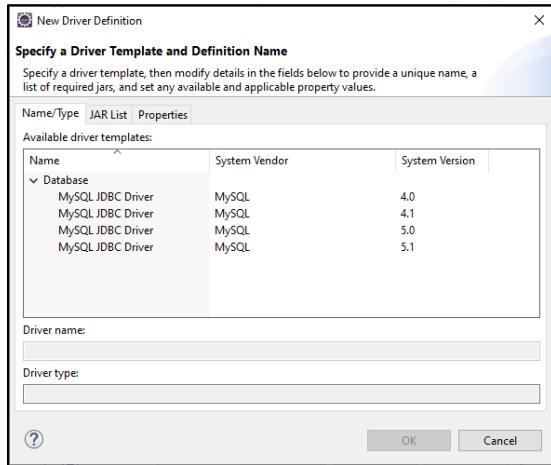
Nota: Elaboración Propia

Figura 24: Especificamos los detalles de la conexión



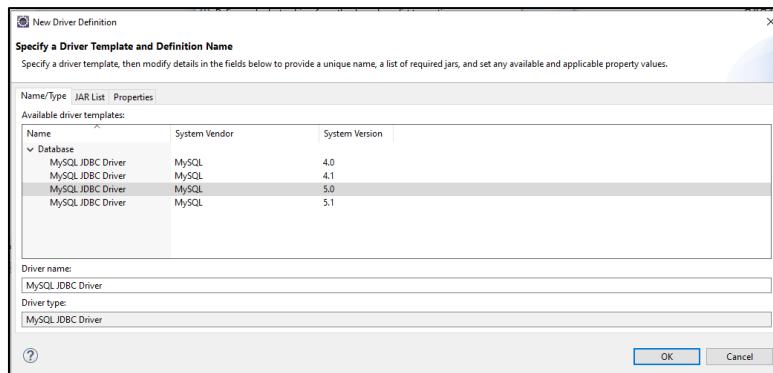
Nota: Elaboración Propia

Figura 25: Seleccionamos el tipo de controlador



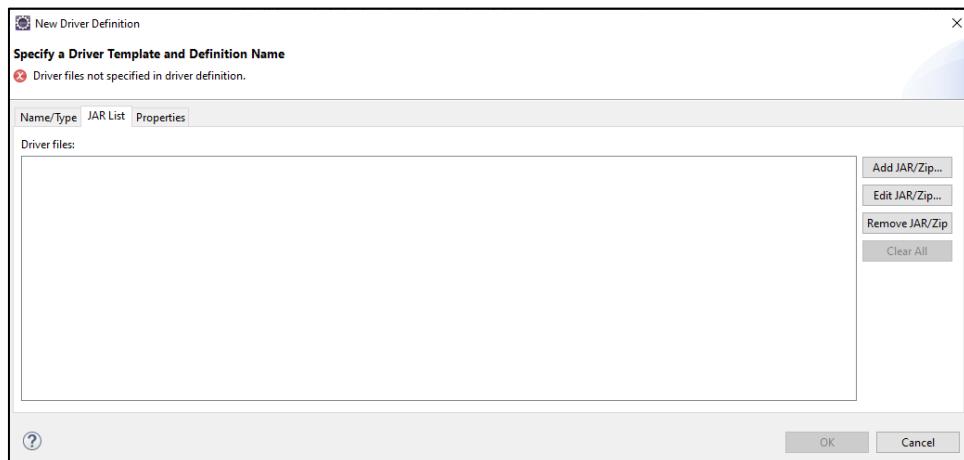
Nota: Elaboración Propia

Figura 26: Seleccionamos el tipo de controlador



Nota: Elaboración Propia

Figura 27: Seleccionamos el controlador



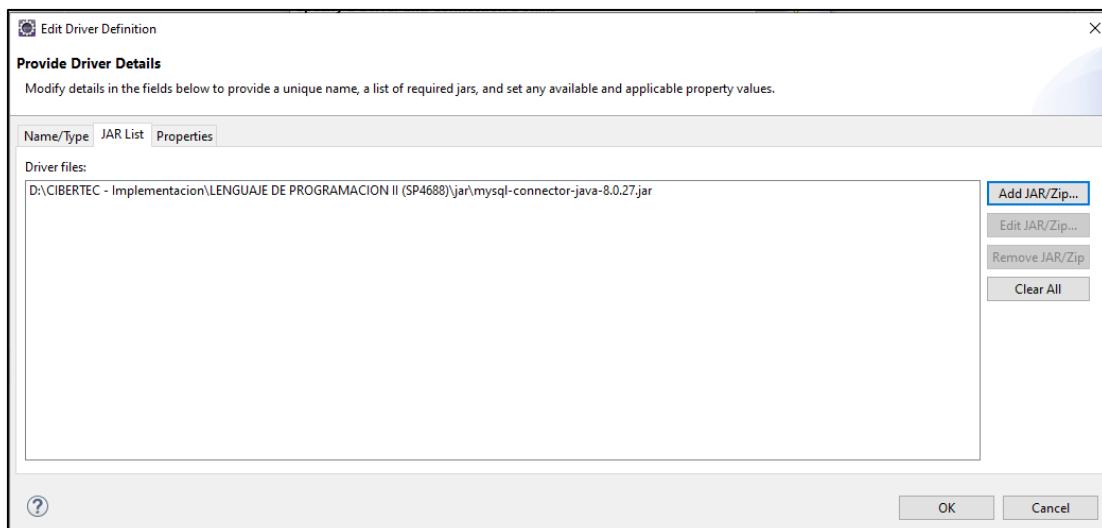
Nota: Elaboración Propia

Figura 28: Seleccionamos el controlador del equipo



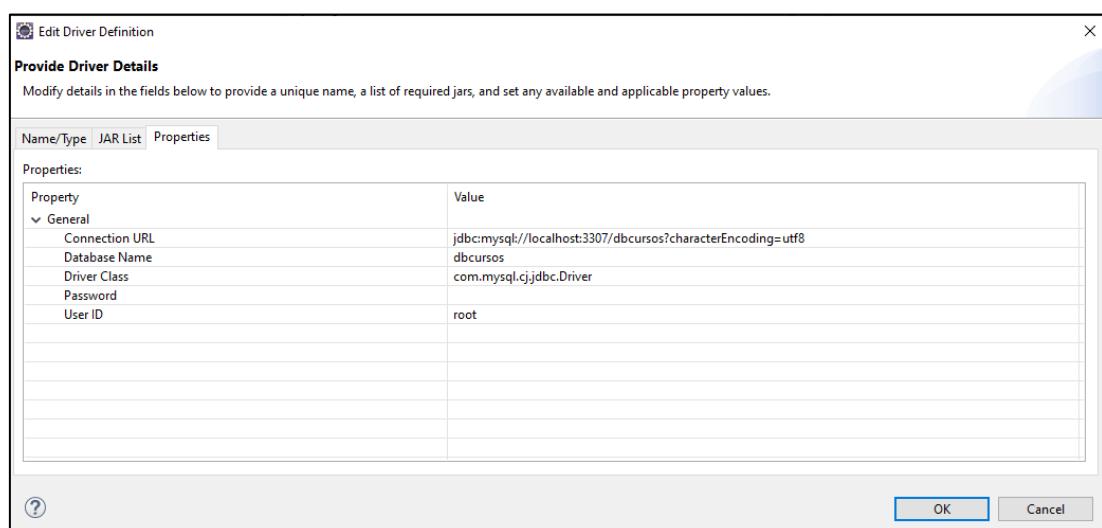
Nota: Elaboración Propia

Figura 29: Seleccionamos el controlador del equipo



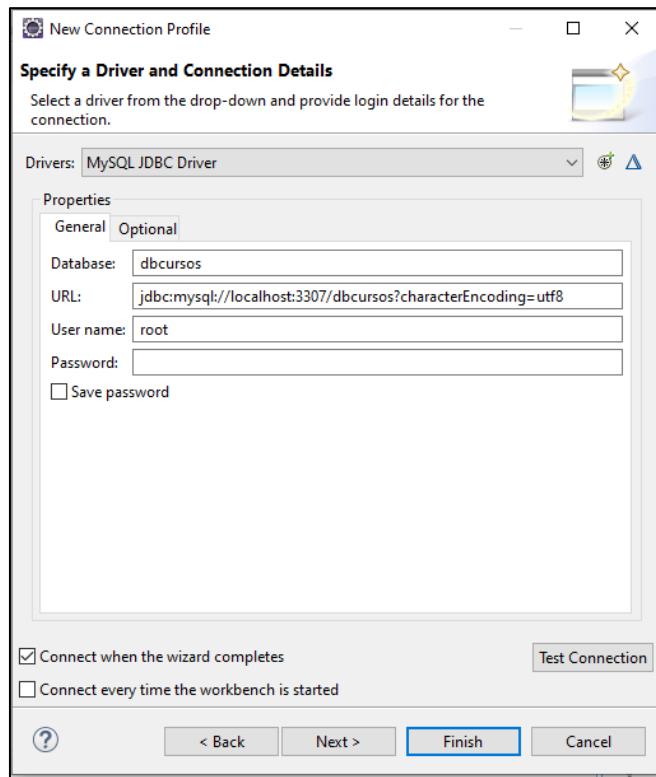
Nota: Elaboración Propia

Figura 30: Ingresamos la cadena de conexión, base de datos, usuario y clave



Nota: Elaboración Propia

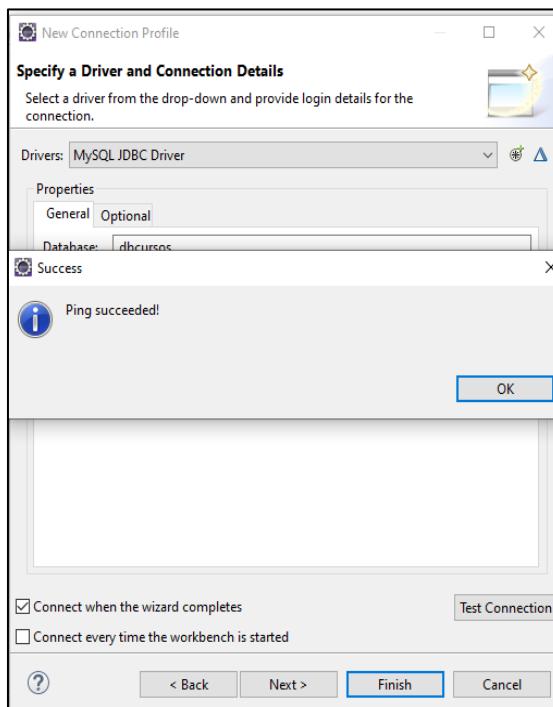
Figura 31: Verificamos que los datos ingresados son correctos



Nota: Elaboración Propia

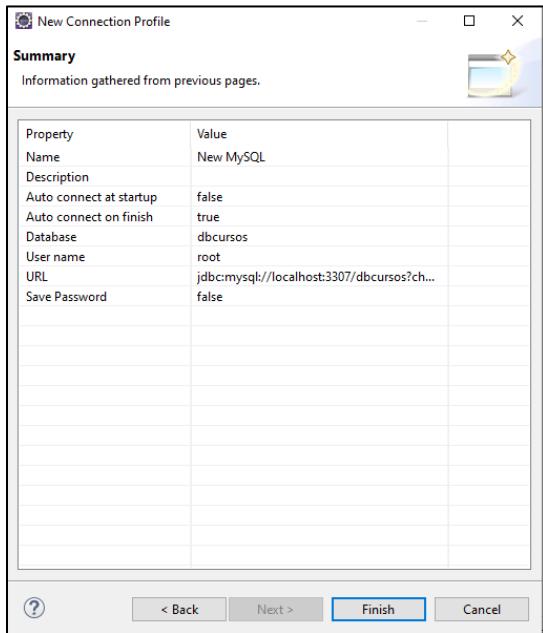
Prueba, si hay conexión.

Figura 32: Verificamos que la conexión es correcta



Nota: Elaboración Propia

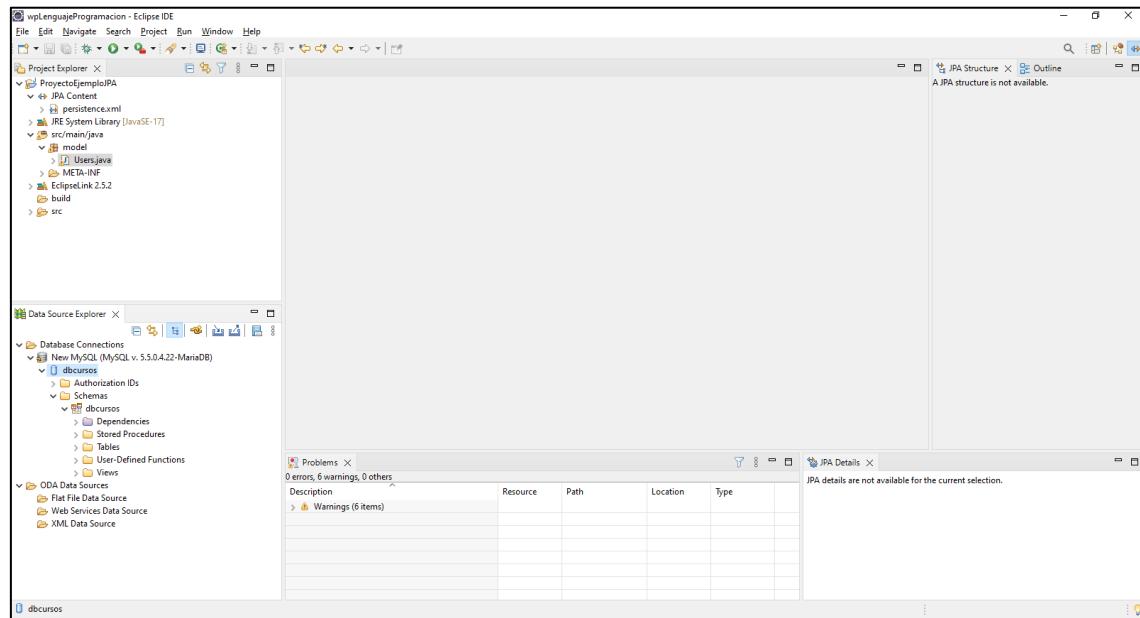
Figura 33: Creamos la cadena de conexión



Nota: Elaboración Propia

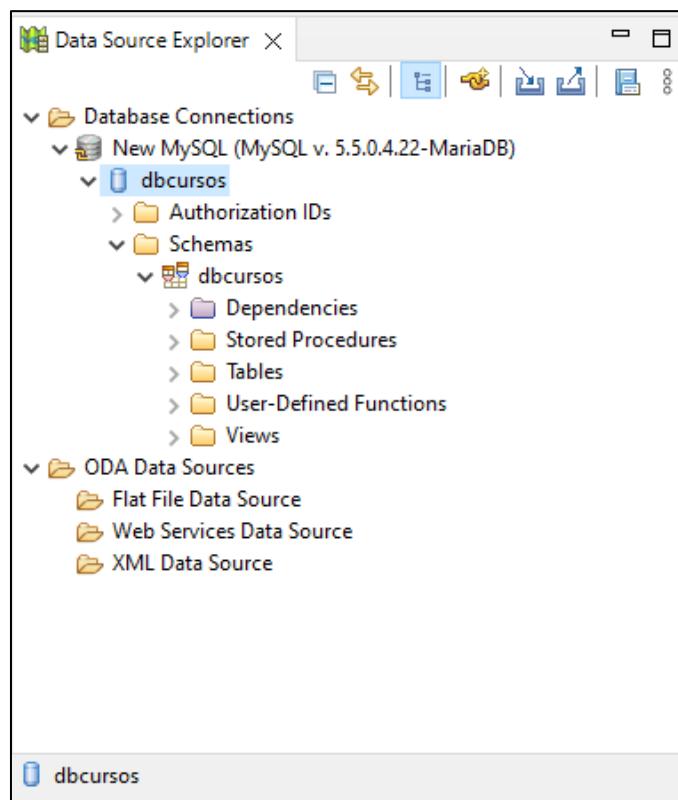
Quedará de la siguiente forma.

Figura 34: Listo el proyecto debería quedar la siguiente forma



Nota: Elaboración Propia

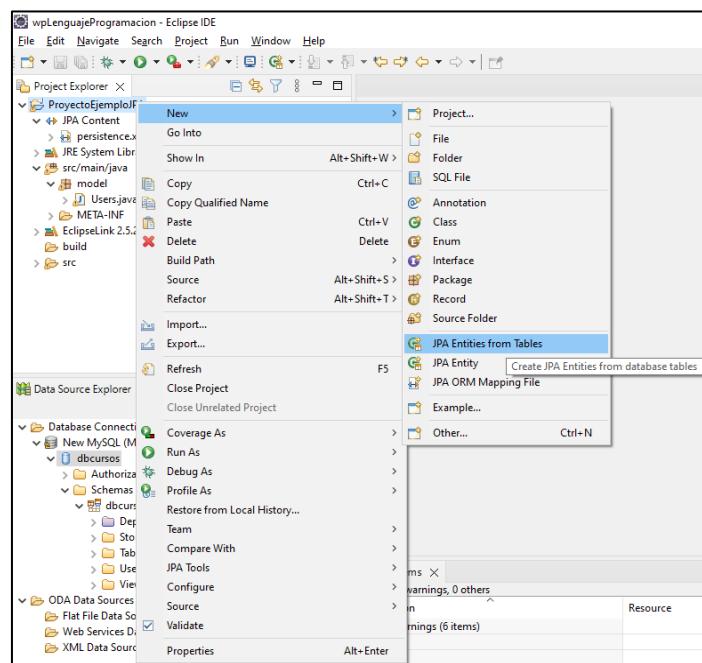
Figura 35: Explorador de fuentes de datos



Nota: Elaboración Propia

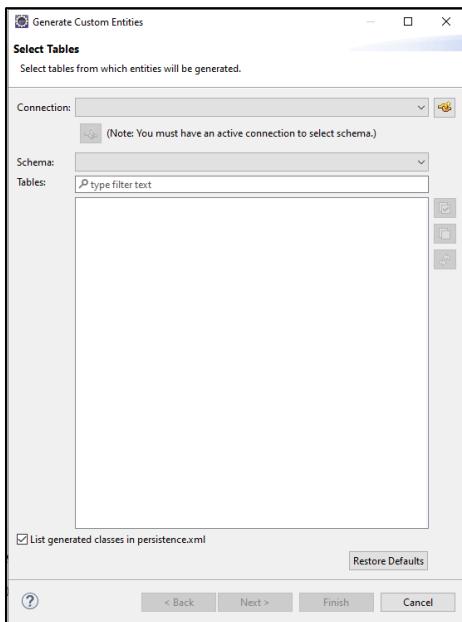
Creando entidades desde la conexión

Figura 36: Nueva entidad desde la fuente de datos



Nota: Elaboración Propia

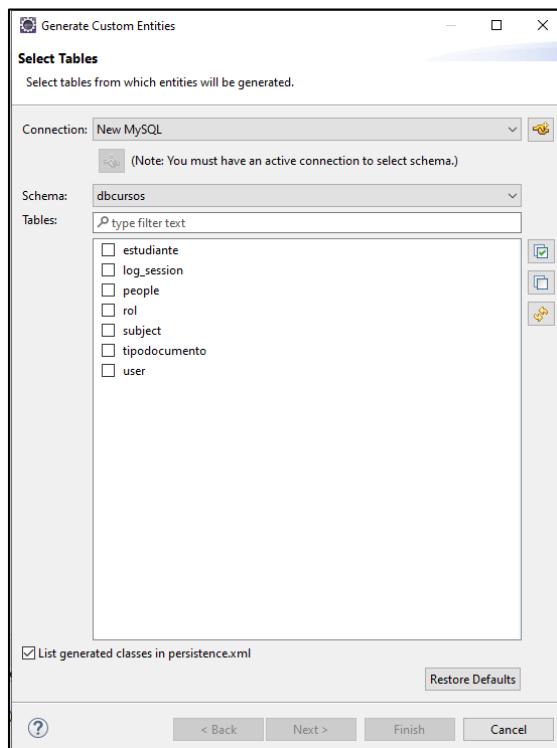
Figura 37: Nueva entidad desde la fuente de datos



Nota: Elaboración Propia

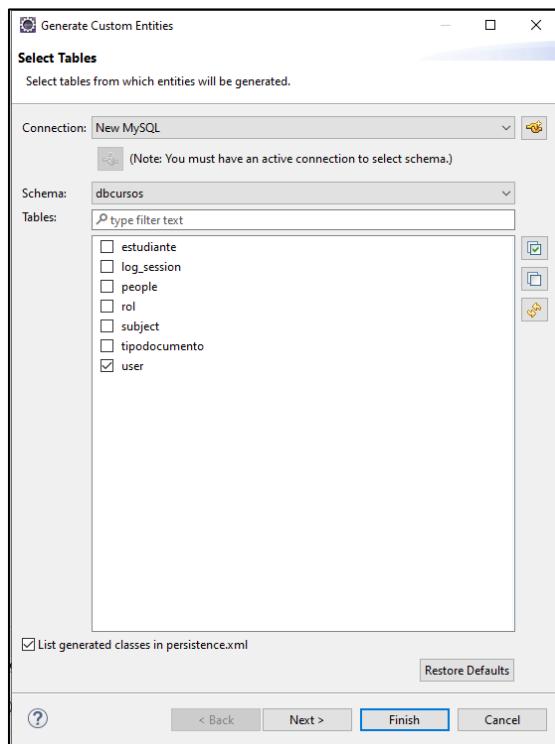
Al seleccionar la conexión, aparecerán las tablas de la conexión con la base de datos, selecciona la(s) tabla(s), donde podrás establecer las relaciones o configuraciones.

Figura 38: Seleccionamos la entidad



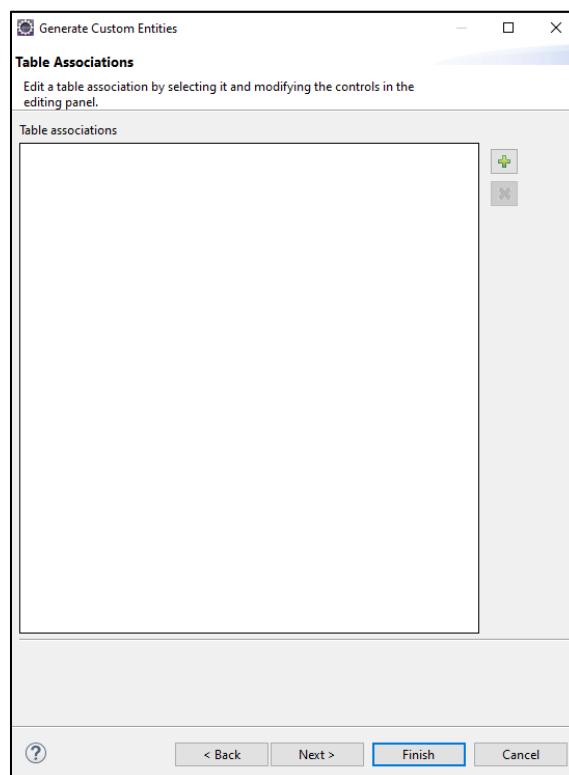
Nota: Elaboración Propia

Figura 39: Seleccionamos la entidad



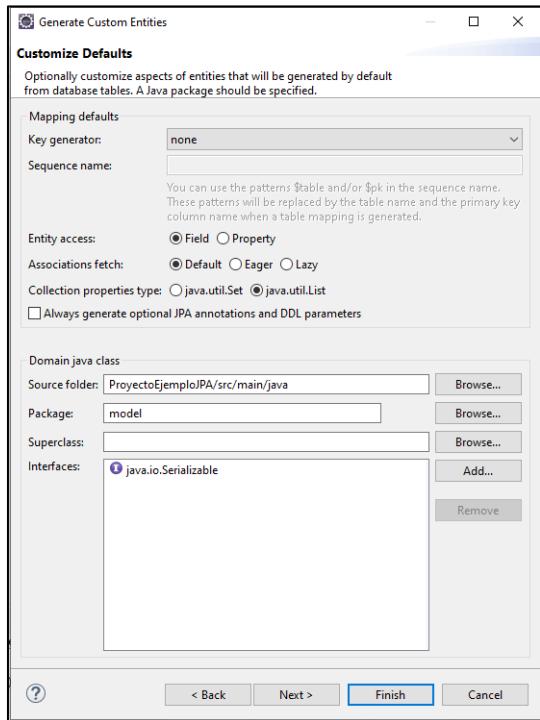
Nota: Elaboración Propia

Figura 40: Seleccionamos la entidad



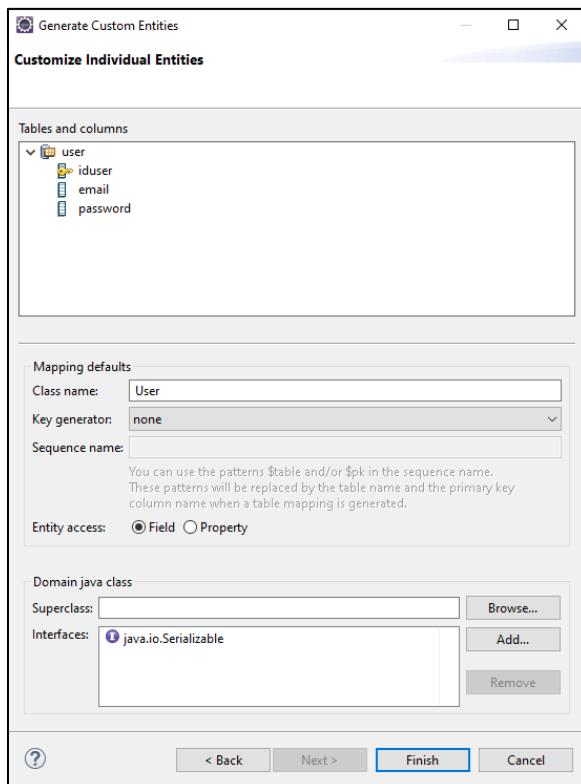
Nota: Elaboración Propia

Figura 41: Seleccionamos paquete y propiedades que tendrá la entidad



Nota: Elaboración Propia

Figura 42: Verificamos atributos que tendrá la entidad



Nota: Elaboración Propia

Finalmente, se creará la clase entidad.

```
@Entity  
@NamedQuery(name="User.findAll", query="SELECT u FROM User u")  
public class User implements Serializable {  
    private static final long serialVersionUID = 1L;  
  
    @Id  
    private int iduser;  
  
    private String email;  
  
    private String password;  
  
    @Column(name="people_idpeople")  
    private int peopleIdpeople;  
  
    @Column(name="rol_idrol")  
    private int roleIdrol;  
  
    public User() {  
    }  
  
    public int getIduser() {  
        return this.iduser;  
    }  
  
    public void setIduser(int iduser) {  
        this.iduser = iduser;  
    }
```

```
public void setIduser(int iduser) {  
    this.iduser = iduser;  
}  
  
public String getEmail() {  
    return this.email;  
}  
  
public void setEmail(String email) {  
    this.email = email;  
}  
  
public String getPassword() {  
    return this.password;  
}  
  
public void setPassword(String password) {  
    this.password = password;  
}  
  
public int getPeopleIdpeople() {  
    return this.peopleIdpeople;  
}  
  
public void setPeopleIdpeople(int peopleIdpeople) {  
    this.peopleIdpeople = peopleIdpeople;  
}
```

```
public void setEmail(String email) {
    this.email = email;
}

public String getPassword() {
    return this.password;
}

public void setPassword(String password) {
    this.password = password;
}

public int getPeopleIdpeople() {
    return this.peopleIdpeople;
}

public void setPeopleIdpeople(int peopleIdpeople) {
    this.peopleIdpeople = peopleIdpeople;
}

public int getRolIdrol() {
    return this.rolIdrol;
}

public void setRolIdrol(int rolIdrol) {
    this.rolIdrol = rolIdrol;
}
```

En el archivo **persistencia** (persistence.xml) de la carpeta **META-INF**, se mostrará la información de la clase creada.



Si el archivo no se crea, lo puedes configurar utilizando:

Figura 43: Plantilla del archivo persistence

Java Persistence API: XML Schemas																
Contents																
<ul style="list-style-type: none">IntroductionUsing Java Persistence SchemasJava Persistence 2.2 Schema ResourcesJava Persistence 2.1 Schema ResourcesJava Persistence 2.0 Schema ResourcesJava Persistence 1.0 Schema Resources																
<h3>Introduction</h3> <p>This document lists the document formats that will be used by the Java Persistence API XML descriptors. The Java Persistence API requires that its XML descriptors be validated with respect to the XML Schema listed by this document.</p>																
<h3>Using Java Persistence Schemas</h3> <p>Starting with the 2.1 version, the Java Persistence API Schemas share the namespace, http://xmlns.jcp.org/xml/ns/persistence/. Previous versions used the namespace http://java.sun.com/xml/ns/persistence/. Each schema document contains a version attribute that contains the version of the Java Persistence specification. This pertains to the specific version of the specification as well as the schema document itself.</p> <p>Each Java Persistence XML Schema document's file name contains the specific version of the Java Persistence specification to which it relates. This is introduced for convenience to locate specific versions of the schema documents. However, XML descriptor instances are not required to refer to a specific file. Instead, an instance must specify the version of the specification by using the version attribute. This allows XML descriptor processors to use the version information to choose the appropriate version of the schema document(s) to process XML descriptor instances.</p>																
<h3>Java Persistence 2.2 Schema Resources</h3> <p>This table contains the XML Schema components for Java Persistence 2.2.</p> <table border="1"><thead><tr><th>Date Published</th><th>File Name</th><th>Description</th><th>Status</th></tr></thead><tbody><tr><td>August 4, 2017</td><td>orm_2_2.xsd</td><td>Object/relational mapping file schema</td><td>Final Release</td></tr><tr><td>August 4, 2017</td><td>persistence_2_2.xsd</td><td>Persistence configuration schema</td><td>Final Release</td></tr></tbody></table>					Date Published	File Name	Description	Status	August 4, 2017	orm_2_2.xsd	Object/relational mapping file schema	Final Release	August 4, 2017	persistence_2_2.xsd	Persistence configuration schema	Final Release
Date Published	File Name	Description	Status													
August 4, 2017	orm_2_2.xsd	Object/relational mapping file schema	Final Release													
August 4, 2017	persistence_2_2.xsd	Persistence configuration schema	Final Release													

Nota: Adaptado de Java Persistence API: XML Schemas, s.a., s.f.,
<https://www.oracle.com/webfolder/technetwork/jsc/xml/ns/persistence/index.html>

Anotaciones

“Los metadatos de anotación son una característica del lenguaje introducida en Java SE 5 que permite adjuntar metadatos estructurados y escritos al código fuente. Aunque JPA no requiere anotaciones, son una forma conveniente de aprender y usar la API” (Mike, 2018)

“EJB 3.1 y la API de persistencia Java™ (JPA) utilizan anotaciones de metadatos, que es un elemento que apareció en J2SE 5.0. Una anotación consta del signo @ precediendo a un tipo de anotación, algunas veces seguido por una lista entre paréntesis de pares elemento-valor. La especificación EJB 3.1 define varios tipos de anotación” (IBM, 2021)

- Anotación que define el componente, tal como **@Stateless**, que especifica el tipo de bean.
- **@Remote** y **@Local** especifican si un bean es accesible de forma remota o local.
- **@TransactionAttribute** especifica atributos de transacción.
- **@MethodPermissions**, **@Unchecked** y **@SecurityRoles** especifican permisos de seguridad y de método.

La API de persistencia Java añade anotaciones específicas de la creación de entidades, por ejemplo:

- **@Entity** es una anotación que define el componente y que especifica que una clase es una entidad.
- **@Table** especifica el origen de datos que se debe utilizar en la clase.
- **@MethodPermissions**, **@Unchecked** y **@SecurityRoles** especifican permisos de seguridad y de método.

El uso de las anotaciones requiere que se importe el paquete “javax.persistence.*” dentro de la clase Java que representa a la Entidad.

El uso de XML es una opción a las anotaciones, aunque su lectura puede resultar compleja para proyectos grandes.

Oracle (2023) nos dice que las anotaciones tienen varios usos, entre ellos:

- **Información para el compilador:** el compilador puede utilizar las anotaciones para detectar errores o suprimir advertencias.
- **Procesamiento en tiempo de compilación y tiempo de implementación:** las herramientas de software pueden procesar información de anotaciones para generar código, archivos XML, etc.
- **Procesamiento en tiempo de ejecución:** algunas anotaciones están disponibles para ser examinadas en tiempo de ejecución.

Las anotaciones pueden clasificarse en dos grupos:

- **Anotaciones lógicas:** describen el modelo de entidades desde el punto de vista del modelamiento orientado a objetos. Constituyen una especie de metadata del modelo.
- **Anotaciones físicas:** están relacionadas con el modelo en la base de datos (modelo físico) y tienen que ver con tablas, columnas, etc.

Las anotaciones dentro de una clase Java se pueden colocar a nivel de atributos o a nivel de métodos. Si se colocan a nivel de atributos se denomina “Field Access”, mientras que si se coloca a nivel de métodos se denomina “Property Access”.

En la especificación de JPA 2.0 se introduce la anotación @Access que permite combinar los dos modos presentados en el ejemplo. Esta anotación permite sobrescribir el modo de acceso por defecto, aunque no es muy usual hacerlo.

Para definir una entidad basta con emplear la anotación @Entity y la anotación @Id.

Se puede generar la entidad mapeando los datos, lo que permitirá crear las anotaciones.

Field Access:

“Anotar los campos de la entidad hará que el proveedor utilice el acceso a los campos para obtener y establecer el estado de la entidad. Los métodos getter y setter pueden o no estar presentes, pero si están presentes, el proveedor los ignora. Todos los campos deben declararse como protected, package o private.” (Mike, 2013)

Es importante tener presente que: “La anotación @Id indica no solo que el campo id es el identificador persistente o la clave principal de la entidad, sino que también se debe asumir el acceso al campo. Los campos de nombre y salario se vuelven persistentes de forma predeterminada y se asignan a columnas con el mismo nombre.” (Mike, 2013)

```
package model;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Estudiante {

    @Id
    private int idEstudiante;
    private String nombresApellidos;
    private String carrera;
    private String telefono;
}
```

Property Access:

“Cuando se utiliza el modo de acceso a la propiedad, se aplica el mismo contrato que para JavaBeans, y debe haber métodos getter y setter para las propiedades persistentes. El tipo de propiedad está determinado por el tipo de retorno del método getter y debe ser el mismo que el tipo del único parámetro pasado al método setter. Ambos métodos deben ser de visibilidad pública o protegida. Las anotaciones de asignación para una propiedad deben estar en el método getter.” (Mike, 2013)

```
@Entity
public class Estudiante {

    private int idEstudiante;
    private String nombresApellidos;
    private String carrera;
    private String telefono;

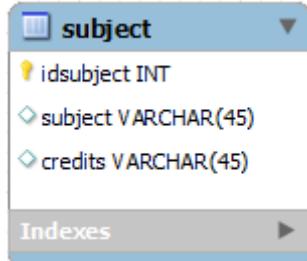
    @Id
    public int getIdEstudiante() {
        return idEstudiante;
    }
    public void setIdEstudiante(int idEstudiante) {
        this.idEstudiante = idEstudiante;
    }
    public String getNombresApellidos() {
        return nombresApellidos;
    }
    public void setNombresApellidos(String nombresApellidos) {
        this.nombresApellidos = nombresApellidos;
    }
    public String getCarrera() {
        return carrera;
    }
    public void setCarrera(String carrera) {
        this.carrera = carrera;
    }
    public String getTelefono() {
        return telefono;
    }
    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }
}
```

Anotación @Table

Por defecto, no es necesario incluir ninguna anotación para referenciar a una tabla. JPA asume que la tabla se llama igual que la clase Java en donde se define la entidad.

“En esos casos, el nombre de tabla predeterminado, que era simplemente el nombre no calificado de la clase de entidad, era perfectamente adecuado. Si sucede que el nombre de la tabla por defecto no es el nombre que le gusta, o si ya existe una tabla adecuada que contiene el estado en su base de datos con un nombre diferente, debe especificar el nombre de la tabla. Para ello, anote la clase de entidad con la anotación @Table e incluya el nombre de la tabla mediante el elemento de nombre. Muchas bases de datos tienen nombres breves para las tablas.” (Mike, 2013)

Figura 44: Primer ejemplo de anotación @Table



```

package model;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "subject")
public class Curso {

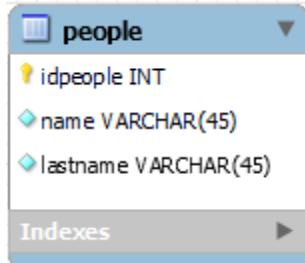
    @Id
    private int idSubject;

}

```

Nota: Elaboración Propia

Figura 45: Segundo ejemplo de anotación @Table



```

package model;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "people")
public class Persona {

    @Id
    private int idPeople;

    private String name;
    private String lastname;

}

```

Nota: Elaboración Propia

Anotación @Basic

“Los tipos simples de Java se asignan como parte del estado inmediato de una entidad en sus campos o propiedades. La lista de tipos persistentes es bastante larga e incluye prácticamente todos los tipos integrados que le gustaría conservar. “(Mike, 2013)

Mike (2013) incluyen los siguientes:

Tipos primitivos	byte, int, short, long, boolean, char, float, double
Clases que encapsulan a tipos primitivos	Byte, Integer, Short, Long, Boolean, Character, Float, Double
Arreglos de bytes y caracteres	byte[], Byte[], char[], Character[]
Números	java.math.BigInteger, java.math.BigDecimal
Cadenas de caracteres	java.lang.String
Tipos de datos que manejan fechas	java.util.Date, java.util.Calendar
Tipos de datos que manejan fecha JDBC	java.sql.Date, java.sql.Time, java.sql.Timestamp
Tipos enumerados	Cualquiera
Objetos serializables	Cualquiera

La anotación @Basic (que es opcional) se utiliza para indicar de forma explícita que dicho atributo debe ser almacenado en la base de datos.

“Se puede colocar una anotación @Basic opcional en un campo o propiedad para marcarlo explícitamente como persistente. Esta anotación es principalmente para fines de documentación y no es necesaria para que el campo o la propiedad sean persistentes. Si no está allí, se asume implícitamente en ausencia de cualquier otra anotación de mapeo. Debido a la anotación, las asignaciones de tipos simples se denominan asignaciones básicas, ya sea que la anotación @Basic esté realmente presente o simplemente se suponga. ”(Mike, 2013)

Anotación @Transient

Se emplea para marcar aquellos atributos de la entidad que NO deben ser guardados en la base de datos.

Figura 46: Primer ejemplo de anotación @Transient

```

package model;

import java.sql.Date;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;

@Entity
@Table(name = "people")
public class Persona {

    @Id
    private int idPeople;

    private String name;
    private String lastname;

    @Transient
    private Date lastLogin;

}

```

Nota: Elaboración Propia

Si serializamos este objeto *Person* y lo pasamos a otro servicio en nuestro sistema, el campo *lastLogin* se incluirá en la serialización. Si no desea incluir *lastLogin*, se reemplaza la anotación *@Transient* con la palabra *transient*:

```

@Entity
@Table(name = "people")
public class Persona implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    private int idPeople;

    private String name;
    private String lastname;

    private transient Date lastLogin;

}

```

Anotación @Column

Es una anotación de tipo físico, pues indica las características físicas de la columna en la base de datos.

Si no se especifica para un atributo determinado marcado como persistente, JPA asume que la columna se llama igual que dicho atributo. En cambio, si la columna tiene un nombre diferente, se deberá especificar con el uso de la anotación `@Column`.

“La anotación `@Basic` (o el mapeo básico asumido en su ausencia) se puede considerar como una indicación lógica de que un atributo determinado es persistente. La anotación física que es la anotación complementaria de la asignación básica es la anotación `@Column`. Especificar `@Column` en el atributo indica características específicas de la columna de la base de datos física que preocupan menos al modelo de objetos. De hecho, es posible que el modelo de objetos ni siquiera necesite saber a qué columna está asignado, y el nombre de la columna y los metadatos de asignación física se pueden ubicar en un archivo XML separado.”(Mike, 2013)

Los elementos que acompañan a la anotación `@Column` son los siguientes:

Elemento	Descripción	Valor por defecto
<code>String columnDefinition</code>	(Opcional) Es el fragmento de SQL utilizado para generar el DDL de la columna (depende del manejador de base de datos)	””
<code>boolean insertable</code>	(Opcional) Indica si la columna ser incluirá dentro de una sentencia SQL <code>INSERT</code> , generada por el Persistence Provider.	true
<code>int length</code>	(Opcional) Indica la longitud de la columna en la tabla y funciona únicamente cuando la columna es un <code>String</code> o cadena de caracteres.	255
<code>String name</code>	(Opcional) Indica el nombre de la columna. Por defecto, se asume que la columna se llama igual que el atributo de la entidad.	””
<code>boolean nullable</code>	(Opcional) Indica si la columna permite valores nulos.	true
<code>int precision</code>	(Opcional) Indica la precisión para una columna numérica (válido solo para columnas decimales).	0
<code>int scale</code>	(Opcional) Indica la escala para una columna numérica (válido solo para columnas decimales).	0
<code>String table</code>	(Opcional) Indica el nombre de la tabla en donde se asocia la columna.	””
<code>boolean unique</code>	(Opcional) Se emplea cuando la clave única corresponde a una sola columna.	false
<code>boolean updatable</code>	(Opcional) Indica si la columna ser incluirá dentro de una sentencia SQL <code>UPDATE</code> generada por el Persistence Provider.	True

```
package model;

import java.sql.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;

@Entity
@Table(name = "people")
public class Persona {

    @Id
    @Column(name = "idpeople")
    private int id;

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

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

    @Transient
    private Date lastLogin;

}

package model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "subject")
public class Curso {

    @Id
    @Column(name = "idSubject")
    private int id;

    @Column(name = "teacher")
    private String profesor;

    @Column(name = "level")
    private String nivel;

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

    @Column(name = "code")
    private String codigo;
}
```

“Tenga en cuenta que @Column se puede usar con asignaciones de @Id, así como con asignaciones básicas. El campo de identificación se anula para que se asigne a la columna de identificación en lugar de a la columna de identificación predeterminada. El campo de nombre no está anotado con @Column, por lo que el nombre de columna predeterminado NOMBRE se usará para almacenar y recuperar el nombre del empleado. ”(Mike, 2013)

Anotación @Lob

Para el manejo de objetos binarios (imágenes o archivos, generalmente) se requieren accesos especiales en el driver JDBC para efectuar conversiones entre el objeto Java y la columna en la tabla de la base de datos.

“La anotación @Lob actúa como anotación de marcador para cumplir con este propósito y puede aparecer junto con la anotación @Basic, o puede aparecer cuando @Basic está ausente y se supone implícitamente que está en la asignación. ”(Mike, 2013)

Ahora bien, los campos LOB (acrónimo de Large Object) se pueden clasificar de dos maneras, siendo el manejo de cada manera un tanto diferente.

Si el objeto es ...	El tipo java a usar es ...
Character Large Objets (CLOB)	char[], Character[], String
Binary Large Objects (BLOB)	byte[], Byte[], tipos serializables

En ambos casos, el driver JDBC es responsable de hacer las conversiones entre el objeto Java y la base de datos.

```
package model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;

@Entity
@Table(name = "subject")
public class Curso {

    @Id
    @Column(name = "idSubject")
    private int id;

    @Column(name = "teacher")
    private String profesor;

    @Column(name = "level")
    private String nivel;

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

    @Lob
    @Column(name = "code")
    private String codigo;
}
```

Anotación @Temporal

Sirve para especificar tipos de datos basados en el tiempo. Estos tipos de datos se pueden clasificar en dos ramas: los que vienen del paquete java.sql y los que vienen del paquete java.util.

“Los tipos temporales son el conjunto de tipos basados en el tiempo que se pueden usar en asignaciones de estado persistentes. La lista de tipos temporales admitidos incluye los tres tipos de java.sql, java.sql.Date, java.sql.Time y java.sql.Timestamp, y los dos tipos de java.util, java.util.Date y java.util . Calendario . Los tipos java.sql son completamente libres de problemas. Actúan como cualquier otro tipo de mapeo simple y no necesitan ninguna consideración especial. Sin embargo, los dos tipos java.util necesitan metadatos adicionales para indicar cuál de los tipos java.sql de JDBC se debe utilizar al comunicarse con el controlador JDBC. Esto se hace anotándolos con la anotación @Temporal y especificando el tipo JDBC como un valor del tipo enumerado TemporalType. Hay tres valores enumerados de DATE, TIME y TIMESTAMP para representar cada uno de los tipos de java.sql. “(Mike, 2013)

En el paquete **java.sql**, los tipos se trabajan directamente:

- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

En cambio, en el paquete **java.util**:

- java.util.Date
- java.util.Calendar

Se debe especificar la anotación @Temporal y, además, especificar el atributo “TemporalType” con uno de los tres valores que representan a cada uno de los tipos java.sql (DATE, TIME o TIMESTAMP).

```

package model;

import java.sql.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Transient;

@Entity
@Table(name = "people")
public class Persona {

    @Id
    @Column(name = "idpeople")
    private int id;

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

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

    @Transient
    @Temporal(TemporalType.DATE)
    private Date lastLogin;

}

```

Manejo de la llave primaria

Cada entidad debe tener una llave primaria. La anotación empleada es `@Id` sobre el atributo que contiene la llave. Adicionalmente, se puede usar `@Column` para asociar al atributo con la columna en la tabla.

Una llave primaria se asume que es “insertable”, pero no puede ser “nullable” o “updatable” por lo que se debe tener cuidado de no sobrescribir esos atributos salvo excepciones muy específicas (cuando se manejan relaciones).

Los tipos de datos soportados para una llave primaria son:

Tipos primitivos	byte, int, short, long, char
Clases de tipos primitivos	Byte, Integer, Short, Long, Character
Cadenas de caracteres	java.lang.String

Números grandes	java.math.BigInteger
Tipos basados en tiempo	java.util.Date, java.sql.Date

También, se conoce como “Generación del ID” y se realiza mediante la anotación @GeneratedValue. En base a dicha anotación, el “Persistence Provider” genera el ID para cada entidad y lo inserta en la columna respectiva.

Se debe tener en cuenta que, dependiendo de la estrategia de generación del ID, el valor obtenido puede que no esté disponible hasta que se ejecute un “flush” o un “commit” a la transacción.

“También se permiten los tipos de punto flotante como float y double, así como las clases contenedoras Float y Double y java.math.BigDecimal, pero no se recomiendan debido a la naturaleza del error de redondeo y la falta de confianza del operador equals() cuando se aplica. a ellos El uso de tipos flotantes para claves primarias es un esfuerzo arriesgado y definitivamente no se recomienda.”(Mike, 2013)

Existen cuatro estrategias posibles (que son un tipo enumerado de “.GenerationType”):

- AUTO
- TABLE
- SEQUENCE
- IDENTITY

“A veces, las aplicaciones no quieren molestarse en tratar de definir y garantizar la unicidad en algún aspecto de su modelo de dominio y se contentan con permitir que los valores del identificador se generen automáticamente para ellas. Esto se denomina generación de ID y se especifica mediante la anotación @GeneratedValue. ”(Mike, 2013)

ESTRATEGIA “.GenerationType.AUTO”

Este tipo de estrategia delega en el “Persistence Provider” la selección de la mejor forma de generación de los “ID”. Cualquiera sea la forma elegida por el provider, se confiará en los recursos de la base de datos para la obtención de los ID's.

En el caso particular de EclipseLink con MySQL, la estrategia AUTO emplea una tabla denominada “sequence”.

“Si a una aplicación no le importa qué tipo de generación utiliza el proveedor, pero quiere que se produzca, puede especificar una estrategia de AUTO. Esto significa que el proveedor utilizará cualquier estrategia que desee para generar identificadores. ”(Mike, 2013)

Figura 47: Primer ejemplo de generación

```

package model;

import java.sql.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Transient;

@Entity
@Table(name = "people")
public class Persona {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name = "idpeople")
    private int id;

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

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

    @Transient
    @Temporal(TemporalType.DATE)
    private Date lastLogin;

}

```

Nota: Elaboración Propia

Figura 48: Segundo ejemplo de generación

```

package model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;

@Entity
@Table(name = "subject")
public class Curso {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name = "idSubject")
    private int id;

    @Column(name = "teacher")
    private String profesor;

    @Column(name = "level")
    private String nivel;

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

    @Lob
    @Column(name = "code")
    private String codigo;
}

```

Nota: Elaboración Propia

ESTRATEGIA “**GenerationType.TABLE**”

La tabla requiere de dos columnas, una conteniendo el identificador para generar la secuencia y la otra columna, contiene el último valor generado. Cada fila de la tabla es un generador diferente para los ID's.

“La forma más flexible y portátil de generar identificadores es utilizar una tabla de base de datos. No solo se trasladará a diferentes bases de datos, sino que también permite almacenar varias secuencias de identificadores diferentes para diferentes entidades dentro de la misma tabla. Una tabla de generación de ID debe tener dos columnas. La primera columna es un tipo de cadena que se utiliza para identificar la secuencia del generador en particular. Es la clave principal para todos los generadores de la tabla. La segunda columna es un tipo integral que almacena la secuencia de identificación real que se está generando. El valor almacenado en esta columna es el último identificador que se asignó en la secuencia. Cada generador definido representa una fila en la tabla. ”(Mike, 2013)

Dado que no se ha especificado el nombre de un “generador” ni el nombre de una tabla, el Persistence provider seleccionará sus propios valores. Lo más común es que busque (la tabla debe existir en la base de datos) una tabla como la indicada en la figura.

¿Qué sucede si se desea especificar una tabla en particular? Se debe emplear la anotación `@TableGenerator`.

```
package model;

import java.sql.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Transient;

@Entity
@Table(name = "people")
public class Persona {

    @Id @GeneratedValue(strategy=GenerationType.TABLE)
    @Column(name = "idpeople")
    private int id;

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

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

    @Transient
    @Temporal(TemporalType.DATE)
    private Date lastLogin;

}
```

```
package model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;

@Entity
@Table(name = "subject")
public class Curso {

    @Id @GeneratedValue(strategy=GenerationType.TABLE)
    @Column(name = "idSubject")
    private int id;

    @Column(name = "teacher")
    private String profesor;

    @Column(name = "level")
    private String nivel;

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

    @Lob
    @Column(name = "code")
    private String codigo;
}
```

El atributo **allocationSize** indica el incremento en la generación del ID (para el caso del ejemplo, va de uno en uno). Por defecto, el incremento es 50.

ESTRATEGIA “**GenerationType.SEQUENCE**”

Esta estrategia depende de las capacidades de la base de datos para manejar objetos de tipo “secuencia” (caso de Oracle).

Al igual que en la estrategia TABLE, basta con escribir la anotación para que el “Persistence Provider” seleccione la mejor secuencia dentro de la base de datos.

Ejemplo:

```

@Entity (name = "subject")
@Table(name = "tbsubject")
public class Curso {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)

    @Column(name = "idSubject", nullable = false)
    private int id;

    @Column(name = "teacher")
    private String profesor;
}

```

Si se desea especificar una secuencia en particular, debe indicarse el “generator” y la anotación **@SequenceGenerator**. Se debe considerar que la secuencia debe existir, previamente, en la base de datos (salvo que la opción de generación de DDL, esté habilitada en la aplicación).

Ejemplo (la secuencia es para Oracle):

Figura 49: Ejemplo de generación

```

@Entity (name="ORDERS")
@Table(name="tborders")
public class Order {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE ,
                   generator="ORDEN_SEQ")
    @SequenceGenerator(name = "ORDEN_SEQ", sequenceName="SEQ222" )

    @Column(name = "ORDER_ID", nullable = false)
    private long orderId;

    @Column(name = "CUST_ID")
    private long custId;
}

```

CREATE SEQUENCE SEQ222
MINVALUE 1
START WITH 1
INCREMENT BY 50

Nota: Elaboración Propia

ESTRATEGIA “**.GenerationType.IDENTITY**”

Esta estrategia aprovecha las facilidades de las bases de datos para utilizar columnas de tipo “autoincremento”. Sin embargo, es menos eficiente porque el identificador generado no está disponible hasta después que ocurre el INSERT.

No requiere una anotación para el “generador” dado que el campo autoincremental es parte de la definición de la tabla que corresponde a la entidad.

```
@Entity (name = "subject")
@Table(name = "tbsubject")
public class Curso {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)

    @Column(name = "idSubject", nullable = false)
    private int id;

    @Column(name = "teacher")
    private String profesor;
```

Llave primaria compuesta

En algunas situaciones, en donde se requiere que la llave primaria de una entidad esté compuesta de múltiples atributos, JPA proporciona dos formas de soportar esta necesidad mediante las anotaciones:

- `@IdClass`
- `@EmbeddedId`

Además, se debe tener en cuenta que:

- En ambos casos, se requiere de una clase Java externa que sea la que maneje los atributos de la llave primaria.
- La clase Java que maneja los atributos de la llave primaria, debe implementar los métodos `equals()` y `hashCode()`, con el fin que el Persistence Manager pueda almacenar e identificar las entidades.
- La clase Java que representa a la llave primaria debe ser pública, implementar a la interface serializable y tener un constructor sin argumento.

Ejemplo:

Dada la siguiente tabla “**tbventa**” con una llave primaria compuesta, la entidad y la clase Java que maneja la llave primaria pueden representarse así (no olvidar que se debe generar los métodos getter/setter en ambas clases Java).

Figura 50: Ejemplo de llave primaria

```

@Entity
@Table (name = "tbventa")
@IdClass (VentaPK.class)
public class Venta implements Serializable {

    @Id
    private String codVenta;
    @Id
    private String fecha;
    @Id
    private String codProducto;
}

public class VentaPK implements Serializable {
    private String codVenta;
    private String fecha;
    private String codProducto;

    public VentaPK() {
    }

    public VentaPK(String codVenta, String fecha, String codProducto) {
        this.codVenta = codProducto;
        this.fecha = fecha;
        this.codProducto = codProducto;
    }
}

```

Nota: Elaboración Propia

Observe que la anotación **@IdClass** especifica el nombre de la clase Java que maneja la llave primaria.

Observe, también, que la clase Java que maneja la llave primaria no posee anotaciones; sin embargo, debe implementar los métodos nombrados líneas arriba.

```

public boolean equals(Object obj) {
    if (obj instanceof VentaPK) {
        VentaPK vpk = (VentaPK) obj;

        return codVenta == vpk.codVenta &&
               fecha == vpk.fecha &&
               codProducto == vpk.codProducto;
    } else {
        return false;
    }
}

@Override
public int hashCode() {
    return new String(codVenta + fecha + codProducto).hashCode();
}

```

El método `equals()` compara uno a uno los atributos de la llave primaria contra los atributos de otra entidad para verificar que no se trate de la misma entidad.

El método hashCode () devuelve un código “hash” de los valores de la llave primaria.

Para consultar una entidad con una llave primaria compuesta, se requiere generar una instancia de la clase que maneja la llave primaria, cargarle los valores necesarios y pasar dicha variable al EntityManager.

Figura 51: Ejemplo de llave primaria

```
Cargar los valores de llave primaria
    ↓
VentaPK pk = new VentaPK("4585236985", "2023-04-10", "0025");
ProcesoVenta pVenta = em.find(ProcesoVenta.class, pk);
    ↓
EntityManager
```

Nota: Elaboración Propia

Objetos embebidos

Un objeto embebido es aquel que es dependiente de una entidad: no tiene identidad por sí mismo. Entender este concepto es muy útil para el manejo de relaciones entre entidades.

Si bien, a nivel de Java, los objetos embebidos se administran de forma separada, a nivel de base de datos, la entidad y la clase embebida se almacenan sobre el mismo registro de la tabla.

Por ejemplo, en el siguiente gráfico, se tiene la entidad “CUSTOMER” y la tabla “tbcustomer”. Observe que los datos de la dirección pueden constituir una clase separada.

Figura 52: Ejemplo de objetos embebidos

```
@EntityListeners ( CustomerListener.class)
@Entity (name="CUSTOMER")
@Table ( name="tbcustomer")
public class Customer {

    @Id
    @Column(name="CUST_ID", nullable=false)
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long clienteId;

    @Column(name="FIRST_NAME", nullable=false, length=50)
    private String nombre;
    @Column(name="LAST_NAME", length=50)
    private String apellido;
    @Version
    @Column(name="LAST_UPDATED_TIME")
    private long fechaUpdate;

    @Column(name="ZIP_CODE", nullable=false, length=10)
    private String codPostal;
    @Column(name="STREET", length=50)
    private String direccion;
    @Column(name="CITY", nullable=false, length=25)
    private String ciudad;
    @Column(name="APPT", nullable=false, length=20)
    private String numero;
```

Columns and Indices		Table Options	Advanced Options
Column Name	Datatype	NOT NULL	AUTO INC
CUST_ID	BIGINT(20)	✓	
CUST_TYPE	VARCHAR(10)		
FIRST_NAME	VARCHAR(50)	✓	
LAST_NAME	VARCHAR(50)		
LAST_UPDATE...	BIGINT(20)		
ZIP_CODE	VARCHAR(10)	✓	
STREET	VARCHAR(50)		
CITY	VARCHAR(25)	✓	
APPT	VARCHAR(20)	✓	

Nota: Elaboración Propia

Si se convierte la dirección en una clase “embebida”, quedaría como en la imagen.

Figura 53: Ejemplo de objetos embebidos

```

@Entity
@Table ( name="tbcustomer")
public class Customer {

    @Id
    @Column(name="CUST_ID", nullable=false)
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long clienteId;

    @Column(name="FIRST_NAME", nullable=false, length=50)
    private String nombre;
    @Column(name="LAST_NAME", length=50)
    private String apellido;

    @Embedded
    private Address direccion;           @Embeddable
                                         public class Address {

        @Column(name="STREET", length=50)
        private String direccion;
        @Column(name="APPT", nullable=false, length=20)
        private String numero;
        @Column(name="CITY", nullable=false, length=25)
        private String ciudad;
        @Column(name="ZIP_CODE", nullable=false, length=10)
        private String codPostal;
    }
}

```

Nota: Elaboración Propia

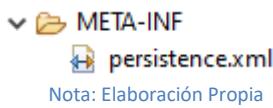
Se debe observar que:

- La entidad declara un atributo con el tipo de dato de la clase “Address” y a este atributo le coloca la anotación “@Embedded”, para indicar que esa clase es “embebida”.
- La clase “Address” NO tiene anotaciones que indiquen que es una entidad. Únicamente, tiene la anotación “@Embeddable” para indicar que hay “otra” clase que la incluye (o que la referencia).
- Ambas clases tienen sus getter/setter.
- Ambas clases deben definirse en el archivo persistence.xml.
- Finalmente, es importante saber que solo se puede ejecutar series sobre la clase marcada como “Entidad”.

Unidad de persistencia

La configuración de una unidad de persistencia se escribe en un archivo llamado **persistence.xml**, el cual debe estar ubicado dentro del folder META-INF de un proyecto Java.

Figura 54: Ubicación de la unidad de persistencia



Cada unidad de persistencia tiene un nombre, el cual es referenciado por la factoría al momento de pedirle que genere un EntityManager.

Un archivo persistence.xml puede contener una o más unidades de persistencia, siendo cada una diferente de la otra.

Debido a que es un archivo XML, debe tener un DTD, donde se incluye la definición de la unidad de persistencia, el proveedor y las clases Java definidas como entidades.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xm]
  <persistence-unit name="JPA_sesion01" transaction-type="RESOURCE_LOCAL">
    <class>model.Usuario</class>

    </persistence-unit>
</persistence>
```

El valor de **RESOURCE_LOCAL** indica que la conexión a la base de datos se realizará desde la misma aplicación (No emplea Pool de conexiones).

Después, se definen las propiedades de conexión a la base de datos.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="JPA_clase01"
    transaction-type="RESOURCE_LOCAL">
    <class>model.Usuario</class>
    <properties>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/ciberfarmadawi?serverTimezone=UTC" />
      <property name="javax.persistence.jdbc.user" value="root" />
      <property name="javax.persistence.jdbc.password" value="mysql" />
      <property name="hibernate.hbm2ddl.auto" value="create" />
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
      <property name="hibernate.show_sql" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

Entity Manager

La mayoría de las llamadas a las API's de JPA se encapsulan dentro de lo que se conoce como "Entity Manager" y mediante el cual, se puede alcanzar a la base de datos.

Esta encapsulación es implementada dentro de una interface conocida como EntityManager que es la que ejecuta todo el trabajo de persistencia. Por tanto, una entidad mientras que no se trabaje con el Entity Manager es un objeto Java simple como cualquier otro.

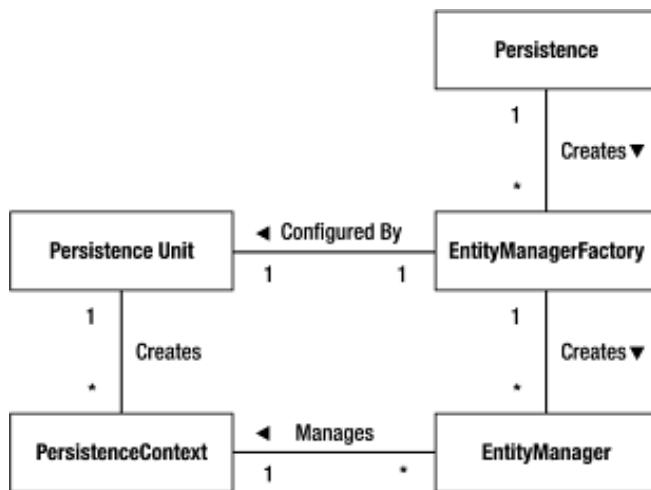
Cuando el Entity manager obtiene una referencia a una entidad, se dice que dicha entidad está en estado "managed".

El conjunto de entidades en estado "managed" dentro de un Entity Manager se conoce como "persistence context".

Los Entity Managers son configurados para trabajar con determinados tipos de objetos, bases de datos y son implementados por un proveedor (provider) conocido como “persistence provider”. En términos prácticos este provider es la implementación de la especificación JPA.

Además, los Entity Managers se generan a partir de una factoría de tipo EntityManagerFactory, que genera una especie de plantilla para la persistencia, pero toma la configuración particular desde una unidad de persistencia conocida como “persistence unit”, la cual contiene la configuración implícita o explícita (con un nombre asociado) para las entidades y para el Entity Manager.

Figura 55: Ubicación de la unidad de persistencia



Nota: Adaptado de JPA 2: Mastering the Java Persistence API

Un Entity Manager se obtiene de la siguiente forma:

```

EntityManagerFactory factory = Persistence.createEntityManagerFactory("JPA_clase01");
EntityManager manager = factory.createEntityManager();
  
```

Considerando que el nombre de la “persistence unit” debe ser el mismo que aparece en el archivo **persistence.xml**.

Operaciones básicas

Para aquellos desarrolladores acostumbrados al SQL en bases de datos relacionales, la equivalencia es sencilla en JPA.

- SQL INSERT = Método Persist
- SQL SELECT = Método Find (o también puede usarse el SELECT JPQL)
- SQL UPDATE = Método Merge
- SQL DELETE = Método Remove

El “persist” de una entidad significa crear un objeto en memoria y luego, almacenarlo en la base de datos para recuperarlo, posteriormente. Como se ha mencionado, equivale a insertar uno o más registros en la base de datos.

[OBJ]

```

Empleado emp = new Empleado();
emp.setId(8);
emp.setNombre ("Pedro");
emp.setApellido ("Picapiedra");
emp.setEdad(50);
emp.setArea("100");
em.persist(emp);

        Si ocurre un error durante la ejecución del persist,
        se lanza la excepción PersistenceException, la cual
        debe ser propagada, debiendo ser manejada por
        el programa.

        Para ubicar a una entidad empleando el método
        find, generalmente, se requiere solo una línea de
        código:

Empleado emp = em.find(Empleado.class, 8);

```

Si la entidad con la llave primaria indicada no existe, el EntityManager devolverá NULL. La aplicación debe verificar el valor antes de usar la variable **emp**, en el caso del ejemplo.

Para eliminar una entidad, se hace uso del método **remove**; sin embargo, se debe tener en consideración que para eliminar una entidad en JPA, primero debe colocarse en estado "managed", es decir, debe cargarse al contexto de persistencia.

```

Empleado emp = em.find(Empleado.class, 8);

em.remove(emp);

```

Como se mencionó anteriormente, si la entidad no existe el EntityManager devolverá NULL, por lo que se debe evaluar dicha condición antes de invocar al método remove.

Si se envía un valor de NULL al remove, JPA lanzará la excepción:
java.lang.IllegalArgumentException.

Para actualizar atributos de una entidad, se emplea el método merge. Se requiere ubicar a la entidad antes de actualizarla:

```

Empleado emp = em.find(Empleado.class, 8);

emp.setApellido ("MARMOL");

em.merge(emp);

```

En este ejemplo, se está actualizando el apellido del empleado con ID = 8.

Transacciones

El único método que puede estar fuera de una transacción es el **find** dado que no cambia atributos de las entidades.

En una aplicación Java StandAlone (Java SE), se debe invocar el contexto transaccional de forma explícita, mientras que en una aplicación Java EE, se asume que el container proporciona dicho contexto transaccional.

```
em.getTransaction().begin();

Empleado emp = em.find(Empleado.class, 8);

emp.setApellido("MARMOL");

em.merge(emp);

em.getTransaction().commit();
```

JPA proporciona unos métodos denominados callbacks (listeners) para ejecutar acciones en los diferentes estados que pueden suceder dentro del ciclo de vida de una entidad. Por ejemplo, imagine que desea actualizar una entidad, pero antes de hacerlo debe verificar que algunos datos estén presentes.

En el gráfico, se puede apreciar que una entidad no existe hasta que se distancia el objeto y se graba en la base de datos. De ahí, pasa al estado “manejado” o “administrado” por el EntityManager y luego de ello, se puede remover, actualizar, liberar (“detach”) o incluso volver a leer (refrescar).

Las anotaciones que proporciona JPA para manejar los Callbacks son:

- **@PostLoad:** se ejecuta luego de un refresh a la entidad.
- **@PrePersist:** se ejecuta antes de insertar la entidad.
- **@PostPersist:** se ejecuta después de haber insertado la entidad.
- **@PreUpdate:** se ejecuta antes de un update a la entidad.
- **@PostUpdate:** se ejecuta después de un update a la entidad.
- **@PreRemove:** se ejecuta antes de eliminar la entidad en la base de datos.
- **@PostRemove:** se ejecuta después de haber eliminado a la entidad.

Los métodos callback se pueden declarar dentro de la misma entidad o también en una clase Java separada.

Por ejemplo, si se declaran dentro de la misma entidad.

```

@Entity
@Table ( name="tbempleados")
public class Empleado {

    @Id
    private int id;
    private String apellido;
    private String nombre;
    private int edad;
    private String area;

    ... getters/setters ...

    @PostLoad
    public void despues_de_select() {
        System.out.println("EmpleadoListener: en despues_de_select");
    }
}

La Anotación va sobre la firma del método

```

No hay parámetro

1.1.3. JPA Query Language

"En muchos sentidos, JPQL es similar a SQL, una herramienta estándar para interactuar con una base de datos relacional. Ambos se utilizan para interactuar con una base de datos relacional, accediendo y manipulando datos con la ayuda de declaraciones no procesales, que son comandos reconocidos por un intérprete especial."(Vasiliev, 2008)

JPQL es un lenguaje diseñado para combinar la simplicidad de la semántica y sintaxis del lenguaje SQL con la expresividad de un lenguaje orientado a objetos.

JPQL permite abstraer de la base de datos, es decir, las consultas son escritas sobre el modelo de entidades/objetos sin necesidad de conocer como estas son mapeadas.

Tipos de Queries:

- Dynamic Queries:** Usadas para realizar consultas que reciben parámetros durante el tiempo de ejecución.
- Named Queries:** Son consultas que se crearon previamente y que se ejecutan por medio de su nombre.
- Native Queries:** Consultas de tipo nativas SQL.

SENTENCIAS JPQL

A) SELECT

Sentencia utilizada para especificar el tipo de objetos o valores que deberán ser seleccionados:

B) FROM

Sentencia para indicar el dominio de datos al que la consulta hace referencia.

C) WHERE

Sentencia utilizada como modificador opcional, utilizado para limitar los resultados que la consulta retornará.

D) GROUP BY

Sentencia opcional que permite agrupar los resultados de la consulta.

E) HAVING

Sentencia opcional que permite incluir condiciones de filtrado.

F) ORDER BY

Sentencia opcional que permite ordenar los resultados que la sentencia retorna.

Observe que la notación es muy similar al SQL normal, pero con ligeras diferencias:

- En JPQL, lo que sigue a la cláusula “FROM” es el nombre de la entidad, es decir, no se coloca el nombre de la tabla (recordar que la entidad “mapea” a una tabla).
- En JPQL es obligatorio que las entidades sean “calificadas” con un “alias”: en el caso del ejemplo, el alias es “e”. Este “alias” se conoce como “variable de identificación”.
- El alias indicará que el resultado será uno o más entidades del tipo correspondiente a la entidad.
- El tipo de resultado de un Query no puede ser una Colección. Debe ser un tipo simple o una Entidad.

EJEMPLOS

1. Observamos que esta consulta devuelve los datos de todas las personas identificadas por la variable general n:

```
SELECT m FROM PersonaEntity m
```

2. Observamos que esta consulta utiliza el modificador Distinct usada para eliminar de la respuesta los valores duplicados:

```
SELECT DISTINCT m FROM PersonaEntity m
```

3. Observamos que esta consulta utiliza el modificador Distinct usada para eliminar de la respuesta los valores duplicados en el caso que el nombre y apellido concuerde con los parámetros indicados:

```
SELECT DISTINCT m FROM PersonaEntity m WHERE m.nombre = :nombre AND m.apellido = :apellido
```

4. Observamos que esta consulta utiliza LIKE para resultados cuyo parámetro coincide con el valor indicado.

```
SELECT m FROM PersonaEntity m WHERE m.nombre LIKE 'Di%'
```

5. En esta consulta usamos COUNT, cuyo uso es indicar el número de ocurrencias.

```
SELECT COUNT(m) FROM PersonaEntity m
```

6. En esta consulta usamos COUNT, cuyo uso es indicar el número de ocurrencias.

```
SELECT COUNT(m) FROM PersonaEntity m
```

OTRAS SENTENCIAS UTILIZADAS

- A) **NULL:** Sentencia utilizada para verificar la condición de nulidad en las consultas.
- B) **EMPTY:** Sentencia utilizada para indicar que un elemento está vacío.
- C) **BETWEEN-AND:** Sentencia que es utilizada para imponer condiciones en las consultas en un rango de valores.
- D) **Operadores de comparación (<, >, =):** Utilizadas para establecer comparaciones con el modificador WHERE.

EXPRESIONES FUNCIONALES

- A) CONCAT (String, String):** Devuelve un String que es la concatenación de los dos parámetros que se envían.
- B) LENGTH (String):** Devuelve el entero que indica la longitud del String pasado como parámetro.
- C) LOCATE (String, String [, start]):** Devuelve un entero con la posición de un determinado String dentro de otro String. Si no se localiza, se devuelve un 0.
- D) SUBSTRING (String, start, length):** Devuelve un String que es un subconjunto del pasado como parámetro comenzando en la posición marcada por start y de longitud length.
- E) TRIM ([[LEADING|TRAILING|BOTH] char) FROM] (String):** Esta función elimina un determinado carácter desde el comienzo o final de un String. Si no ese especifica ningún carácter, se eliminan espacios en blanco.
- F) LOWER(String):** Convierte un String al equivalente en minúsculas.
- G) UPPER(String):** Convierte un String al equivalente en mayúsculas.

SENTENCIA UPDATE

“Las consultas JPQL UPDATE proporcionan una forma alternativa de actualizar objetos de entidad. A diferencia de las consultas SELECT, que se utilizan para recuperar datos de la base de datos, las consultas UPDATE no recuperan datos de la base de datos, pero cuando se ejecutan, actualizan el contenido de los objetos de entidad especificados en la base de datos.” (ObjectDB)

```
UPDATE Pais SET poblacion = poblacion * 11 / 10
```

```
UPDATE Pais c SET c.poblacion = c.poblacion * 11 / 10
```

Esta sentencia nos permite actualizar valores almacenados por la entidad. Puede incluir de forma opcional el modificador WHERE.

```
UPDATE PersonaEntity m SET m.edad = 25 WHERE m.nombre = 'Diana' OR m.nombre = 'Nycole'
```

SENTENCIA DELETE

“Las consultas JPQL DELETE proporcionan una forma alternativa de eliminar objetos de entidad. A diferencia de las consultas SELECT, que se utilizan para recuperar datos de la base de datos, las consultas DELETE no recuperan datos de la base de datos, pero cuando se ejecutan, eliminan objetos de entidad específicos de la base de datos.” (ObjectDB)

Sentencia que nos permite borrar un registro completo correspondiente a una entidad. Al igual que UPDATE, puede incluir de forma opcional el modificador WHERE.

```
DELETE FROM PersonaEntity m WHERE m.nombre = 'Diana'
```

```
DELETE FROM PersonaEntity m WHERE m.edad = 25
```

```
DELETE FROM PersonaEntity m WHERE m.nombre IS EMPTY
```

USO DEL JPQL

Las consultas expresadas en JPQL serán ejecutadas a través del interfaz EntityManager y los métodos createX como los siguientes:

- **Query createQuery(String qlString):** este método se utiliza para crear consultas dinámicas que toman valor en ejecución.
- **Query createNamedQuery(String name):** este método se utiliza para crear consultas estáticas que están codificadas previamente o predefinidas como metadatos a través del objeto javax.persistence.NamedQuery que las marca con la anotación @NamedQuery.

El interfaz proporciona además otro método: Query createNativeQuery(String sqlString) que permite ejecutar directamente sentencias SQL nativas.

JOIN ENTRE ENTIDADES

Al igual que en SQL, si se desea navegar entre las relaciones de las entidades y retornar elementos de la colección, se debe ejecutar un JOIN entre entidades.

Se puede ejecutar el JOIN al más puro estilo del tradicional SQL indicando los criterios de JOIN en la cláusula WHERE; sin embargo, JPQL proporciona la facilidad de especificar el JOIN dentro de la cláusula FROM con la finalidad de expresar el JOIN en términos de la relación existente entre las entidades: JPA se encargará de armar la sentencia SQL equivalente.

Un JOIN ocurre si se cumple cualquiera de las siguientes condiciones en el SELECT:

- Dos o más declaraciones de variables son listadas en la cláusula FROM y aparecen en la cláusula SELECT.

- El operador JOIN es empleado para extender a una variable de identificación usando “expression path”.
- Un “path expression” en cualquier parte del query navega, a través de un campo de asociación en la misma o en otra entidad.
- Una o más condiciones WHERE comparan atributos de variables de identificación diferentes.
- Se debe tener en cuenta que ante la ausencia de condiciones de JOIN entre entidades, se generará un producto cartesiano entre la primera entidad y cada ocurrencia de la segunda entidad.

INNER JOIN

JPQL proporciona un tipo adicional de variable de identificación, una variable de combinación, que representa una iteración más limitada sobre colecciones específicas de objetos. La sintaxis básica es:

```
SELECT p, u FROM Persona p INNER JOIN p.usuarios u
```

OUTER JOIN

La palabra clave OUTER es opcional (LEFT OUTER JOIN es equivalente a LEFT JOIN). Al usar OUTER JOIN, si una variable externa específica no tiene ningún valor interno coincidente, obtiene al menos un valor NULL como valor coincidente en la iteración FROM. Por lo tanto, un país c sin ciudad capital tiene una representación mínima de (c, NULL) en la iteración FROM.

Su sintaxis es la siguiente:

```
SELECT e, d FROM Employee e LEFT JOIN e.department d
```

QUERIES AGREGADOS

La sintaxis es muy similar a SQL: se requiere el uso del agrupamiento con GROUP BY.

Es opcional el uso del filtro mediante la cláusula HAVING. JPA incluye cinco funciones agregadas:

- AVG: promedio aritmético.
- COUNT: cantidad de repeticiones.
- MIN: menor valor.
- MAX: mayor valor.
- SUM: suma de valores.

Ejemplo: Se obtienen todos los departamentos, la cantidad de empleados de cada departamento, el sueldo máximo y el sueldo promedio, teniendo en consideración solo aquellos departamentos que tengan más de 5 empleados.

```
SELECT d, COUNT(e), MAX(e.sueldo), AVG(e.sueldo) FROM Department d JOIN d.empleados e  
GROUP BY d HAVING COUNT(e) >= 5
```

QUERIES

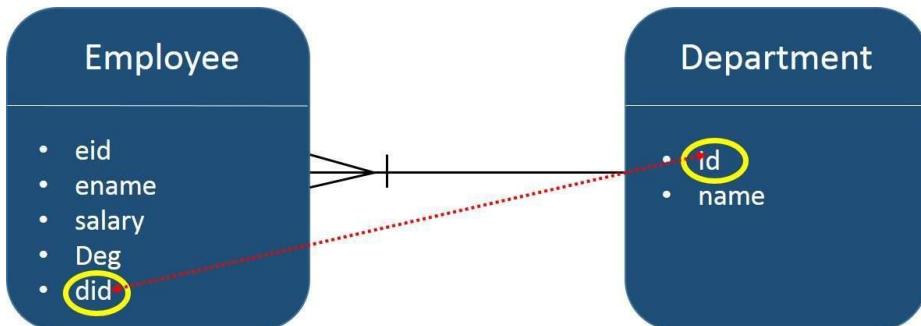
Existen dos formas para definir “queries” en JP-QL:

- La primera forma es definirlo, dinámicamente, en tiempo de ejecución como una cadena de caracteres que se construye de acuerdo con el flujo de la aplicación. Esto implica compilar el “query” cada vez.
- La segunda forma es definir el “query” vía anotación o XML, y referenciarlo por el nombre cada vez que se requiera (algo similar a IBATIS). A diferencia de la forma anterior, los “queries nombrados” son estáticos, pero son mucho más eficientes para ser ejecutados.

1.1.4 JPA y relaciones

“En JPA, puede definir una relación de propiedad en el modelo de datos que imponga restricciones mediante la gestión de cambios. Con las relaciones propias, puede decir que un libro tiene cero o más reseñas de libros, y JPA se asegura de que no pueda tener una reseña de libro sin un libro. Si elimina un objeto Libro, JPA sabe que también debe eliminar todos sus objetos LibroRevisiones. En el código Java, la relación se representa mediante un campo cuyo tipo es de la clase de datos relacionada, lo que garantiza que solo se utilicen las clases apropiadas en cualquier lado de la relación”. (Sanderson, 2010)

Figura 56: JPA y Relaciones



Nota: Adaptado de [Tutorialspoint](#)

Mapeo de relaciones

“Puede usar las relaciones propias de JPA para realizar transacciones en entidades relacionadas, y la implementación de JPA administrará los grupos de entidades automáticamente.”. (Sanderson, 2010)

La implementación del motor de aplicaciones de JPA admite relaciones de uno a uno y de uno a muchos.

Relación One to One

Para especificar una relación de propiedad uno a uno, crea un campo cuyo tipo es de la clase relacionada y asigna al campo anotaciones @OneToOne.

Figura 57: Ejemplo de relación uno a uno

```
import javax.persistence.Entity;
import javax.persistence.OneToOne;
import book.store.BookCoverImage;

@Entity(name = "Book")
public class Book {

    @OneToOne(cascade=CCascadeType.ALL)
    private BookCoverImage bookCoverImage;

}
```

Nota: Adaptado de Programming Google App Engine

Relación One To Many – Many To One

El argumento mappedBy le dice a JPA que el campo Libro se refiere al objeto Libro que está relacionado con este objeto. Esto se gestiona desde el lado del "propietario" de la relación; cuando la LibroPortadalmImagen se asigna al campo Libros, JPA sabe que la referencia inversa se refiere al objeto Libro.

Para especificar una relación de uno a varios, utilice un tipo de campo que sea una Lista o un Conjunto de la clase relacionada y utilice la anotación @OneToMany en la clase "uno"

Figura 58: Ejemplo de relación muchos a uno

```
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.OneToMany;
import book.store.BookReview;

@Entity(name = "Book")
public class Book {

    @OneToMany(cascade=CCascadeType.ALL)
    private List<BookReview> bookReviews;

}
```

```
@Entity(name = "BookReview")
public class BookReview {

    @ManyToOne(mappedBy="bookReviews")
    private Book book;

}

@Entity(name = "Book")
public class Book {

    @OneToMany(cascade=CCascadeType.ALL, mappedBy="book")
    private List<BookReview> bookReviews;

}
```

Nota: Adaptado de Programming Google App Engine

Resumen

1. Recordemos que ORM ayuda a administrar el desajuste de impedancia entre una aplicación orientada a objetos y una base de datos relacional. ORM ayuda a escribir aplicaciones menos complejas.
2. La API de persistencia de Java es un marco ligero basado en POJO para la persistencia de Java.
3. Las entidades son clases que permiten construir los objetos de objetos de transferencia de datos, es decir, de la base de datos al proyecto y viceversa.
4. Los metadatos de anotación son una característica del lenguaje introducida en Java SE 5 que permite adjuntar metadatos estructurados y escritos al código fuente.
5. El único método que puede estar fuera de una transacción es el find dado que no cambia atributos de las entidades.
6. JPQL es similar a SQL, una herramienta estándar para interactuar con una base de datos relacional.
7. Las consultas expresadas en JPQL serán ejecutadas a través del interfaz EntityManager.
8. En JPA, puede definir una relación de propiedad en el modelo de datos que imponga restricciones mediante la gestión de cambios.

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://somospnt.com/blog/235-relaciones-con-jpa>
- <https://www.baeldung.com/jpa-join-types>
- https://www.tutorialspoint.com/es/jpa/jpa_entity_relationships.htm
- <https://www.objectdb.com/java/jpa/query/jpql/delete>

1.2. MAVEN

1.2.1. Introducción y configuración

"Apache Maven es popular como herramienta de compilación. Sin embargo, en realidad, va más allá de ser solo una herramienta de construcción. Proporciona una plataforma integral de gestión de compilación. Antes de Maven, los desarrolladores tenían que dedicar mucho tiempo a crear un sistema de compilación. No había una interfaz común. Difería de un proyecto a otro, y cada vez que un desarrollador pasaba de un proyecto a otro, había una curva de aprendizaje". (Siriwardena, 2015)

Maven es una herramienta de software para la gestión y construcción de proyectos Java, tiene un modelo de configuración de construcción basado en un formato XML, utilizando un archivo POM (Project Object Model) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado (Auribox Training, 2018).

Instalación

Se puede utilizar Maven desde la consola del sistema o mediante aplicaciones.

Paso 1. Ir a la página oficial de Apache Maven, descargar el archivo **Binary zip archive** y descomprimir.

Figura 59: Página oficial de Maven

The screenshot shows the official Apache Maven website. At the top, there's a navigation bar with links for 'Apache' and 'Maven' under 'Welcome'. The main title 'Apache Maven Project' is displayed with a feather icon, and the URL 'http://maven.apache.org/'. To the right, there's a large 'Maven™' logo. Below the title, the page is titled 'Welcome to Apache Maven'. It features a brief introduction about Maven's purpose and how it can help manage projects. On the left, a sidebar menu includes 'Welcome', 'License', 'ABOUT MAVEN', 'What is Maven?', 'Features', 'Download', 'Use', 'Release Notes', 'DOCUMENTATION', 'Maven Plugins', 'Maven Extensions', 'Index (Category)', 'User Centre', 'Plugin Developer Centre', 'Maven Repository Centre', 'Maven Developer Centre', 'Books and Resources', and 'Security'. The main content area has sections for 'Use' (with 'Download, Install, Configure, Run Maven' and 'Maven Plugins and Maven Extensions'), 'Extend' (with 'Write Maven Plugins' and 'Improve the Maven Central Repository'), and 'Contribute' (with 'Help Maven' and 'Develop Maven'). A note at the bottom states: 'Each guide is divided into a number of trails to get you started on a particular topic, and includes a reference area and a "cookbook" of common examples.'

Nota. Adaptado de Welcome to Apache Maven, The Apache Software Foundation, 2023, <https://maven.apache.org/>

Figura 60: Página oficial de Maven

The screenshot shows the Apache Maven 3.9.1 download page. On the left is a sidebar with links like Welcome, License, About Maven, What is Maven?, Features, Download (which is selected), Use, Release Notes, Documentation, Maven Plugins, Maven Extensions, Index (category), User Centre, Plugin Developer Centre, Maven Repository Centre, Maven Developer Centre, Books and Resources, Security, Community, Overview, and Project Roles. The main content area has a title 'Downloading Apache Maven 3.9.1' and a sub-section 'System Requirements'. It lists requirements for Java Development Kit (JDK), Memory, Disk, and Operating System. Below that is a section 'Files' with a table of available archive types, their links, checksums, and signatures.

Link	Checksums	Signature
Binary tar.gz archive apache-maven-3.9.1-bin.tar.gz	apache-maven-3.9.1-bin.tar.gz.sha512	apache-maven-3.9.1-bin.tar.gz.asc
Binary zip archive apache-maven-3.9.1-bin.zip	apache-maven-3.9.1-bin.zip.sha512	apache-maven-3.9.1-bin.zip.asc
Source tar.gz archive apache-maven-3.9.1-src.tar.gz	apache-maven-3.9.1-src.tar.gz.sha512	apache-maven-3.9.1-src.tar.gz.asc
Source zip archive apache-maven-3.9.1-src.zip	apache-maven-3.9.1-src.zip.sha512	apache-maven-3.9.1-src.zip.asc

Nota. Adaptado de Welcome to Apache Maven, The Apache Software Foundation, 2023, <https://maven.apache.org/>

Paso 2. Revisar las variables del sistema y agregar las opciones M2_HOME y MAVEN_HOME a la ubicación del archivo descomprimido.

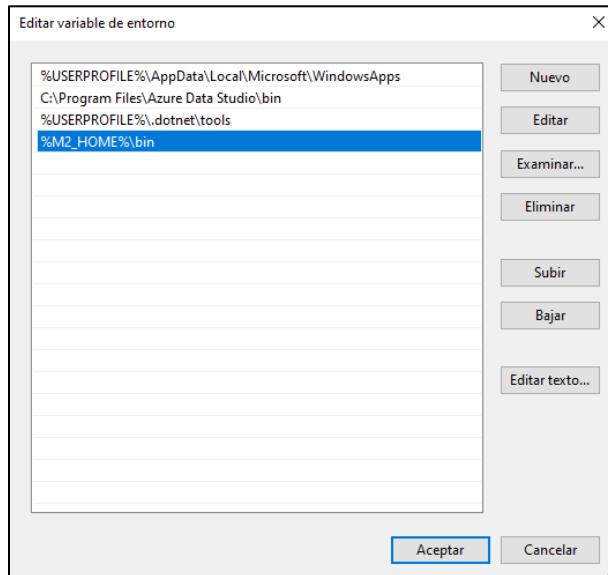
Figura 61: Opciones Variables del Sistemas del diálogo Variables de entorno

The screenshot shows the Windows Control Panel under 'Sistemas y Seguridad' (Systems and Security) with 'Variables del sistema' (System variables) selected. A table lists environment variables: JAVA_HOME (C:\Program Files\Java\jdk1.8.0_202), M2_HOME (C:\Recursos-LP-II\apache-maven-3.9.1), and MAVEN_HOME (C:\Recursos-LP-II\apache-maven-3.9.1). A note at the bottom says 'Nota: Elaboración Propia'.

Variable	Valor
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_202
M2_HOME	C:\Recursos-LP-II\apache-maven-3.9.1
MAVEN_HOME	C:\Recursos-LP-II\apache-maven-3.9.1

En la variable PATH, agregar M2_HOME editando:

Figura 62: Agregando la ruta a la variable del sistema PATH



Nota: Elaboración Propia

Paso 3. Comprobar, abriendo un cuadro de consola y escribiendo el código:

Figura 63: Comprobando la conexión con Maven

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.19044.2846]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pc>mvn -v
Apache Maven 3.9.1 (2e178502fcdbffcc201671fb2537d0cb4b4cc58f8)
Maven home: C:\Recursos-LP-II\apache-maven-3.9.1
Java version: 1.8.0_202, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_202\jre
Default locale: es_ES, platform encoding: Cp1252
OS name: "Windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\pc>
```

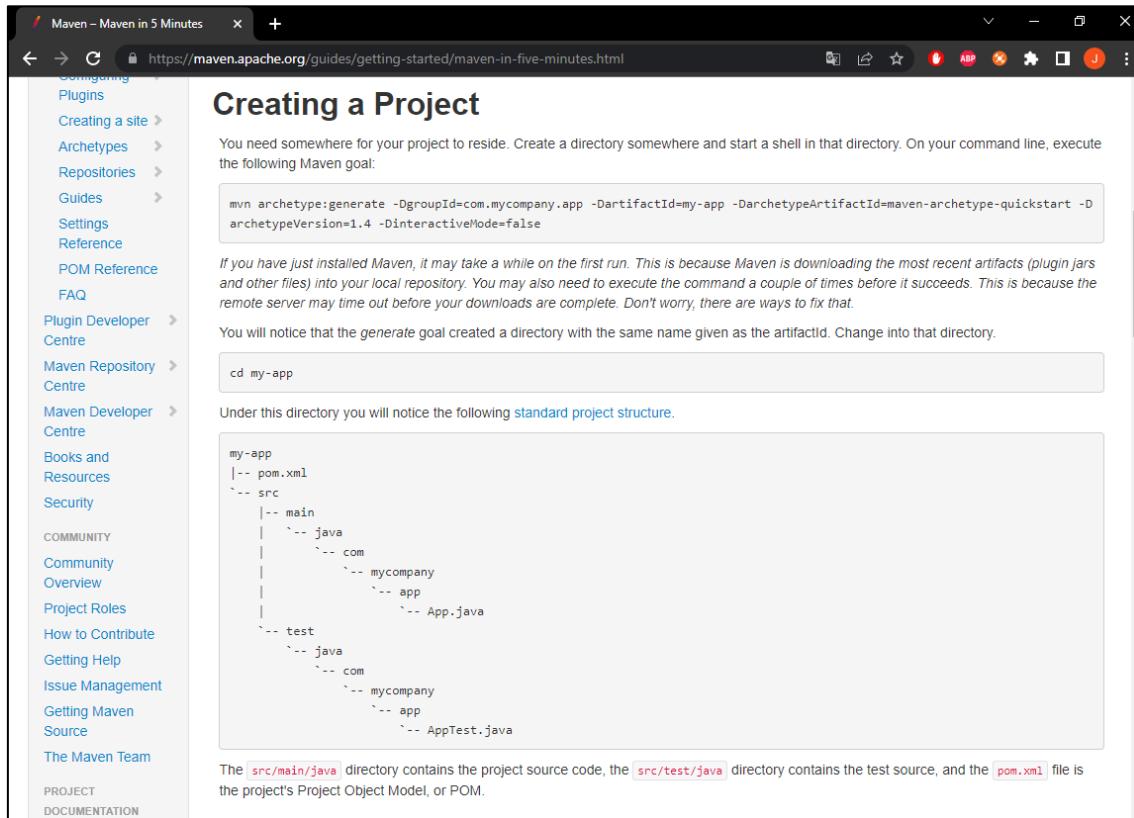
Nota: Elaboración Propia

1.2.2. Creación de un proyecto

Creando un proyecto desde consola

Existe una estructura y comando que podemos ejecutar para la construcción de los proyectos, sean de escritorio o web.

Figura 64: Comprobando la conexión con Maven



Nota: Adaptado de Maven in 5 Minutes, The Apache Software Foundation, 2023,
<https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

Ejemplo:

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false
```

Figura 65: Comprobando la conexión con Maven

```
C:\Users\pc>mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false
```

Nota: Elaboración Propia

Donde:

- **groupId.** El identificador único para crear un artefacto. Se suele poner el mismo que en un paquete java.

- **artifactId**. Es el nombre del proyecto que lo gestiona Maven y que incluye un archivo llamado pom.xml.
- **pom.xml**. Son las siglas de **Project Object Model**. Es un archivo XML que contiene la configuración del artefacto.
- **archetypeArtifactId=maven-archetype-quickstart**. Permite especificar el tipo de plantilla o modelo:
 - maven-archetype-quickstart (proyecto Java simple),
 - maven-archetype-webapp (java web)

Figura 66: Comprobando la conexión con Maven

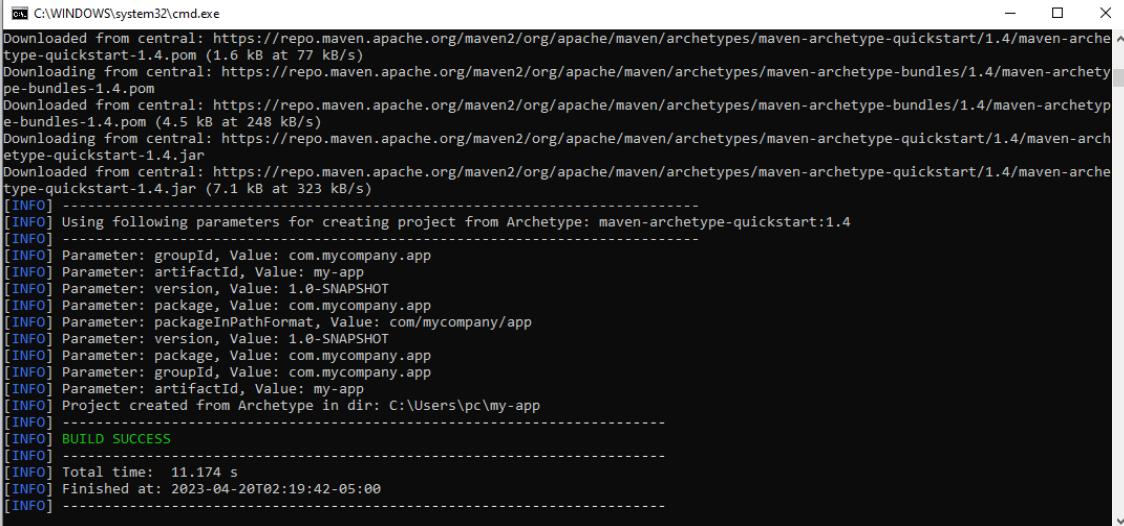
The screenshot shows a web browser window with the URL maven.apache.org/guides/introduction/introduction-to-archetypes.html. The left sidebar has a 'Archetypes' section selected. The main content area is titled 'Provided Archetypes' and contains a table listing ten archetypes:

Archetype Artifacts	Description
maven-archetype-archetype	An archetype to generate a sample archetype project.
maven-archetype-j2ee-simple	An archetype to generate a simplified sample J2EE application.
maven-archetype-mojo	An archetype to generate a sample a sample Maven plugin.
maven-archetype-plugin	An archetype to generate a sample Maven plugin.
maven-archetype-plugin-site	An archetype to generate a sample Maven plugin site.
maven-archetype-portlet	An archetype to generate a sample JSR-268 Portlet.
maven-archetype-quickstart	An archetype to generate a sample Maven project.
maven-archetype-simple	An archetype to generate a simple Maven project.
maven-archetype-site	An archetype to generate a sample Maven site which demonstrates some of the supported document types like APT, XDoc, and FML and demonstrates how to i18n your site.
maven-archetype-site-simple	An archetype to generate a sample Maven site.
maven-archetype-webapp	An archetype to generate a sample Maven Webapp project.

Nota: Adaptado de [Introduction to Archetypes](#)What is Archetype?, The Apache Software Foundation, 2023, <https://maven.apache.org/guides/introduction/introduction-to-archetypes.html>

- **interactiveMode=false**. Permite desactivar las consultas o preferencias al momento de generar el proyecto.

Figura 67: Comprobando la creación del proyecto



```

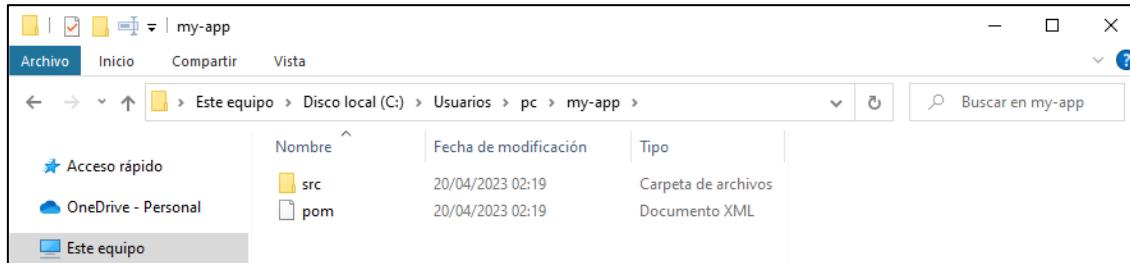
C:\WINDOWS\system32\cmd.exe
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.pom (1.6 kB at 77 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-bundles/1.4/maven-archetype-bundles-1.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-bundles/1.4/maven-archetype-bundles-1.4.pom (4.5 kB at 248 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.jar (7.1 kB at 323 kB/s)
[INFO] -----
[INFO] Using following parameters for creating project from Archetype: maven-archetype-quickstart:1.4
[INFO] -----
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: my-app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: packageInPathFormat, Value: com/mycompany/app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: my-app
[INFO] Project created from Archetype in dir: C:\Users\pc\my-app
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11.174 s
[INFO] Finished at: 2023-04-20T02:19:42-05:00
[INFO] -----

```

Nota: Elaboración Propia

En nuestra carpeta, se tendrá la siguiente estructura.

Figura 68: Comprobando la creación del proyecto en la carpeta



Nota: Elaboración Propia

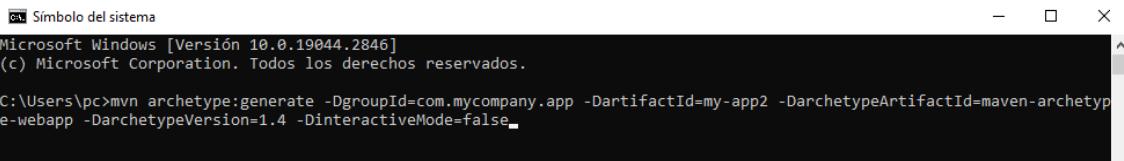
Ejemplo 2: Creando un proyecto web

Figura 69: Comando de creación del proyecto web

```

mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app2 -
-DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion=1.4 -
DinteractiveMode=false

```



```

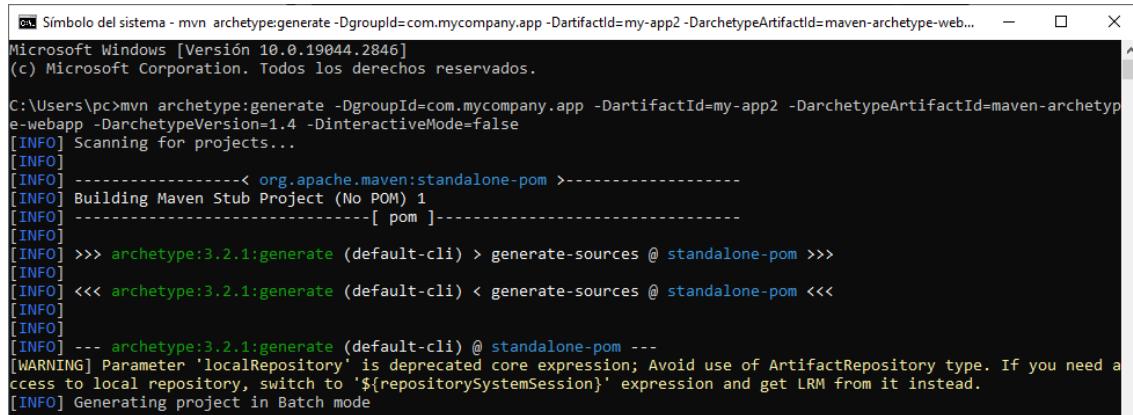
Símbolo del sistema
Microsoft Windows [Versión 10.0.19044.2846]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pc>mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app2 -DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion=1.4 -DinteractiveMode=false

```

Nota: Elaboración Propia

Figura 70: Creación del proyecto web

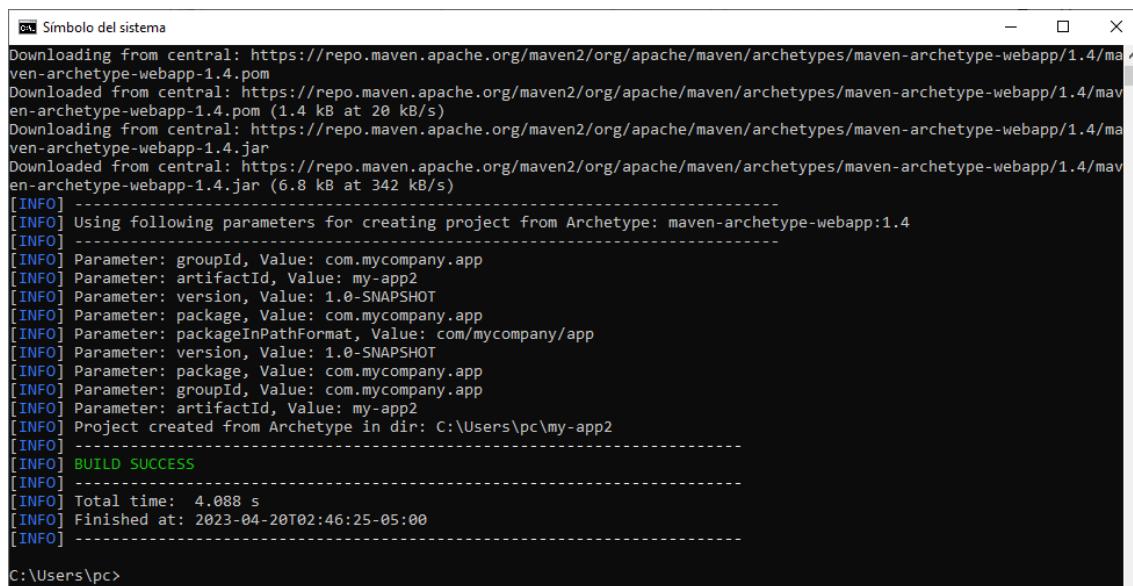


```
C:\> Símbolo del sistema - mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app2 -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
Microsoft Windows [Versión 10.0.19044.2846]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pc>mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app2 -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] -----> org.apache.maven:standalone-pom <-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
[INFO]
[INFO] >>> archetype:3.2.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< archetype:3.2.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- archetype:3.2.1:generate (default-cli) @ standalone-pom ---
[WARNING] Parameter 'localRepository' is deprecated core expression; Avoid use of ArtifactRepository type. If you need access to local repository, switch to '${repositorySystemSession}' expression and get LRM from it instead.
[INFO] Generating project in Batch mode
```

Nota: Elaboración Propia

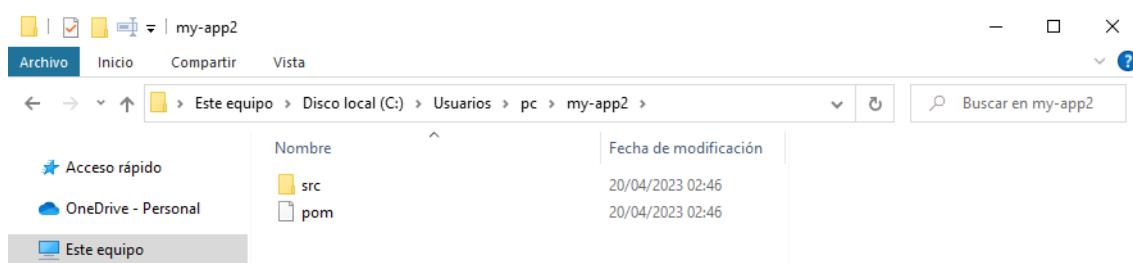
Figura 71: Creando un proyecto tipo web



```
C:\> Símbolo del sistema
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/1.4/maven-archetype-webapp-1.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/1.4/maven-archetype-webapp-1.4.pom (1.4 kB at 20 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/1.4/maven-archetype-webapp-1.4.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/1.4/maven-archetype-webapp-1.4.jar (6.8 kB at 342 kB/s)
[INFO]
[INFO] Using following parameters for creating project from Archetype: maven-archetype-webapp:1.4
[INFO]
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: my-app2
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: packageInPathFormat, Value: com/mycompany/app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: my-app2
[INFO] Project created from Archetype in dir: C:\Users\pc\my-app2
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 4.088 s
[INFO] Finished at: 2023-04-20T02:46:25-05:00
[INFO]
```

Nota: Elaboración Propia

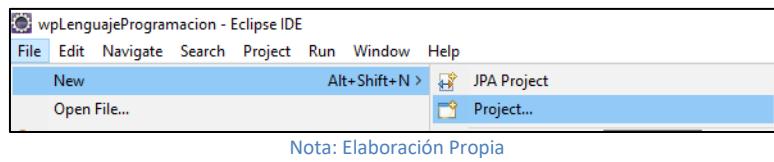
Figura 72: Comprobando la creación del proyecto web en la carpeta



Nota: Elaboración Propia

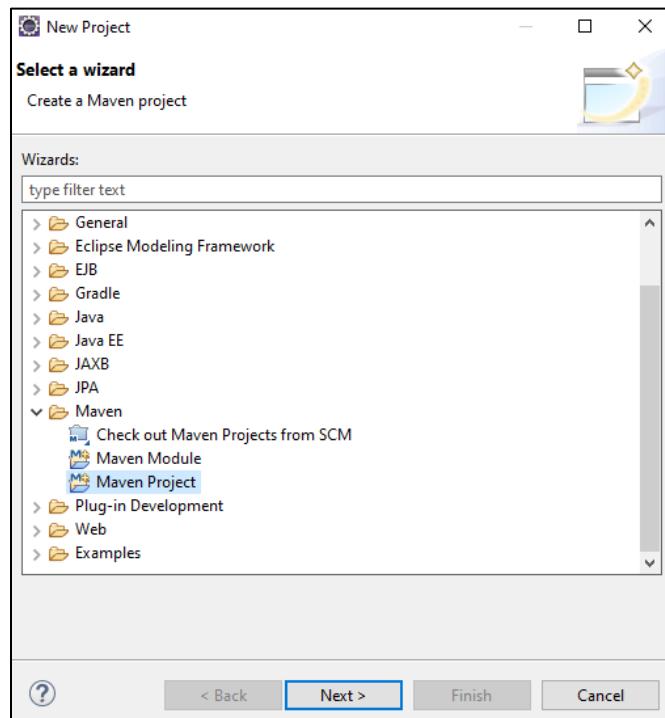
Creando un proyecto con Maven

Figura 73: Creación del proyecto con Maven



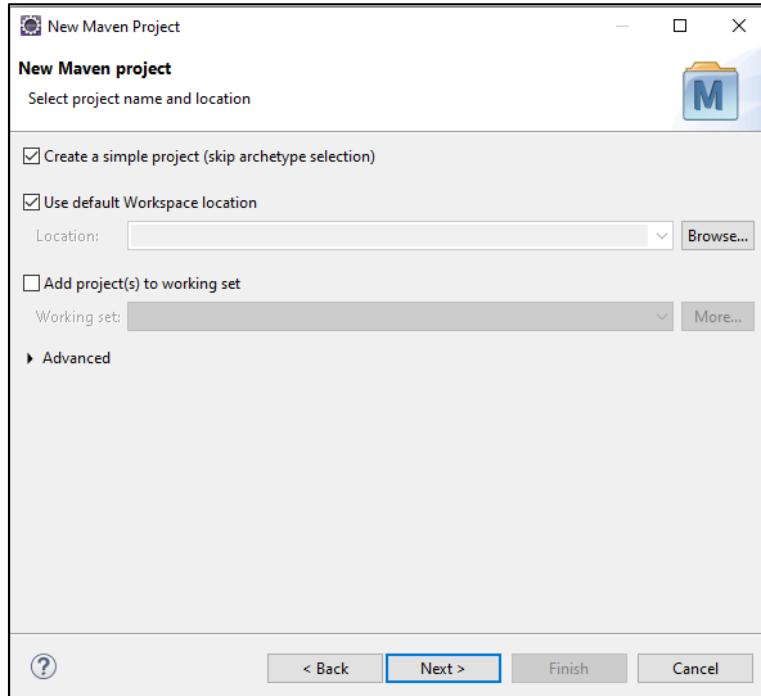
Nota: Elaboración Propia

Figura 74: Seleccionando el proyecto tipo Maven



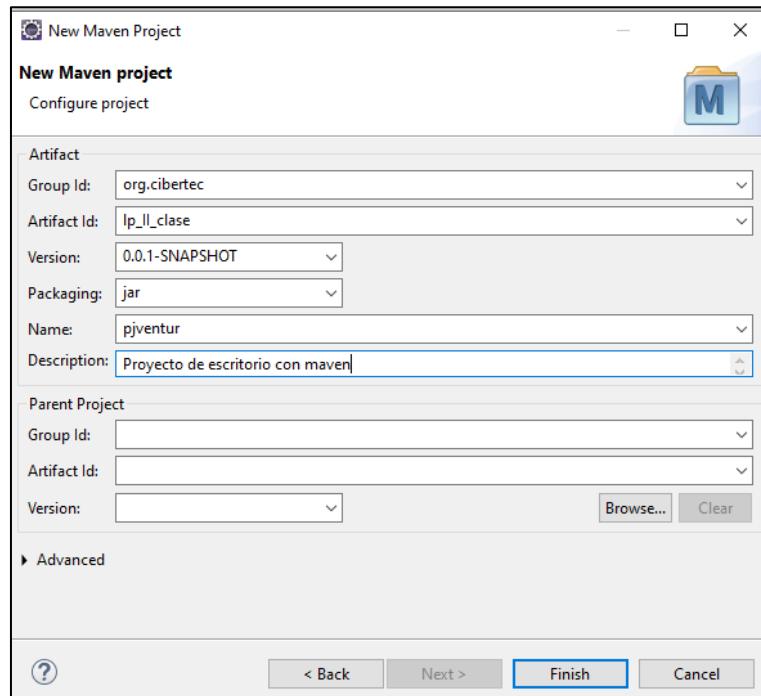
Nota: Elaboración Propia

Figura 75: Verificaciones del proyecto Maven



Nota: Elaboración Propia

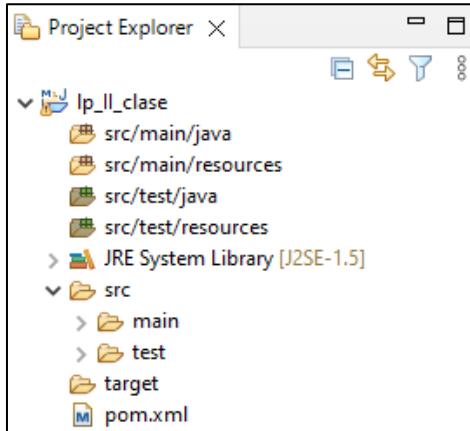
Figura 76: Configurando datos del proyecto Maven



Nota: Elaboración Propia

 Se creará la estructura del proyecto, usando el **JDK 1.5**

Figura 77: Estructura del proyecto



Nota: Elaboración Propia

 Para cambiar la versión, configurar sus propiedades en el archivo **pom.xml** y actualizar.

Figura 79: Contenido del archivo pom.xml

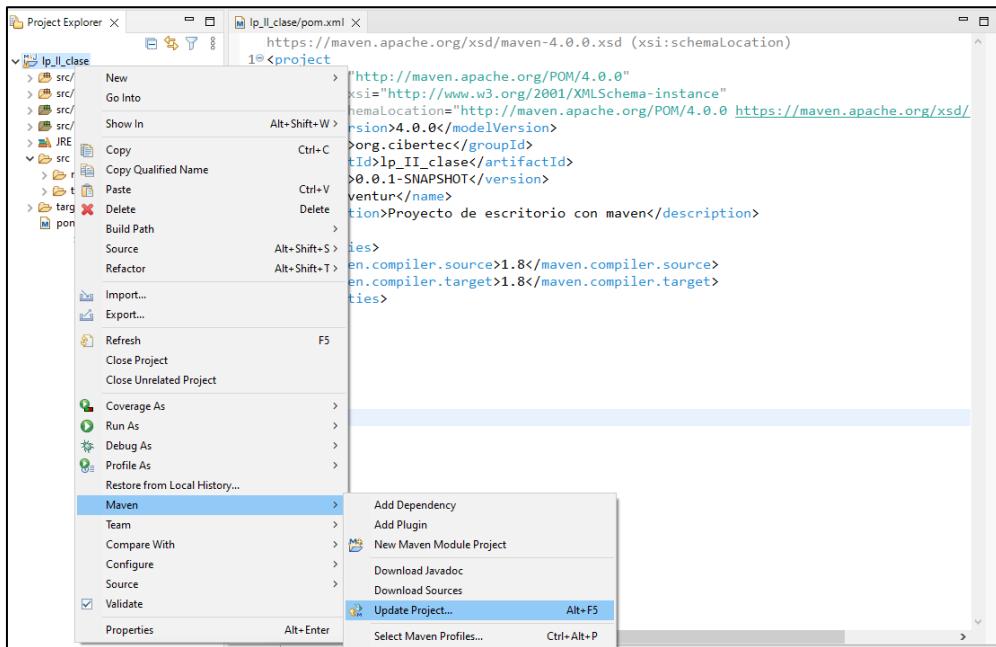
```

lp_ll_clase/pom.xml X https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
1<project
2  xmlns="http://maven.apache.org/POM/4.0.0"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemalocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
5  <modelVersion>4.0.0</modelVersion>
6  <groupId>org.cibertec</groupId>
7  <artifactId>lp_ll_clase</artifactId>
8  <version>0.0.1-SNAPSHOT</version>
9  <name>pjventur</name>
10 <description>Proyecto de escritorio con maven</description>
11
12<properties>
13   <maven.compiler.source>1.8</maven.compiler.source>
14   <maven.compiler.target>1.8</maven.compiler.target>
15 </properties>
16
17 </project>

```

Nota: Elaboración Propia

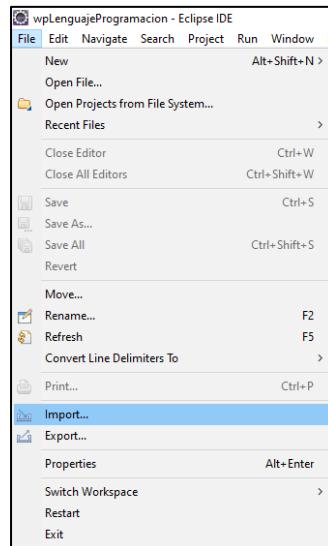
Figura 80: Actualizando el proyecto



Nota: Elaboración Propia

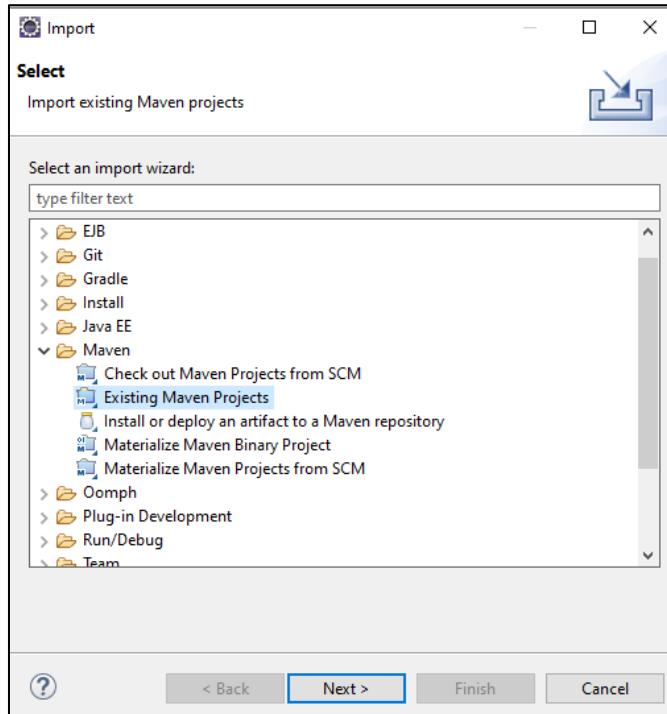
Recuperando proyectos

Figura 81: Importar un proyecto Maven



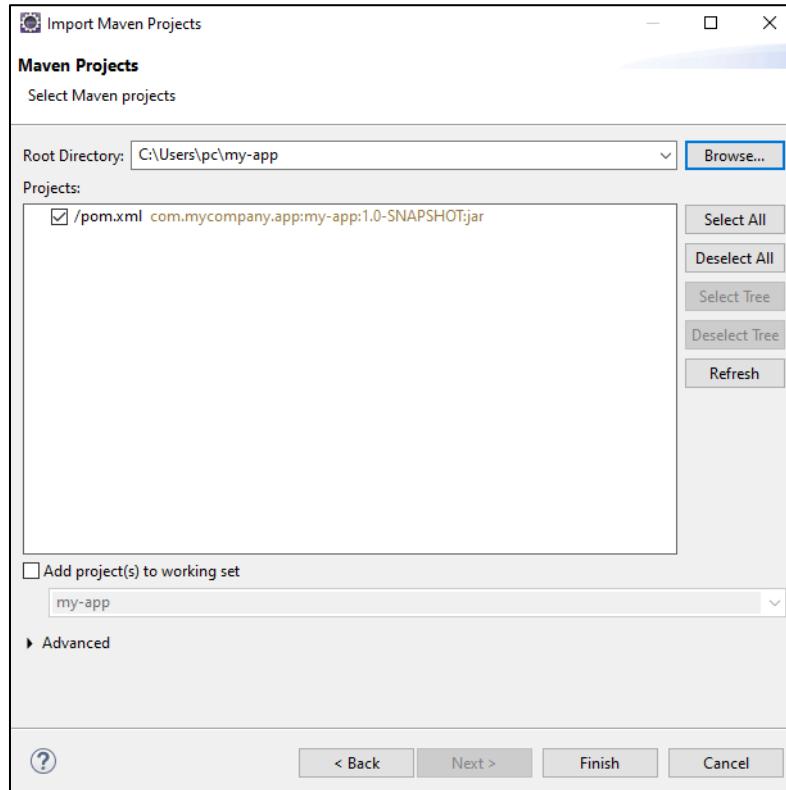
Nota: Elaboración Propia

Figura 82: Importando proyecto Maven



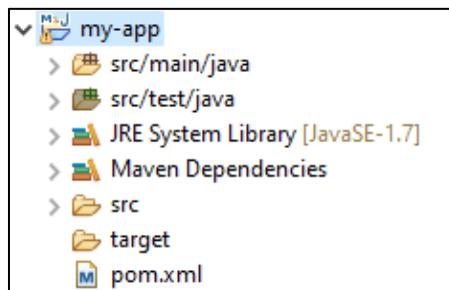
Nota: Elaboración Propia

Figura 83: Importando proyecto Maven



Nota: Elaboración Propia

Figura 84: Estructura del proyecto Maven importado



Nota: Elaboración Propia

1.2.3. Ciclo de vida

El ciclo de vida principal de un proyecto Maven es:

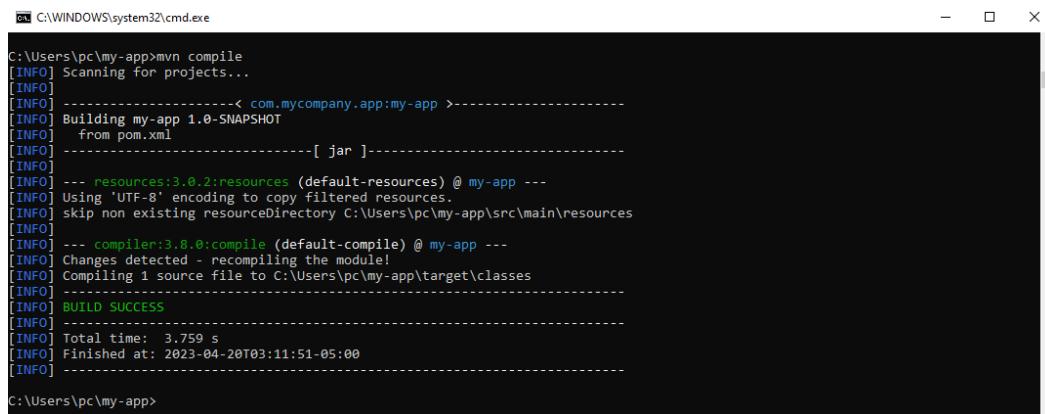
- **compile:** genera los ficheros .class compilando los archivos fuente .java

Figura 85: Comando compile

```
C:\WINDOWS\system32\cmd.exe
C:\Users\pc\my-app>mvn compile
```

Nota: Elaboración Propia

Figura 86: Ejecución del comando compile



```
C:\Users\pc\my-app>mvn compile
[INFO] Scanning for projects...
[INFO] ...
[INFO] --- < com.mycompany.app:my-app > ---
[INFO] Building my-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] ...
[INFO] --- [ jar ] ---
[INFO] ...
[INFO] --- resources:3.0.2:resources (default-resources) @ my-app ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\pc\my-app\src\main\resources
[INFO] ...
[INFO] --- compiler:3.8.0:compile (default-compile) @ my-app ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\pc\my-app\target\classes
[INFO] ...
[INFO] BUILD SUCCESS
[INFO] ...
[INFO] Total time: 3.759 s
[INFO] Finished at: 2023-04-20T03:11:51-05:00
[INFO] ...
```

Nota: Elaboración Propia

 Genera la carpeta **target** con la información de las clases generados.

Figura 87: Carpeta target



Nota: Elaboración Propia

- **test:** ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
- **package:** genera el fichero .jar (ejecutable) con los .class compilados.

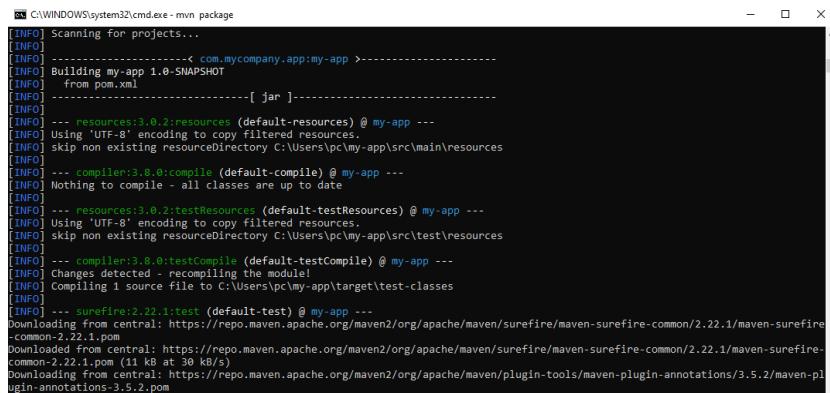
Figura 88: Comando Package



```
C:\Users\pc\my-app>mvn package
```

Nota: Elaboración Propia

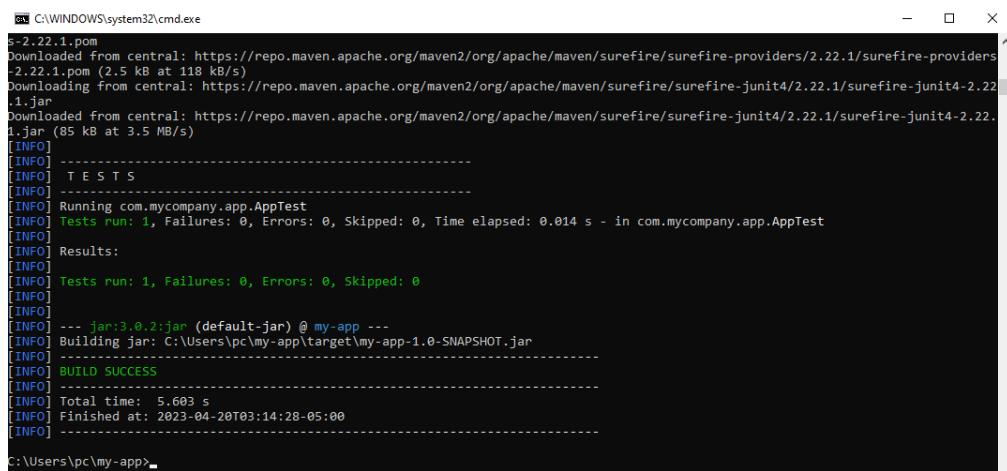
Figura 89: Ejecución del comando Package



```
C:\Users\pc\my-app>mvn package
[INFO] Scanning for projects...
[INFO] ...
[INFO] --- < com.mycompany.app:my-app > ---
[INFO] Building my-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] ...
[INFO] --- [ jar ] ---
[INFO] ...
[INFO] --- resources:3.0.2:resources (default-resources) @ my-app ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\pc\my-app\src\main\resources
[INFO] ...
[INFO] --- compiler:3.8.0:compile (default-compile) @ my-app ---
[INFO] Nothing to compile - all classes are up to date
[INFO] ...
[INFO] --- resources:3.0.2:testResources (default-testResources) @ my-app ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\pc\my-app\src\test\resources
[INFO] ...
[INFO] --- compiler:3.8.0:testCompile (default-testCompile) @ my-app ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\pc\my-app\target\test-classes
[INFO] ...
[INFO] --- surefire:3.22.1:test (default-test) @ my-app ---
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/maven-surefire-common/2.22.1/maven-surefire-common-2.22.1.pom
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/maven-surefire-common/2.22.1/maven-surefire-common-2.22.1.pom (11 kB at 30 kB/s)
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugin-tools/maven-plugin-annotations/3.5.2/maven-plugin-annotations-3.5.2.pom
```

Nota: Elaboración Propia

Figura 90: Ejecución del comando package



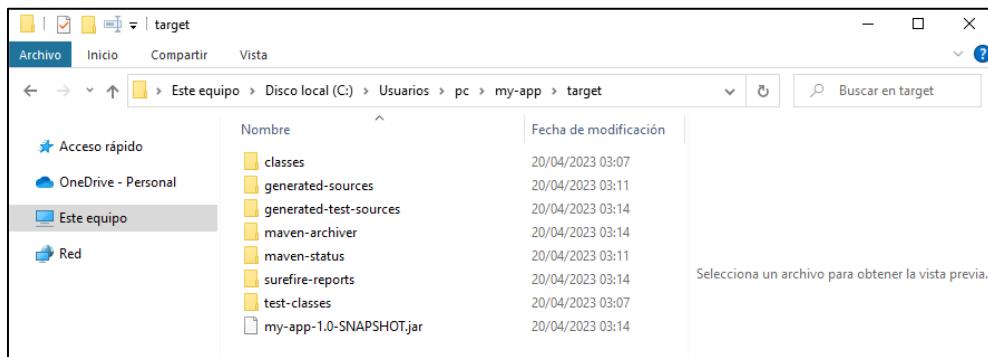
```

C:\WINDOWS\system32\cmd.exe
5-2.22.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-providers/2.22.1/surefire-providers-2.22.1.pom (2.5 kB at 118 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.22.1/surefire-junit4-2.22.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.22.1/surefire-junit4-2.22.1.jar (85 kB at 3.5 MB/s)
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.mycompany.app.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.014 s - in com.mycompany.app.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- jar:3.0.2:jar (default-jar) @ my-app ---
[INFO] Building jar: C:\Users\pc\my-app\target\my-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.603 s
[INFO] Finished at: 2023-04-20T03:14:28-05:00
[INFO] -----
C:\Users\pc\my-app>

```

Nota: Elaboración Propia

Figura 91: Carpeta target



Nota: Elaboración Propia

- **install:** copia el fichero .jar a un directorio de nuestro ordenador donde maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos maven en el mismo ordenador.
- **deploy:** copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

1.2.4. Arquetipos

“Los arquetipos no son más que tipos de proyectos, como si se trataran de plantillas. Por ejemplo, no podemos generar un proyecto Maven orientado a la web que ya dispondrá de una estructura de directorios lista para trabajar. El arquetipo más simple para proyectos Java de consola sería maven-archetype-quickstart”. (Pérez, 2016)

En resumen, un arquetipo es básicamente una plantilla, es decir, un patrón o modelo original desde el que se generarán otros elementos del mismo tipo.

Ventajas:

- Reutilización de código.
- Homogeneidad en la estructura del proyecto, lo que permite una estandarización de las tareas de desarrollar, desplegar y mantener.
- Reducción del tiempo.

Para crear la base de un arquetipo, se dispone del siguiente comando de consola:

```
mvn archetype:generate -DgroupId=[groupID del arquetipo] -  
DartifactId=[artifactId del arquetipo] -DarchetypeArtifactId=maven-archetype-  
archetype
```

Para este ejemplo ejecutar el siguiente comando:

```
mvn archetype:generate -DgroupId=org.cibertec -DartifactId=dawi_clase02 -  
DarchetypeArtifactId=maven-archetype-archetype
```

Se construye un proyecto con la misma estructura definida en el arquetipo.

1.2.5. Gestión de dependencias

“Maven se basa en un Project Object Model (POM) para describir el proyecto de software a construir, sus componentes externos y dependencias de otros módulos, y orden de construcción de los elementos que integran el proyecto.” (Pérez, 2016)

Para gestionar las dependencias, dentro del POM se deberán fijar en el tag dependencies. Por defecto, se tendrá cargada la librería de junit, que servirá para ver un ejemplo de cómo añadir más librerías.

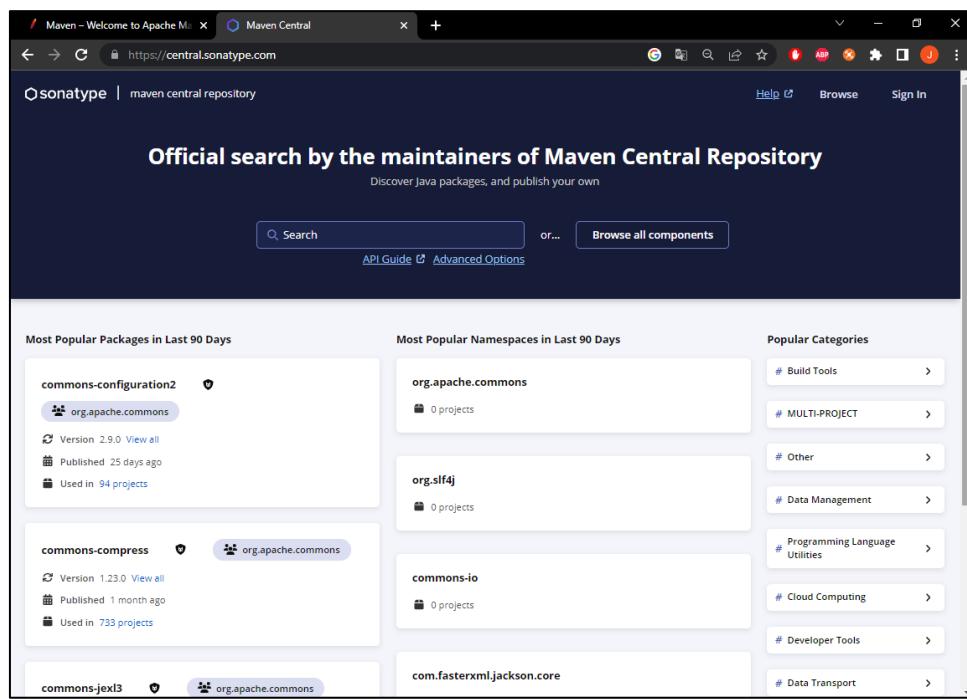
Se tienen, al igual que con los plugin, los tag de groupId y artifactId. Aparte de eso, lo que interesa también es el tercer tag, versión, que permitirá indicar la versión de la librería y en caso de tener que modificarla, bastará con cambiar ese valor y volver a hacer un install.

Existen varios sitios como MVN Repository (<https://mvnrepository.com/>) o Maven Central Repository (<https://search.maven.org/>) que cuenta con un amplio abanico de dependencias organizadas por temas.

Configurando dependencias

Para ello, ingresar a la central de repositorios Maven:

Figura 92: Repositorios Maven



Nota: Adaptado de Official search by the maintainers of Maven Central Repository, sonatype, s.f., <https://search.maven.org/>

Buscar la dependencia para Hibernate (Hibernate-core), copiar la dependencia y esperar:

Figura 93: Ejemplo de búsqueda de repositorios Maven para Hibernate

The screenshot shows the search results for 'hibernate-core' on the Sonatype Maven Central repository. The search bar at the top contains 'hibernate-core'. On the left, there are filtering options for 'Namespace', 'Dependent On', 'Contributor Email', and 'Category'. The main search results area shows two entries for 'hibernate-core': one from 'org.hibernate' and one from 'org.hibernate.orm'. Both entries show a description of 'No description provided'. Below each entry are category tags: '#Data Management', '#Database Access', '#ORM Framework' for the first entry, and '#Uncategorized' for the second. To the right of the entries, there are detailed package information cards. For the first entry, it says 'Latest version 6.2.1.Final View all', 'Published 5 days ago', 'Licenses GNU Library General Public Lic...', 'Used in 0 projects', 'Sonatype Safety Rating 7 out of 10', and 'OSS Index No vulnerabilities fo... View'. For the second entry, it says 'Latest version 6.2.1.Final View all', 'Published 5 days ago', 'Licenses GNU Library General Public Lic...', 'Used in 30 projects', 'Sonatype Safety Rating 8 out of 10', 'BOM Doctor Report View', and 'OSS Index No vulnerabilities fo... View'.

Nota: Adaptado de Official search by the maintainers of Maven Central Repository, sonatype, s.f., <https://search.maven.org/>

Figura 94: Ejemplo de dependencias en repositorios Maven para Hibernate

The screenshot shows the Sonatype Maven Central Repository interface. The search bar at the top has 'hibernate-core' entered. Below the search bar, the artifact name 'hibernate-core' is displayed along with its version '6.0.0.Alpha6'. A link to the 'Used in 1 component' section is shown. The main content area includes tabs for 'Overview', 'Versions', 'Dependents', and 'Dependencies'. The 'Overview' tab is selected, showing a brief description of the Hibernate ORM 6.0.0.Alpha6 release. The 'Snippets' tab contains a code snippet for a Maven dependency:

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.0.0.Alpha6</version>
</dependency>
```

The 'Maven POM File' tab shows the full XML content of a sample POM file. On the right side, there is a 'Sonatype Safety Rating' card with a score of 7 out of 10, and a 'Metadata' section showing the last update was 2 years ago and the license is GNU Library General Public License.

Nota: Adaptado de Official search by the maintainers of Maven Central Repository, sonatype, s.f., <https://search.maven.org/>

Copiar la dependencia en el archivo **pom.xml**.

Figura 95: Contenido archivo pom.xml

```
<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/
<modelVersion>4.0.0</modelVersion>
<groupId>org.cibertec</groupId>
<artifactId>lp_II_clase</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>pjventur</name>
<description>Proyecto de escritorio con maven</description>

<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>

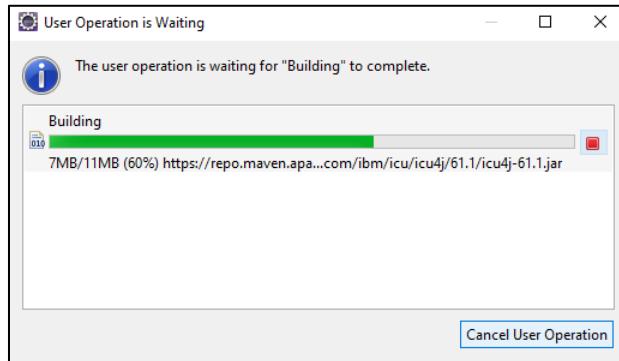
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.0.0.Alpha6</version>
    </dependency>
</dependencies>

</project>
```

Nota: Elaboración Propia

Esperar a que se instalen las librerías y luego, instalar la dependencia del conector MySQL

Figura 96: Carga de repositorios



Nota: Adaptado de Official search by the maintainers of Maven Central Repository, sonatype, s.f., <https://search.maven.org/>

Figura 97: Ejemplo de búsqueda de repositorios Maven para MySQL

Nota: Adaptado de Official search by the maintainers of Maven Central Repository, sonatype, s.f., <https://search.maven.org/>

Figura 98: Ejemplo de dependencias en repositorios Maven para MySQL

The screenshot shows the Sonatype Maven Central Repository search results for the package `mysql-connector-java` at version `8.0.23`. The page includes sections for Overview, Snippets (Apache Maven code snippet), and Metadata (last updated 2 years ago, 2 licenses). The Sonatype Safety Rating is displayed as 7 out of 10.

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.23</version>
</dependency>
```

Nota: Adaptado de Official search by the maintainers of Maven Central Repository, sonatype, s.f., <https://search.maven.org/>

Figura 99: Contenido archivo pom.xml

```
<dependencies>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>6.0.0.Alpha6</version>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.23</version>
</dependency>
</dependencies>
```

Nota: Adaptado de Official search by the maintainers of Maven Central Repository, sonatype, s.f., <https://search.maven.org/>

Resumen

1. Maven es una herramienta que permite la gestión eficiente de proyectos con java. Tiene una configuración de construcción basada en xml.
2. Para poder realizar la gestión de dependencias con Maven es necesario indicar las dependencias y versiones a utilizar en el archivo pom.xml.
3. Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo que provee acceso a muchas versiones de diferentes proyectos Open Source en Java y otras organizaciones y desarrolladores.
4. Maven provee soporte no solo para obtener archivos de su repositorio, sino también, para subir artefactos al repositorio, dejándolos al alcance de todos los usuarios.

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

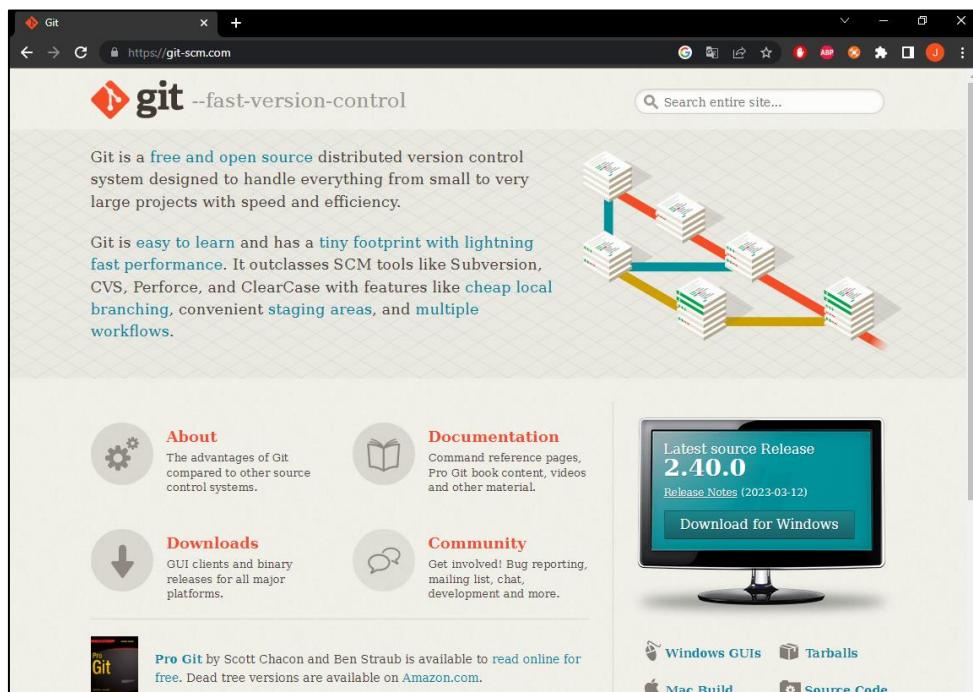
- <https://central.sonatype.com/>
- <https://maven.apache.org/install.html>
- <https://www.adictosaltrabajo.com/2008/06/09/creararquetiposmaven/>
- <https://www.youtube.com/watch?v=zIbuDA5Y9KI>

1.3. GIT

"Cuando se habla de sistema de gestión de versiones, el neófito tiene a considerarlo como un sistema de salvaguardia incremental. Pero Git no solo permite eso, abarca otras muchas posibilidades. Git permite llevar en paralelo varias versiones del mismo software. Git también sirve como documentación completa". (Dauzon, 2018)

No solo las empresas desarrollan proyectos de software de forma colaborativa: también en el sector del código abierto, miles de voluntarios y colaboradores pueden participar en la creación, el mantenimiento, la optimización o la edición de un programa, dependiendo del tamaño del proyecto. Sería prácticamente imposible llevar a cabo estas tareas sin un buen sistema para registrar y controlar los numerosos cambios realizados por todos los desarrolladores.

Figura 100: Página principal Git



Nota: Adaptado de Git, s.f., <http://git-scm.com/>

1.3.3. Introducción a los sistemas de control de versiones

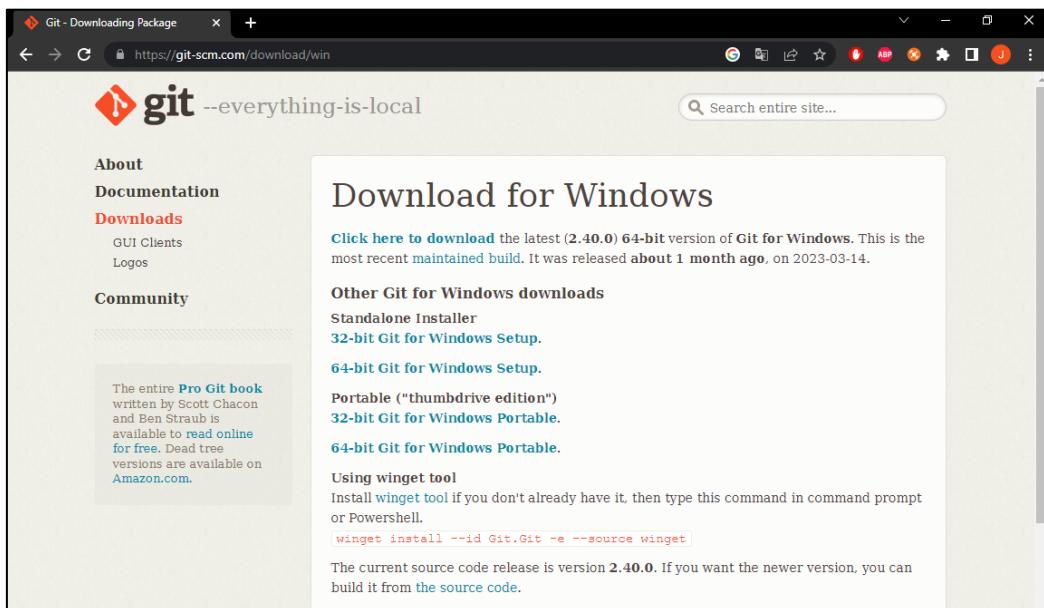
Git es un sistema de control de versiones desarrollado en 2005 por Linus Torvalds, el creador de Linux, y publicado bajo la licencia de software libre GPLv2 de GNU. La particularidad de esta herramienta es que, aunque guarda un repositorio central para cada proyecto, todos los participantes descargan una copia local del mismo en su propio dispositivo. Cada una de estas copias constituye una copia completa de todo el contenido del repositorio, por lo que no es necesario estar conectado a la red para trabajar.

Estos archivos sirven como copia de seguridad en caso de que el repositorio principal falle o resulte dañado. Los cambios en los archivos pueden intercambiarse con todos los demás participantes del proyecto en cualquier momento y si corresponde, añadirse al repositorio.

Cómo instalar Git

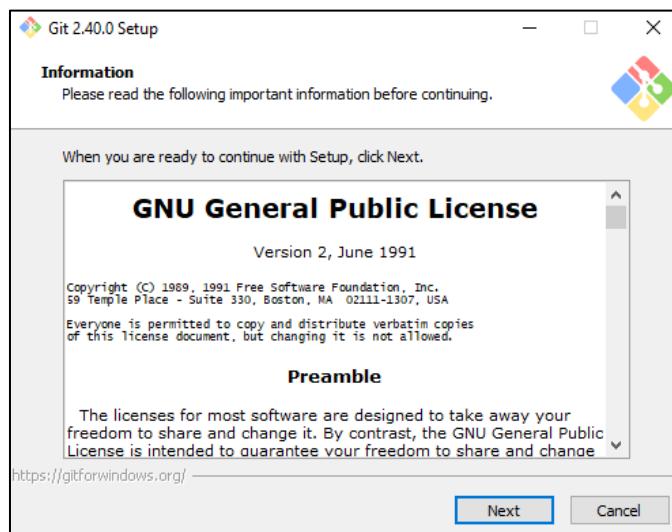
Git está disponible para Windows, Unix/Linux y macOS. En la página oficial del proyecto Git, se encontrarán los archivos de instalación binarios, las instrucciones para instalar el administrador de paquetes y las ediciones portátiles listas para usar para cada sistema operativo. Así como diferentes interfaces gráficas para el sistema de control de versiones, que ofrece la comunidad de Git.

Figura 101: Descarga de Git



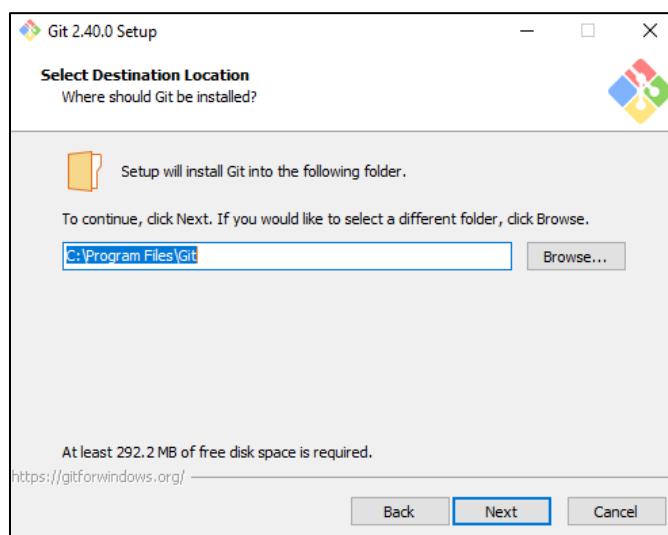
Nota: Adaptado de Git, s.f., <http://git-scm.com/>

Figura 102: Proceso instalación de Git



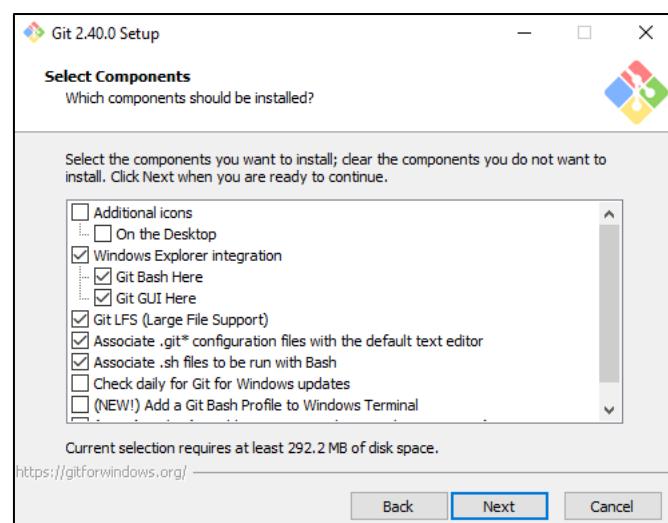
Nota: Adaptado de Git, s.f., <http://git-scm.com/>

Figura 103: Proceso instalación de Git



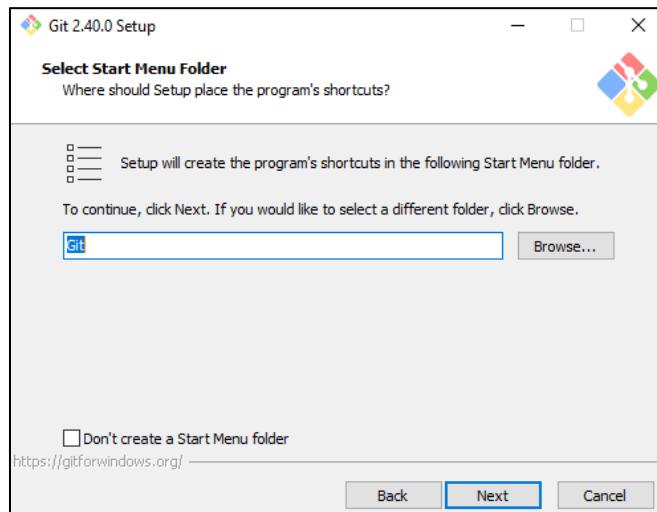
Nota: Elaboración propia

Figura 104: Proceso instalación de Git



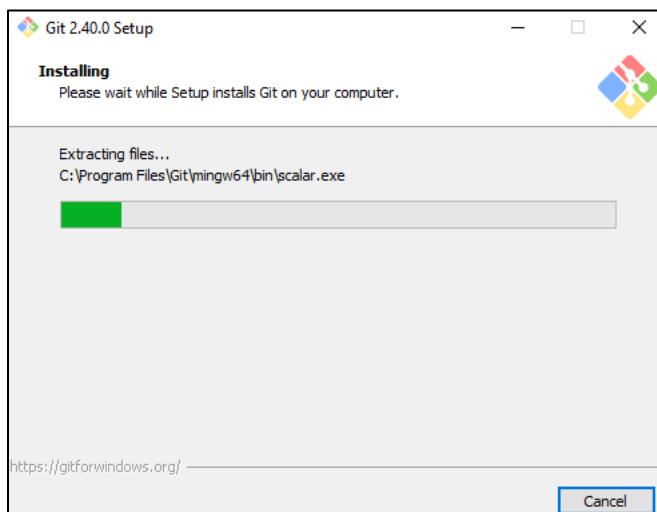
Fuente.- Elaboración propia

Figura 105: Proceso instalación de Git



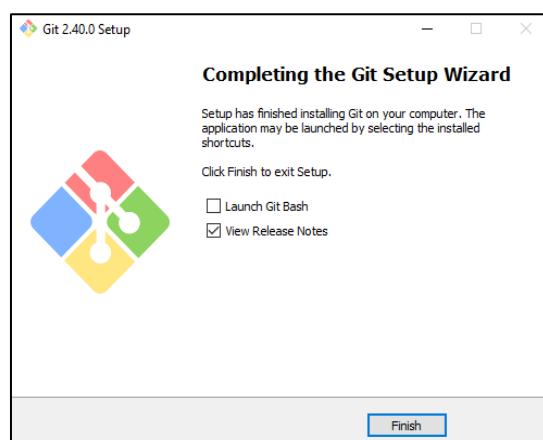
Nota: Elaboración propia

Figura 106: Proceso instalación de Git



Nota: Elaboración propia

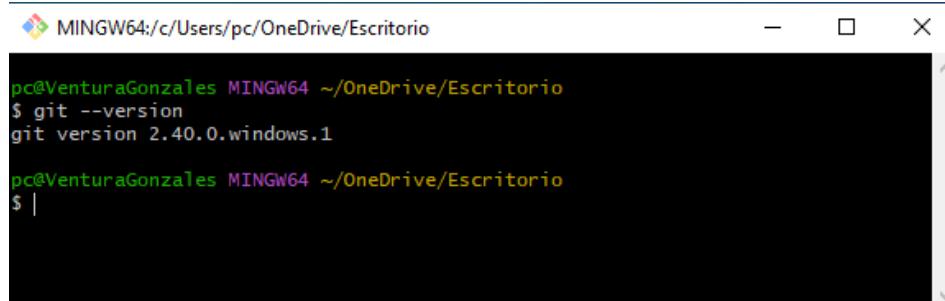
Figura 107: Proceso instalación de Git



Nota: Elaboración propia

Se puede usar la terminal de GIT (git-bash) o la consola (git-cmd)

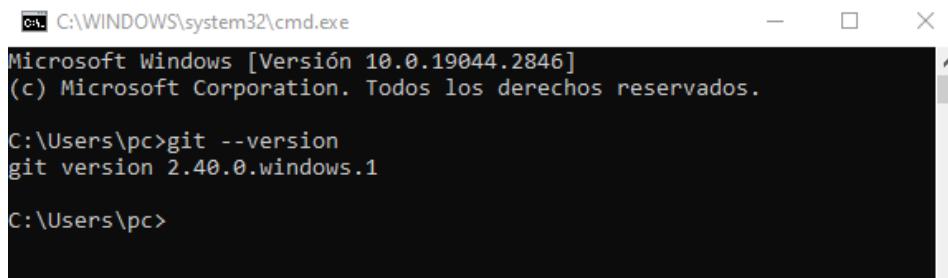
Figura 108: Verificación de versión instalada



```
MINGW64:/c/Users/pc/OneDrive/Escritorio
pc@VenturaGonzales MINGW64 ~/OneDrive/Escritorio
$ git --version
git version 2.40.0.windows.1
pc@VenturaGonzales MINGW64 ~/OneDrive/Escritorio
$ |
```

Nota: Elaboración propia

Figura 109: Ejemplos de las terminales de Git



```
C:\ C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.19044.2846]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pc>git --version
git version 2.40.0.windows.1

C:\Users\pc>
```

Nota: Elaboración Propia

Configuración

Configurar el usuario del control de cambio.

```
git config --global user.name "pjventur"
git config --global user.email "pjventur@cibertec.edu.pe"
```

Verificar.

```
git config --global -l
```

Comandos:

Si se quiere conocer los comandos a utilizar, se puede usar:

```
git help
```

Crear un repositorio Git

El repositorio Git es el directorio central de un proyecto, a través del cual se lleva a cabo el control de todas las versiones.

En la consola, dirigirse a la ruta del proyecto y ejecutar:

```
git init
```

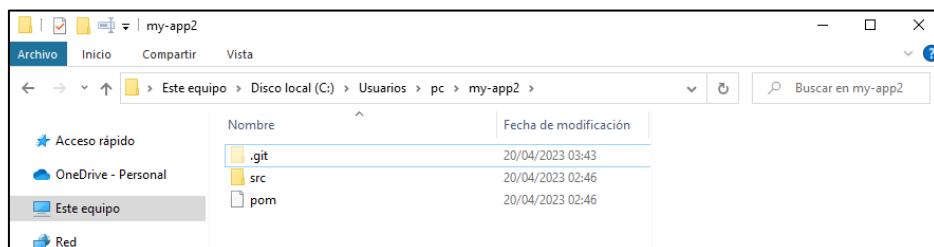
Figura 110: Iniciando el repositorio

```
MINGW64:/c/Users/pc/my-app2
pc@VenturaGonzales MINGW64 ~/my-app2
$ git init
Initialized empty Git repository in C:/Users/pc/my-app2/.git/
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ ...
```

Nota: Elaboración Propia

Se crea una carpeta oculta para el control de versiones.

Figura 111: Carpeta del repositorio



Nota: Elaboración Propia

Si ya existe un repositorio Git para el proyecto, solo hay que introducir el comando `git clone`, seguido de la dirección web o de red de ese repositorio para crear una copia de trabajo.

Comprobar el estado del repositorio

Uno de los conceptos básicos para utilizar Git es organizar, adecuadamente, el propio directorio de trabajo, lo que permite no solo proponer cambios e innovaciones personales a un proyecto, que luego son aceptados mediante el comando `git commit` (enviar), sino también obtener información sobre las actividades de otros usuarios. Se puede comprobar si tu copia de trabajo está actualizada ejecutando este comando:

```
git status
```

Por lo general, en el caso de los repositorios creados recientemente o cuando el repositorio principal, y la copia de trabajo coinciden por completo, recibirás un aviso de que no se ha modificado el proyecto (No commits yet). Además, Git informa de que no has compartido tus cambios para el próximo commit (nothing to commit).

Figura 112: Información del estado del repositorio

```

MINGW64:/c/Users/pc/my-app2
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    pom.xml
    src/

nothing added to commit but untracked files present (use "git add" to track)

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ ...

```

Nota: Elaboración Propia

Agregando archivos

Para añadir un nuevo archivo al control de versiones o para registrar algún cambio para el siguiente commit, introduce el comando git add y el nombre de este archivo. Ejemplo:

```

git add archive
Ejemplo: git add pom.xml

```

Figura 113: Información del estado del repositorio

```

MINGW64:/c/Users/pc/my-app2
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git add pom.xml

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   pom.xml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    src/

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ ...

```

Nota: Elaboración Propia

Podemos comprobar en el estado del repositorio, documentos que está a la espera de someterse a la siguiente fase de confirmación de cambios del proyecto, en que estos se aceptarán o no (Changes to be committed), éstos se encuentran en un espacio intermedio conocido como **stage**.

Se pueden añadir todos los archivos y carpetas usando:

```
git add .
git add carpeta/
git add carpeta/archivo
```

Figura 114: Información del estado del repositorio

```
MINGW64:/c/Users/pc/my-app2
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   pom.xml
    new file:   src/main/webapp/WEB-INF/web.xml
    new file:   src/main/webapp/index.jsp

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$
```

Nota: Elaboración Propia

Si se quiere remover del stage, algún elemento.

```
git reset archivo
git reset carpeta/archivo
```

Confirmando en control de versiones

```
git commit -m "mensaje"
```

Figura 115: Información del estado del repositorio

```
MINGW64:/c/Users/pc/my-app2
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git commit -m "feat: mi primer commit"
[master (root-commit) 1568ef5] feat: mi primer commit
 3 files changed, 79 insertions(+)
 create mode 100644 pom.xml
 create mode 100644 src/main/webapp/WEB-INF/web.xml
 create mode 100644 src/main/webapp/index.jsp

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$
```

Nota: Elaboración Propia

Al visualizar el status, no se observará información:

Figura 116: Información del estado del repositorio

```
MINGW64:/c/Users/pc/my-app2
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git status
On branch master
nothing to commit, working tree clean
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$
```

Nota: Elaboración Propia

Comparando cambios

Ante cualquier cambio en el repositorio, se puede observar lo siguiente:

Figura 117: Información del estado del repositorio después de una modificación

```
MINGW64:/c/Users/pc/my-app2
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   src/main/webapp/index.jsp

no changes added to commit (use "git add" and/or "git commit -a")

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$
```

Nota: Elaboración Propia

git diff

Figura 118: Información de la modificación del repositorio

```
MINGW64:/c/Users/pc/my-app2
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git diff
warning: in the working copy of 'src/main/webapp/index.jsp', LF will be replaced by CRLF the next time Git touches it
diff --git a/src/main/webapp/index.jsp b/src/main/webapp/index.jsp
index c38169b..06134e0 100644
--- a/src/main/webapp/index.jsp
+++ b/src/main/webapp/index.jsp
@@ -1,5 +1,5 @@
 <html>
 <body>
-<h2>Hello World!</h2>
+<h2>Bienvenido al curso de Lenguaje de Programación II</h2>
 </body>
 </html>

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$
```

Nota: Elaboración Propia

Deshacer los cambios realizados

Una vez aceptados los cambios al último commit, se puede editar el contenido o eliminarlo por completo en cualquier momento más adelante.

```
git checkout .
```

Restaura los cambios realizados

Figura 119: Restaurando cambios y muestra es estado del repositorio

```
MINGW64:/c/Users/pc/my-app2
g
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git checkout .
Updated 1 path from the index

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git status
On branch master
nothing to commit, working tree clean

pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$
```

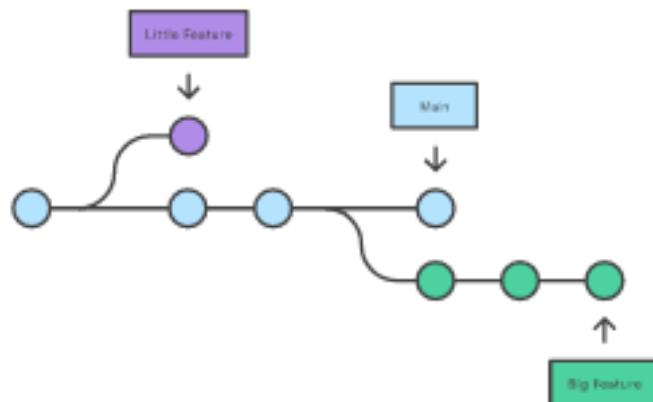
Nota: Elaboración Propia

1.3.2. Gestión de escenarios

Gestión de Ramas

El comando `git branch` te permite crear, enumerar y eliminar ramas, así como cambiar su nombre. No te permite cambiar entre ramas o volver a unir un historial bifurcado.

Figura 120: Ramas en Git



Nota: Adaptado de Atlassian

```
git branch
```

Cambia el nombre de la rama actual a <branch>.

```
git branch -m <branch>
```

Eliminar una rama

```
git branch -d <branch>
```

Moverse entre ramas

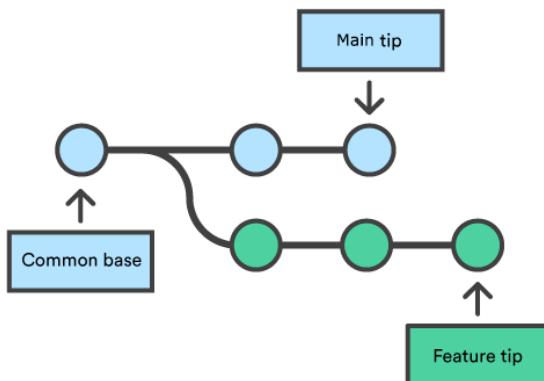
El comando git checkout te permite desplazarte entre las ramas creadas por git Branch

```
git checkout <branch>
```

Fusión de ramas

git merge combinará varias secuencias de confirmaciones en un historial unificado. En los casos de uso más frecuentes, git merge se utiliza para combinar dos ramas.

Figura 121: Fusión de ramas en Git



Nota: Adaptado de Atlassian

```
git merge
```

Gestión de conflictos

Normalmente los conflictos surgen cuando dos personas han cambiado las mismas líneas de un archivo o si un desarrollador ha eliminado un archivo mientras otro lo estaba modificando.

Secuencia de pasos:

Identificación de conflictos:

Figura 122: Comandos Git

```
$ git status
On branch main
You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)

both modified: merge.txt
```

Nota: Adaptado de Atlassian

Verificar archivos encontrados:

Figura 123: Comandos Git

```
$ cat merge.txt
<<<<< HEAD
this is some content to mess with
content to append
=====
totally different content to merge later
>>>>> new_branch_to_merge_later
```

Nota: Adaptado de Atlassian

Resolver los archivos que presenten conflictos.

Cuando se haya terminado de editar los archivos que contienen conflictos, seguir con este comando:

Figura 124: Comandos Git

```
git commit -m "merged and resolved the conflict in merge.txt"
```

Nota: Adaptado de Atlassian

1.3.3. GitHub básico

Git no es GitHub, sino una plataforma de desarrollo colaborativo o también llamada la red social de los desarrolladores donde se alojan los repositorios, el código se almacena de forma pública, pero se puede hacer privado con una cuenta de pago.

Cómo publicar mi proyecto en Github

Paso 1. Ingresar a la página de GitHub y registrarse.

Figura 125: Página oficial de Github

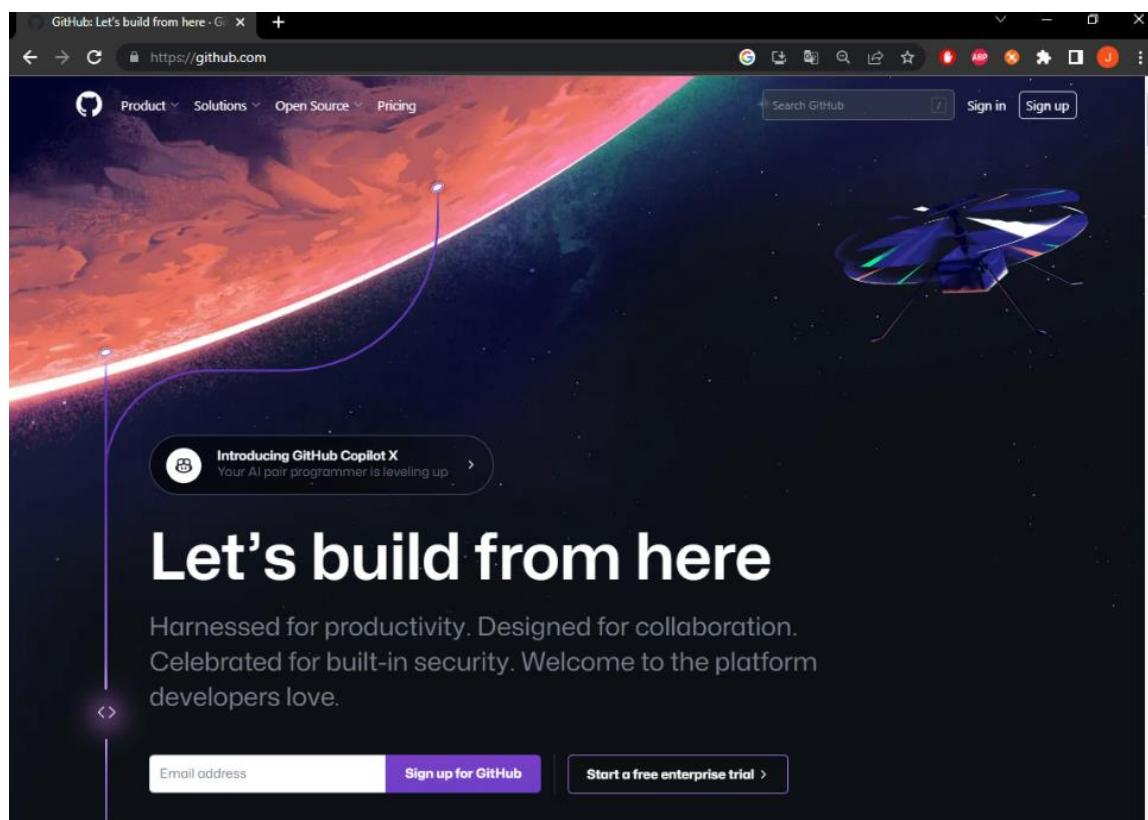
Nota: Adaptado de Github, s.f., <https://github.com/>

Figura 126: Página oficial de Github

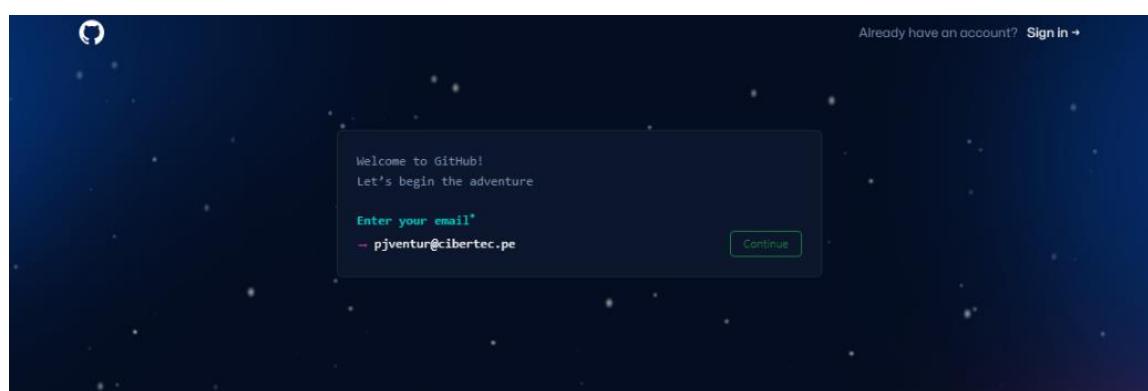
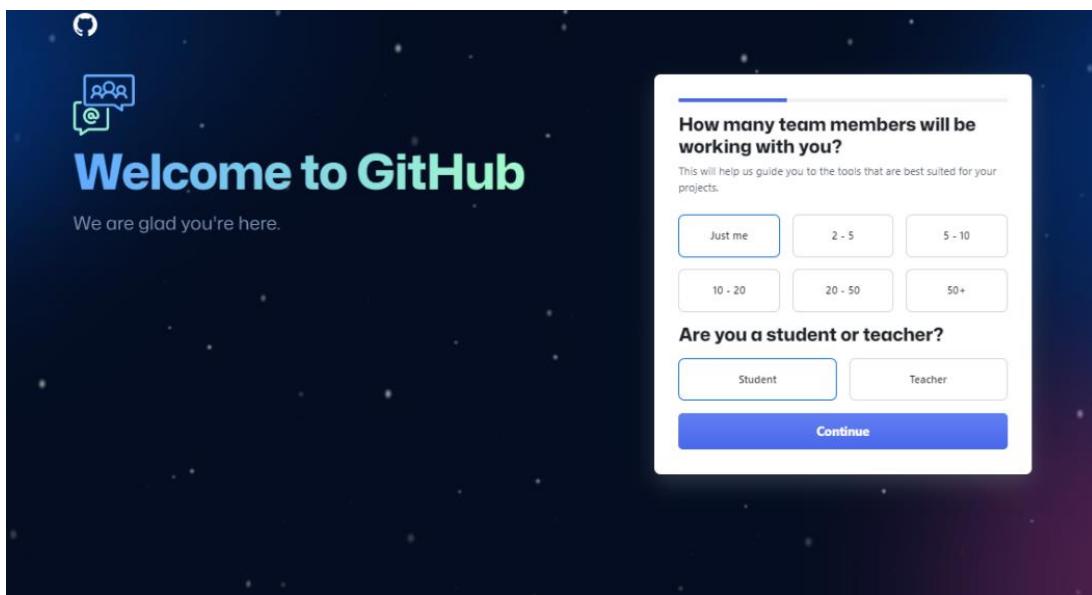
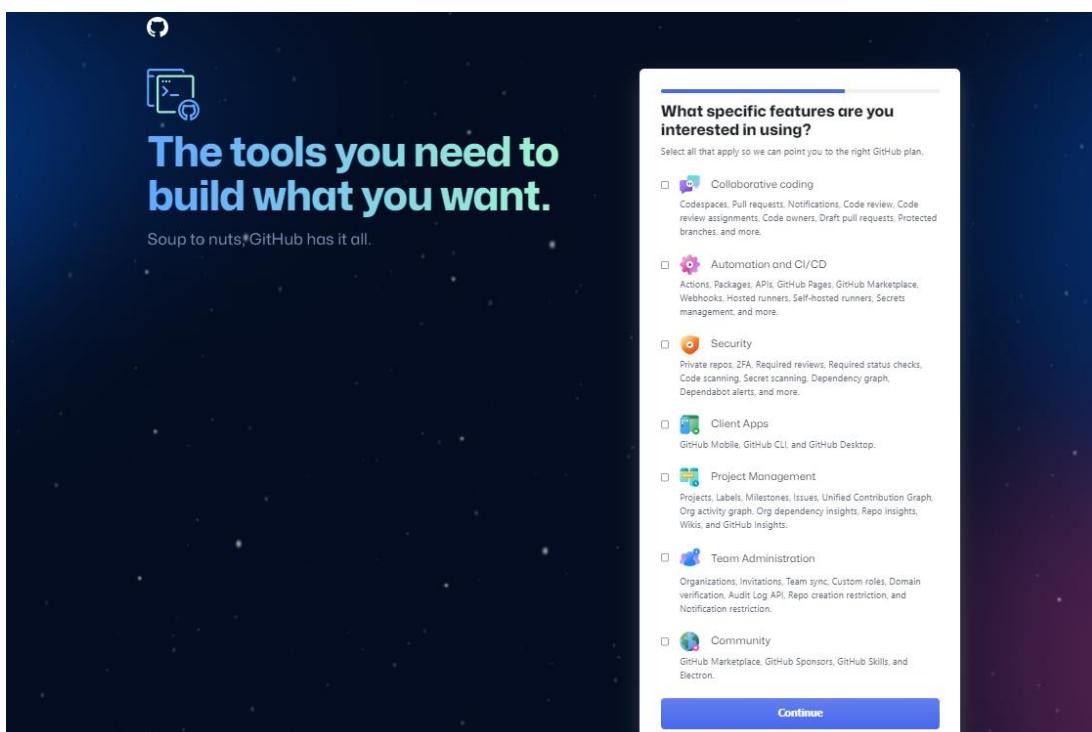
Nota: Adaptado de Github, s.f., <https://github.com/>

Figura 127: Página oficial de Github



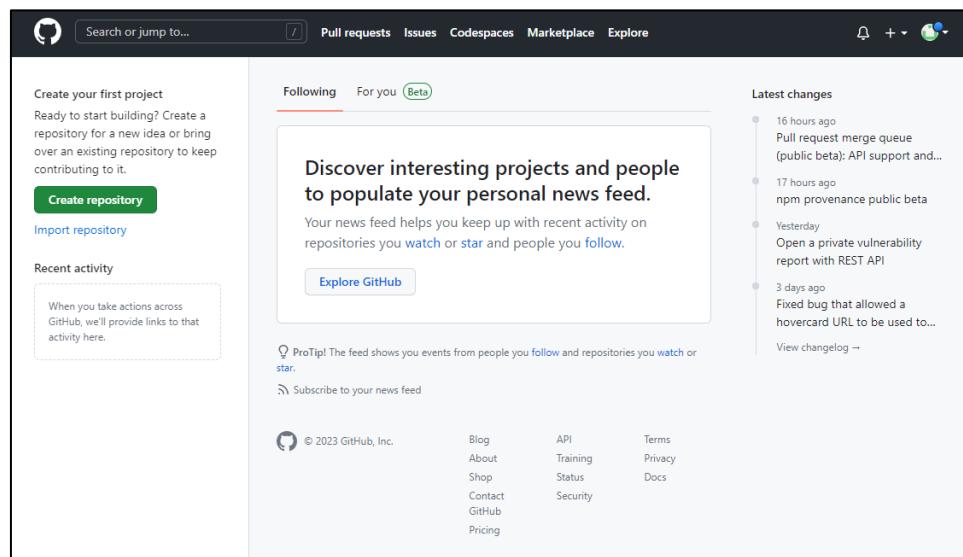
Nota: Adaptado de Github, s.f., <https://github.com/>

Figura 128: Página oficial de Github



Nota: Adaptado de Github, s.f., <https://github.com/>

Figura 129: Espacio de trabajo del usuario



Nota: Adaptado de Github, s.f., <https://github.com/>

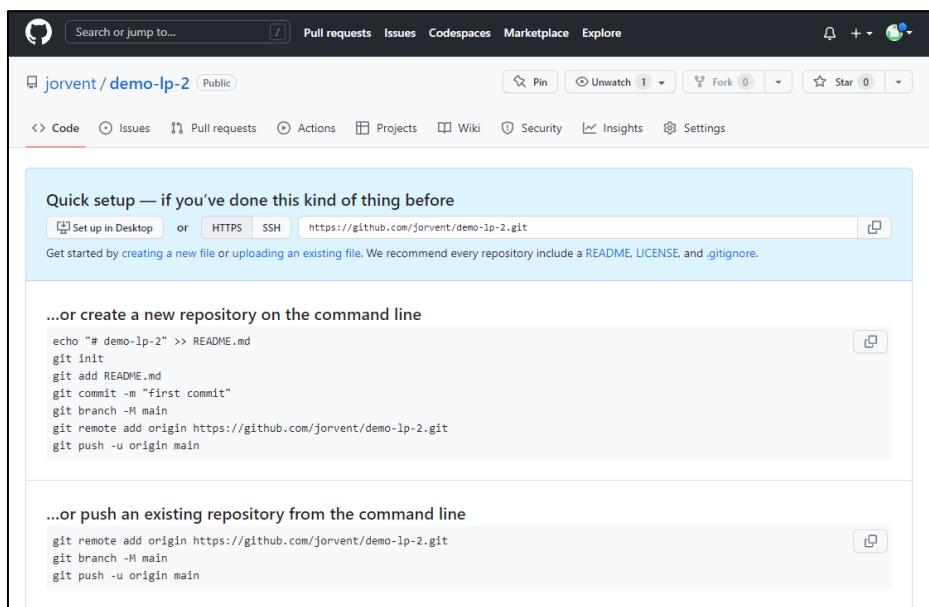
Paso 2. Crear un nuevo repositorio.

Figura 130: Creación de un repositorio en Github

The screenshot shows the 'Create a new repository' form. The title is 'Create a new repository'. It asks if the user already has a project repository elsewhere, with a link to 'Import a repository'. The 'Owner' field is set to 'jorvent' and the 'Repository name' field is empty. Below these fields is a note about repository names being short and memorable, with an example: 'bookish-tribble?'. There's a 'Description (optional)' field with a placeholder. Under 'Visibility', the 'Public' option is selected, with a note that anyone on the internet can see it. The 'Private' option is also available. The 'Initialize this repository with:' section includes a note about skipping if importing an existing repository. There are checkboxes for 'Add a README file' (with a note about writing a long description) and 'Add .gitignore' (with a note about choosing template files). The 'Add .gitignore' section includes a dropdown for '.gitignore template: None'.

Nota: Adaptado de Github, s.f., <https://github.com/>

Figura 131: Creando repositorio



Nota: Adaptado de Github, s.f., <https://github.com/>

Paso 3. Enlazar el repositorio local a GitHub.

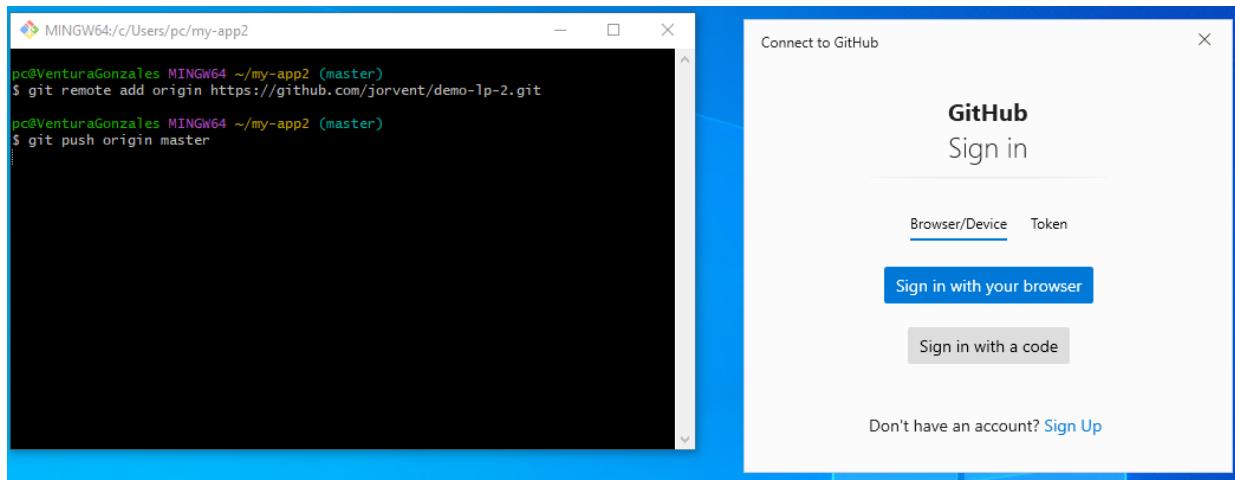
GitHub, presenta una guía del proceso.

```
git remote add origin https://github.com/jorvent/demo-ip-2.git (repositorio)
```

```
git push -u origin master
```

Pedirá las credenciales para subir el repositorio.

Figura 132: Subir cambios al repositorio remoto



Nota: Adaptado de Github, s.f., <https://github.com/>

Figura 133: Verificación de permisos

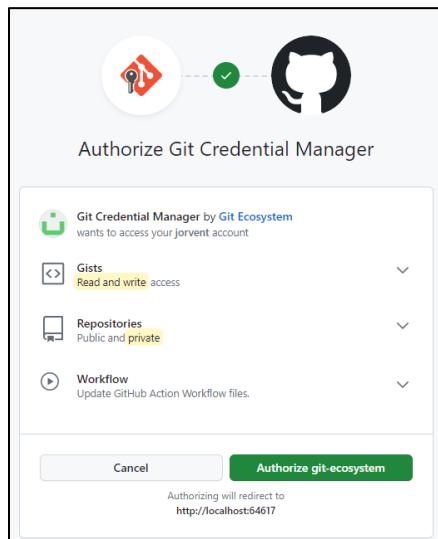
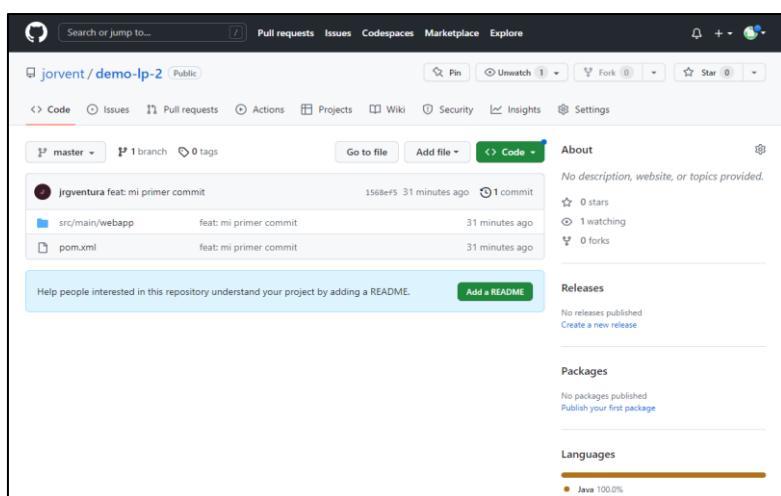
Nota: Adaptado de Github, s.f., <https://github.com/>

Figura 134: Subiendo el repositorio

```
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git remote add origin https://github.com/jorvent/demo-1p-2.git
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ git push origin master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 1.35 KiB | 689.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/jorvent/demo-1p-2.git
 * [new branch]      master -> master
pc@VenturaGonzales MINGW64 ~/my-app2 (master)
$ :
```

Nota: Elaboración Propia

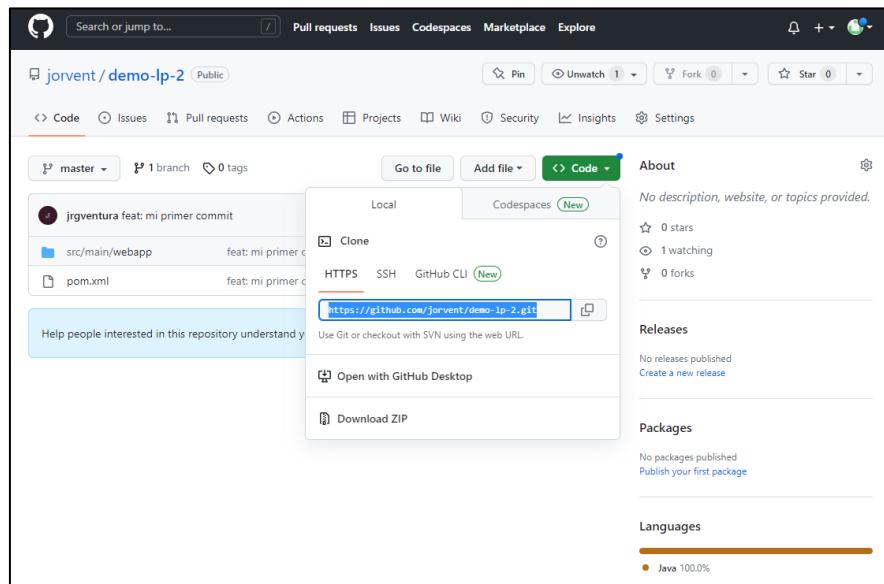
Figura 135: Repositorio con el proyecto

Nota: Adaptado de Github, s.f., <https://github.com/>

Clonar repositorio

Paso 1. En GitHub, abrir la opción de código y copiar la dirección.

Figura 136: Opción del repositorio para clonar o descargar



Nota: Adaptado de Github, s.f., <https://github.com/>

Paso 2. En la consola de Git, ubicamos nuestra carpeta local y escribir el código.

```
git clone https://github.com/jorvent/demo-lp-2.git
```

Posteriormente, realizar las modificaciones, agregar los cambios al control, confirmar colocando un nuevo mensaje y luego, volver a subir.

```
git add .  
git commit -m "nuevos cambios"  
git push origin master
```

Resumen

1. Git es una herramienta, de código abierto, que permite gestionar el versionamiento de código en un proyecto, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.
2. Es una plataforma que permite el alojamiento de proyectos usando el sistema de control de versiones por excelencia, git. Le pertenece a Microsoft y ofrece sus integraciones de manera gratuita.

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://www.atlassian.com/es/git/tutorials/what-is-git>
- <http://git-scm.com/book/en/v2>
- <http://git-scm.com/downloads/guis>
- <https://rogerdudler.github.io/git-guide/index.es.html>



Framework de Persistencia

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno implementa una aplicación java Web utilizando el modelo MVC y toda la funcionalidad provista por los frameworks Spring, JPA e Hibernate.

TEMARIO

- 2.1 Tema 4 : Introducción al Framework Spring**
 - 2.1.1 : Instalación y Core
 - 2.1.2 : Spring Boot
 - 2.1.3 : Uso de plantillas
 - 2.1.4 : Spring DATA (JPA - Hibernate)
 - 2.1.5 : Web
 - 2.1.6 : Transacciones

ACTIVIDADES PROPUESTAS

- Los alumnos implementan diferentes tipos de proyectos usando la estructura Spring Framework.
- Los alumnos implementan diferentes tipos de proyectos integrando Spring, JPA, Hibernate, Security y MySQL.

2.1. INTRODUCCIÓN AL FRAMEWORK SPRING

En la actualidad, para el desarrollo de software hay muchas herramientas, APIs y Frameworks, que permitirán desarrollar aplicaciones en menos tiempo, que sean más robustas y contengan menos errores, haciéndose cargo de todas o casi todas las “complicaciones”.

Spring es el más popular de estos Frameworks Java, proporciona varios módulos los cuales abarcan la mayor parte de las cosas que se debe hacer en cualquiera de las capas de las aplicaciones, desde plantillas para trabajar con JDBC o invocación de Web Services y JMS, pasando por sus propias soluciones, ORM o MVC (web), hasta integración con otros frameworks, como Struts 2, Hibernate, JSF, etc. Todo esto de una forma elegante y haciendo uso de muchos buenos principios de programación. Además, Spring maneja la infraestructura de la aplicación, por lo que solo deberemos preocuparnos de la lógica de ésta (y de la configuración de Spring).

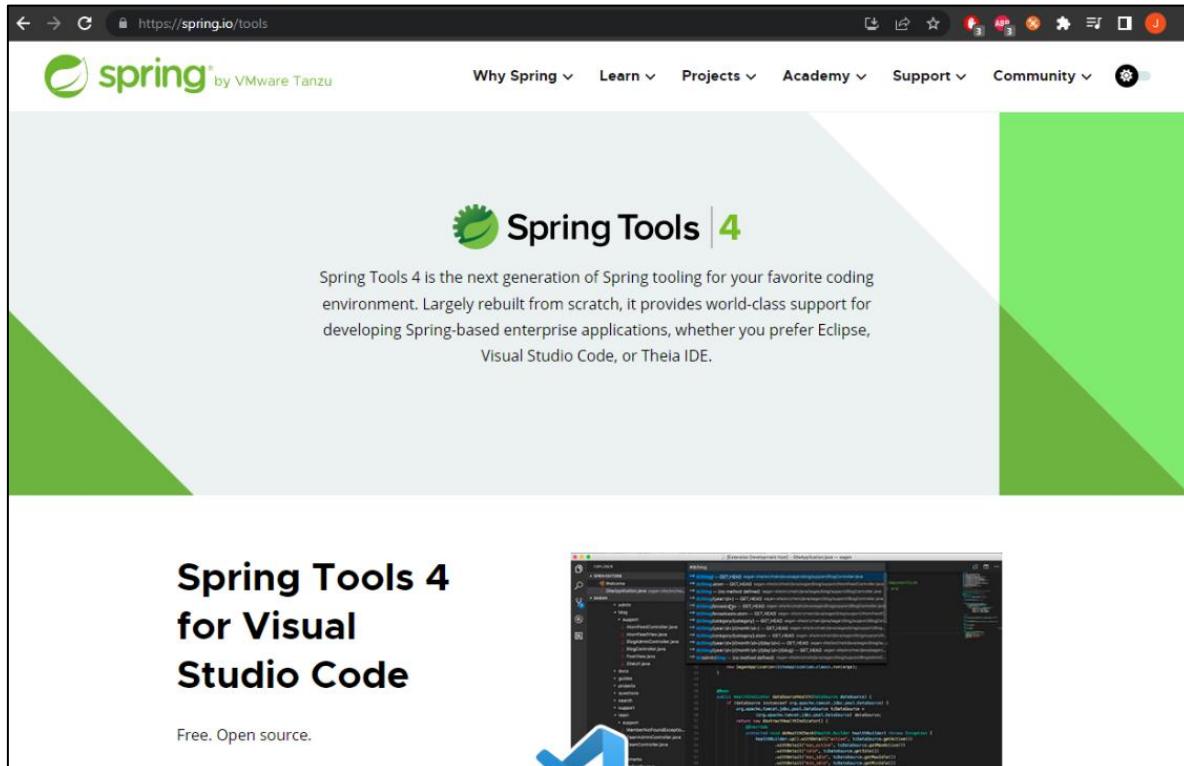
Aunque Spring se encuentra dividido en distintos módulos, cada uno de los cuales se encarga de partes diferentes de nuestra aplicación, esta separación en módulos permite usar solo las partes que necesitamos, sin tener la carga de los que no serán usados.

2.1.1. Instalación y Core

Existen varias maneras de trabajar con Spring. En este caso, se trabajará con las IDEs que proporciona la página oficial.

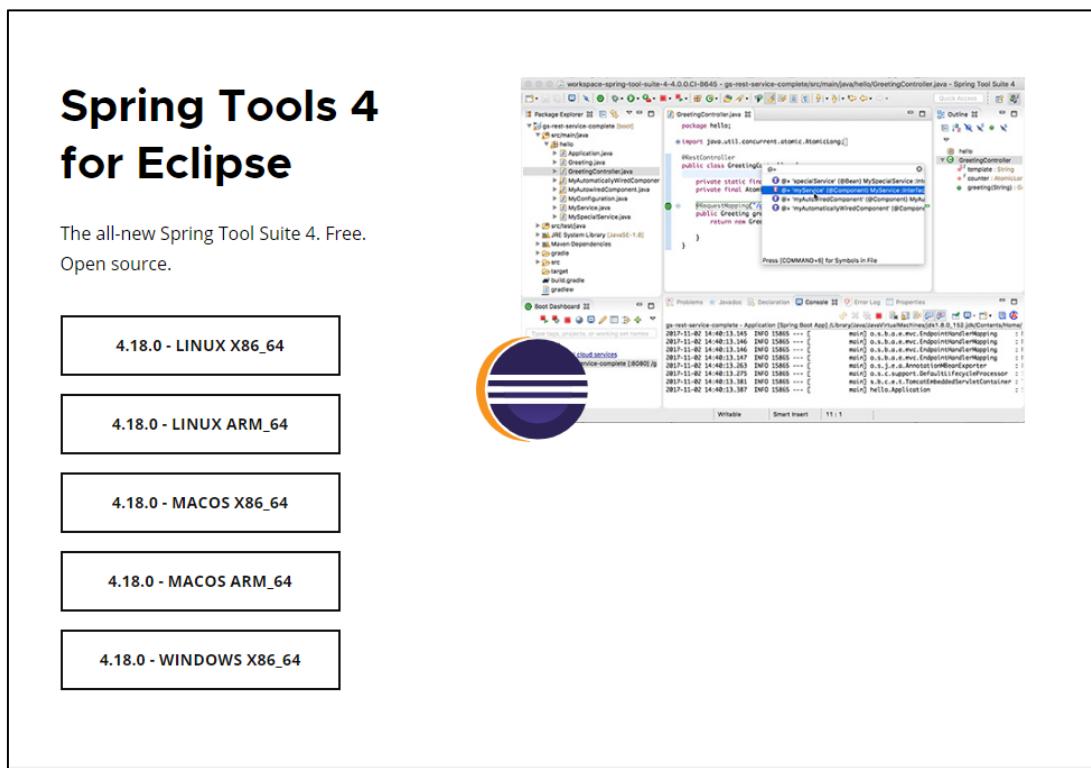
Paso 1. Ingresar a la página oficial y descargar la herramienta.

Figura 137: Página oficial de Spring



Nota: Adaptado de Spring, VMware, Inc., 2023, <https://spring.io/tools>

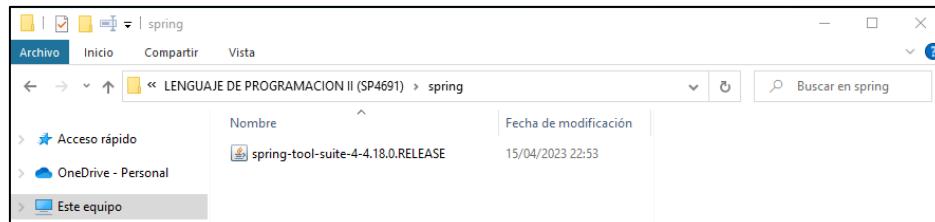
Figura 138: Tools para Eclipse



Nota: Adaptado de Spring, VMware, Inc., 2023, <https://spring.io/tools>

Paso 2. Ejecutar el instalador y esperar a que termine la descarga.

Figura 139: Jar para descomprimir



Nota: Elaboración propia

Figura 140: Jar para descomprimir



Nota: Elaboración propia

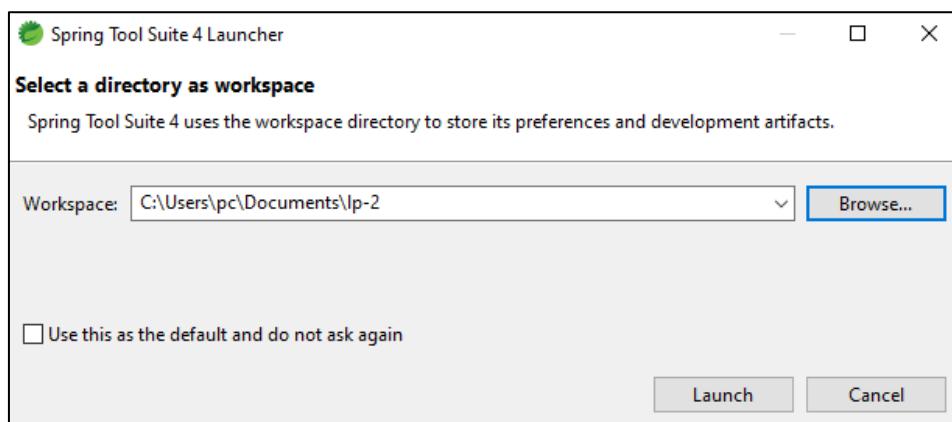
Figura 141: Instalación del STS



Nota: Elaboración Propia

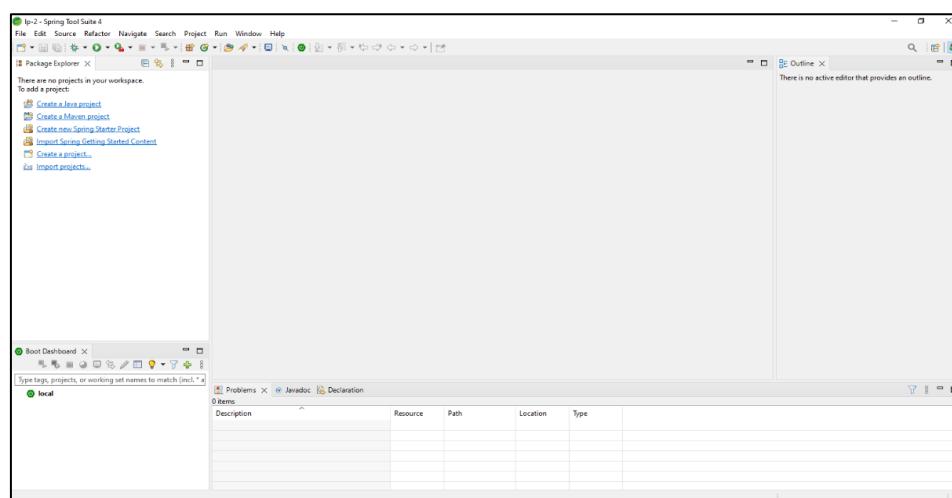
Paso 3. En la carpeta descomprimida, ejecutar la aplicación del **SpringToolSuite4**, al igual que un proyecto eclipse, indicar el Workspace y al aceptar, esperar a que cargue la pantalla principal.

Figura 142: Workspace del STS



Nota: Elaboración Propia

Figura 143: Pantalla principal del STS



Nota: Elaboración Propia

2.1.2. Spring Boot

Es una solución para el framework Spring de Java que sigue el principio de “convención sobre configuración”, reduciendo la complejidad presentada al momento de desarrollar proyectos basados en Spring.

Spring Boot proporciona la estructura básica configurada del proyecto, que incluye las pautas para usar el Framework y todas las dependencias relevantes para la aplicación, lo que permite simplificar mucho la creación de aplicaciones standalone o web.

Las características principales de Spring Boot:

- Incorporación directa de aplicaciones de servidores web/contenedores como Apache Tomcat, eliminando la necesidad de incluir archivos WAR (Web Application Archive).
- Simplificación de la configuración de Maven gracias a los POM (Project Object Models).
- Configuración automática de Spring en la medida de lo posible.

Creando proyecto desde la IDE STS

Paso 1. Clic en la opción **File/New** y seleccionar **Spring Starter Project**.

Figura 144: Creación nuevo proyecto

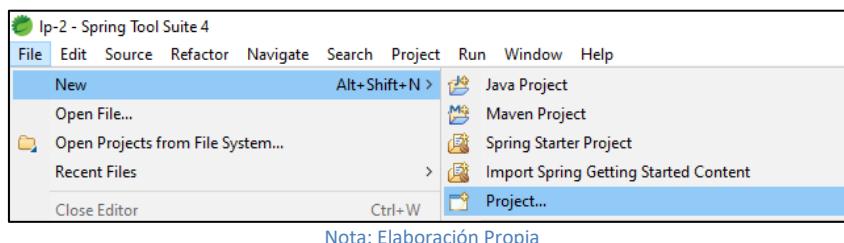
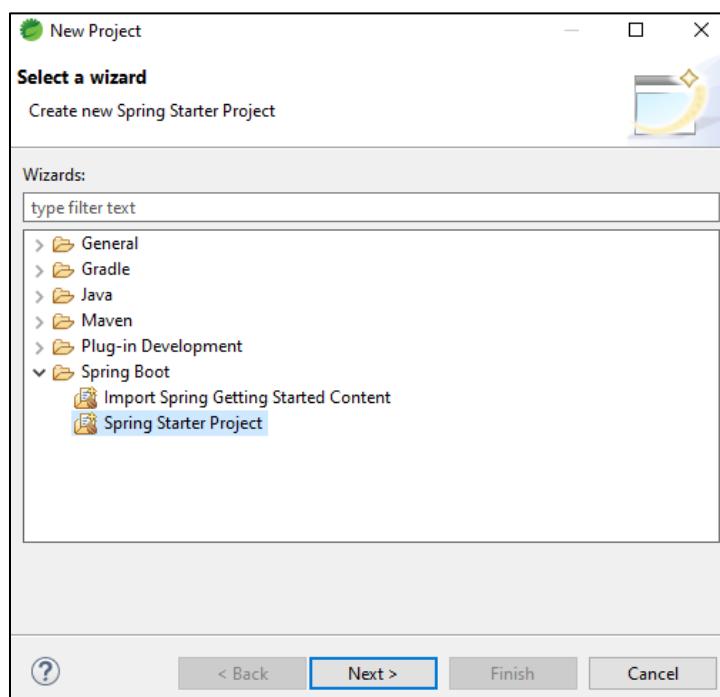
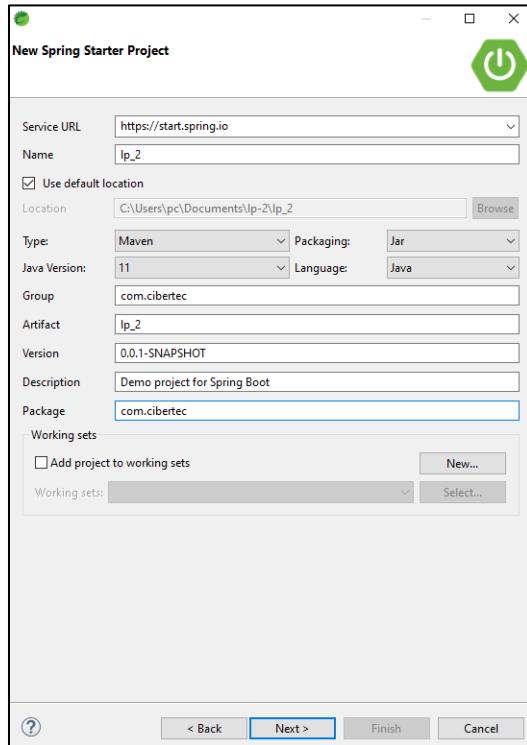


Figura 145: Creación nuevo proyecto



Paso 2. Escribir los datos, seleccionando el tipo **Maven** y el paquete Jar (empaquetado standalone o web) o War (JSP y depende de la estructura WEB-INF, entre otros).

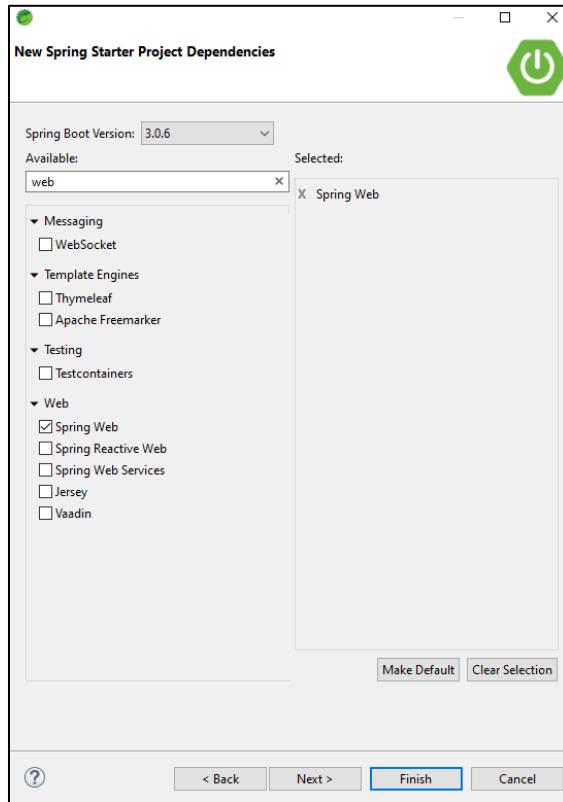
Figura 146: Ejemplo de nuevo proyecto



Nota: Elaboración Propia

En el siguiente paso, agregar las dependencias.

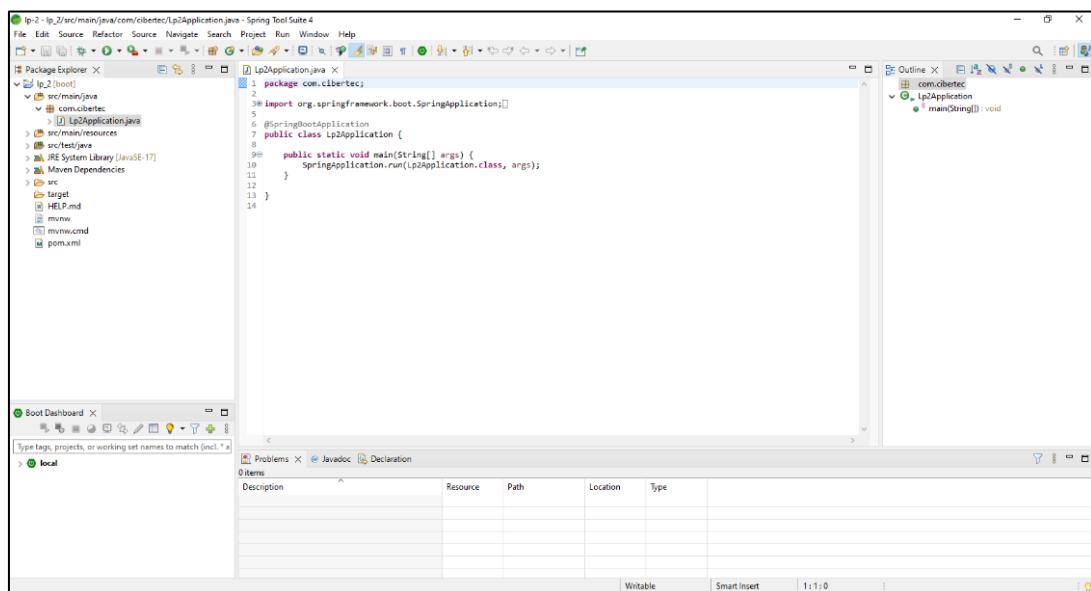
Figura 147: Pantalla principal del STS



Nota: Elaboración Propia

Finalmente, observar la estructura del proyecto generado.

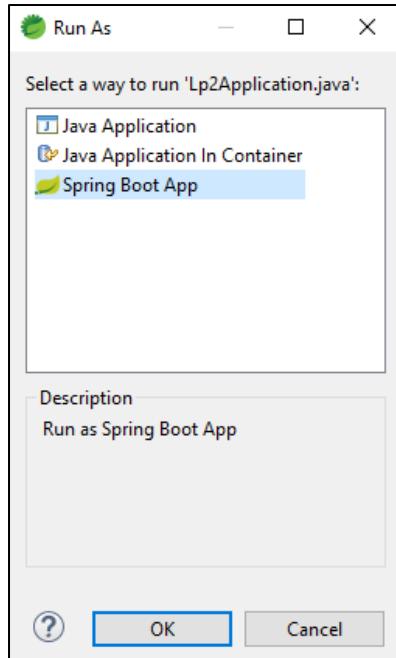
Figura 148: Pantalla principal del STS con el proyecto generado



Nota: Elaboración Propia

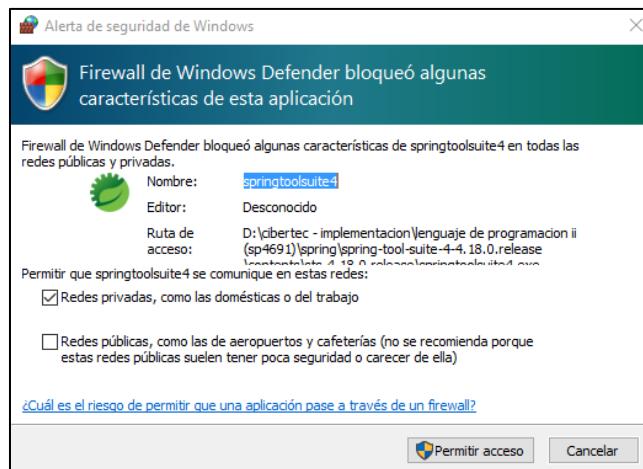
Paso3. Ejecutar el proyecto.

Figura 149: Pantalla principal del STS



Nota: Elaboración Propia

Figura 150: Permiso de Windows



Nota: Elaboración Propia

Spring Boot, ejecutará la aplicación y la subirá al Apache Tomcat embebido, pero si muestra error de puerto en uso, se puede configurar el puerto, agregando propiedades en la carpeta recursos:

Figura 151: Consola indicando error de puerto

```
<terminated> dawi_clase05 - DawiClase05Application [Spring Boot App] C:\Recursos-DAWI\sts-4.9.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.wi  
APPLICATION FAILED TO START  
*****  
Description:  
Web server failed to start. Port 8080 was already in use.  
Action:  
Identify and stop the process that's listening on port 8080 or configure this application to listen on another port.
```

Nota: Elaboración Propia

Figura 152: Configuramos el puerto editando el archivo de propiedades

```
File Edit Source Navigate Search Project Run Window Help  
Package Explorer X Lp2Application.java application.properties X  
1 server.port = 8020  
2
```

Nota: Elaboración Propia

Si no hay problema, aparecerá en la consola la información de la ejecución.

Figura 153: Consola ejecutando sin error

```

lp_2 - lp_2/src/main/resources/application.properties - Spring Tool Suite 4
File Edit Source Navigate Project Run Window Help
Package Explorer X
lp_2 [boot]
src/main/java
com.cibertec
lp2Application.java
src/main/resources
static
templates
application.properties
src/test/java
JRE System Library [JavaSE-17]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml
Boot Dashboard X
Type tags, projects, or working set names:
local

1 server-port = 8020
2

Problems Declaration Console
lp_2 - Lp2Application [Spring Boot App] [pid: 11976]

::: Spring Boot :::
2023-04-20T11:14:56.716+05:00 INFO 11976 --- [main] com.cibertec.lp2Application : Starting Lp2Application using Java 17.0.6 with PID 11976 (C:\Users\pc\Documents\lp_2)
2023-04-20T11:14:56.721+05:00 INFO 11976 --- [main] com.cibertec.lp2Application : No active profile set, falling back to 1 default profile: "default"
2023-04-20T11:14:57.411+05:00 INFO 11976 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8020 (http)
2023-04-20T11:14:57.411+05:00 INFO 11976 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-04-20T11:14:57.411+05:00 INFO 11976 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.8]
2023-04-20T11:14:57.510+05:00 INFO 11976 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Root WebApplicationContext: initialization completed in 737 ms
2023-04-20T11:14:57.810+05:00 INFO 11976 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8020 (http) with context path ''
2023-04-20T11:14:57.820+05:00 INFO 11976 --- [main-8020-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Started Lp2Application in 1.444 seconds (process running for 2.09)
2023-04-20T11:15:06.774+05:00 INFO 11976 --- [main-8020-exec-1] o.s.web.servlet.DispatcherServlet : Initializing DispatcherServlet 'dispatcherServlet'
2023-04-20T11:15:06.775+05:00 INFO 11976 --- [main-8020-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-04-20T11:15:06.776+05:00 INFO 11976 --- [main-8020-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 0 ms

```

Nota: Elaboración Propia

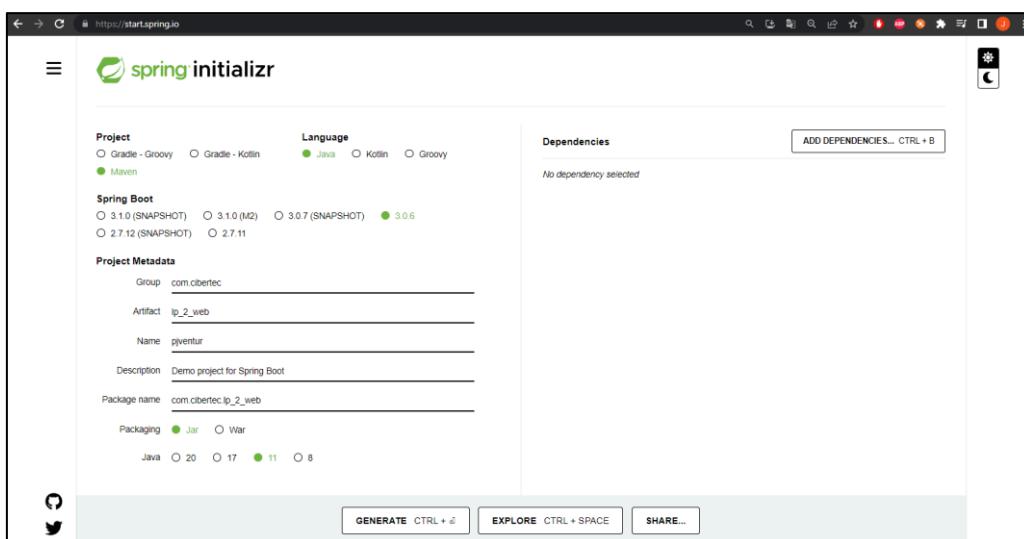
2.1.3. Uso de plantillas

Creando proyecto desde SpringInitializr

Otra forma de crear los proyectos es utilizando la plantilla **Initializr** desde la página de Spring.

Paso 1. En la página, establecer las propiedades de la aplicación que se van a desarrollar con Spring Boot, así como, todas las dependencias relevantes (bases de datos, características de seguridad, interfaces web, servicios en la nube, etc.)

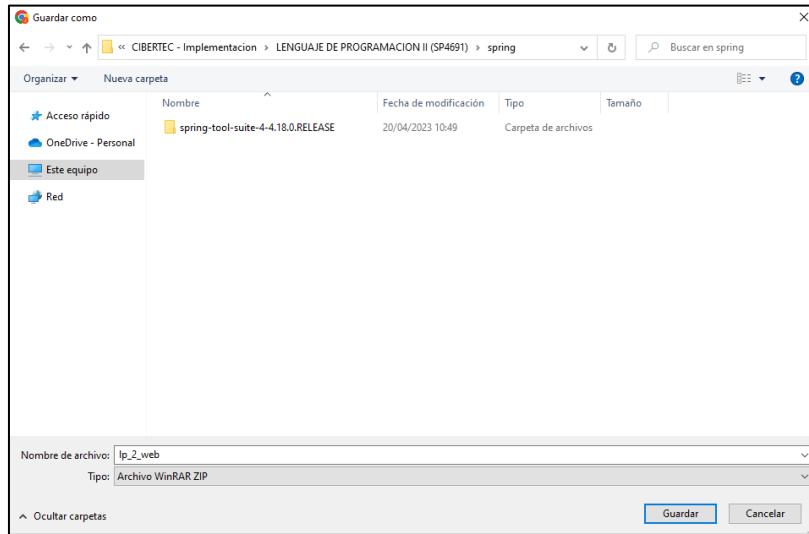
Figura 154: Creando un proyecto con Initializr



Nota: Adaptado de spring, s.f., <https://start.spring.io/>

Finalmente, haz clic en “Generate Project” para crear los archivos del proyecto. Al pulsar **generate**, se crea el proyecto y se descargará.

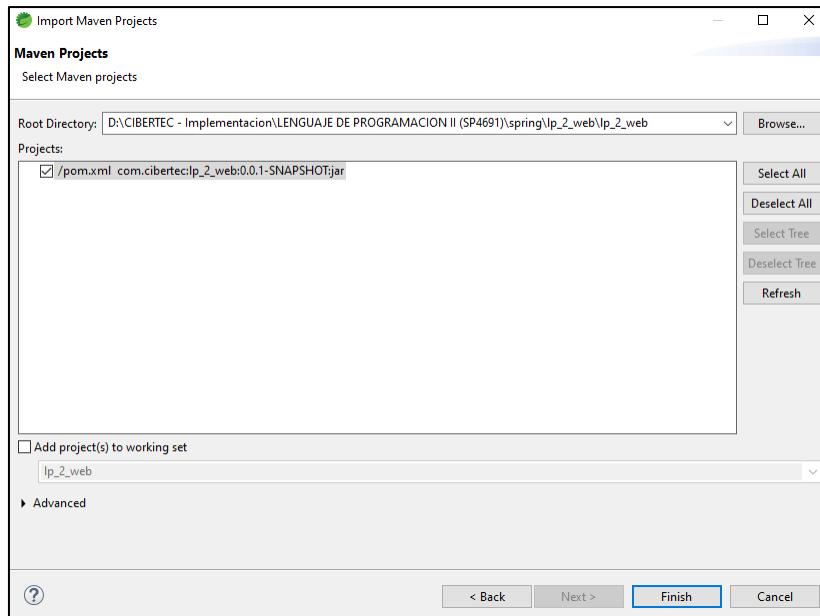
Figura 155: Descargamos el proyecto



Nota: Elaboración Propia

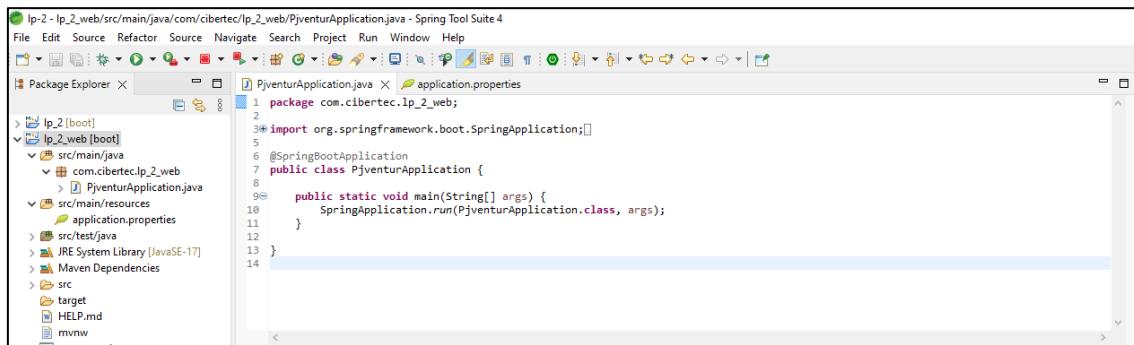
Paso 2. Descomprimir el proyecto e importar a STS como un proyecto Maven.

Figura 156: Importando el proyecto creado con Initializr



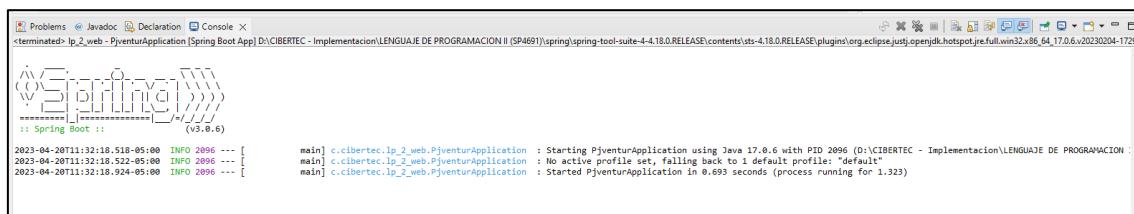
Nota: Elaboración Propia

Figura 157: Proyecto creado con Initializr



Nota: Elaboración Propia

Figura 158: Proyecto creado con Initializr



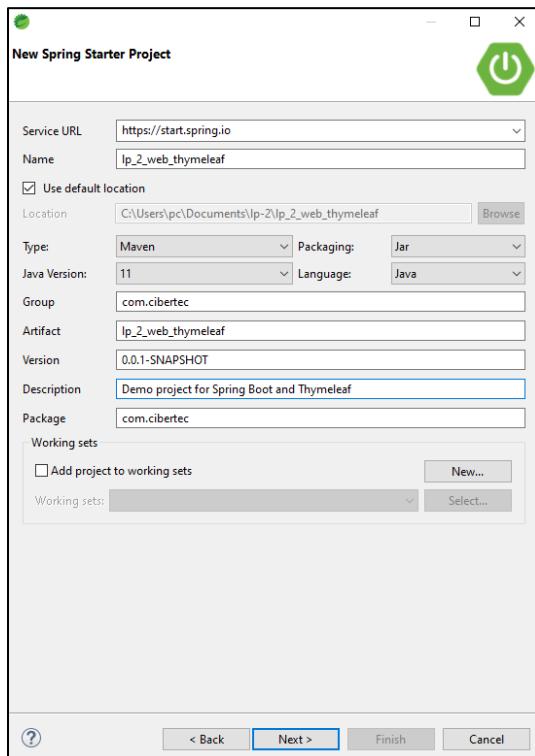
Nota: Elaboración Propia

Thymeleaf

Thymeleaf es el motor de plantillas de Spring Framework que permite trabajar de una forma más natural con html. Se acopla muy bien para trabajar en la capa vista del MVC de aplicaciones web.

Paso 1. Crear un nuevo proyecto.

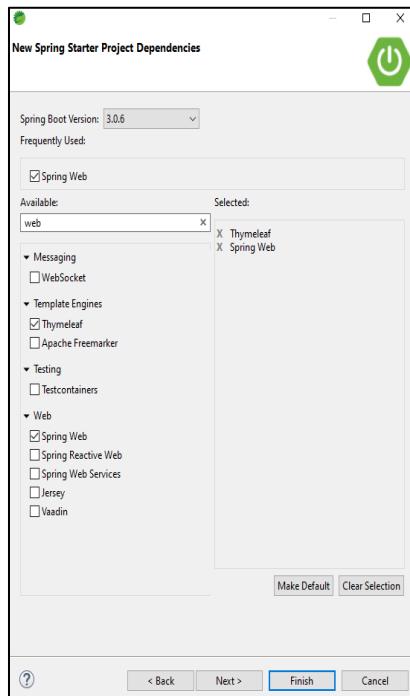
Figura 159: Nuevo proyecto



Nota: Elaboración Propia

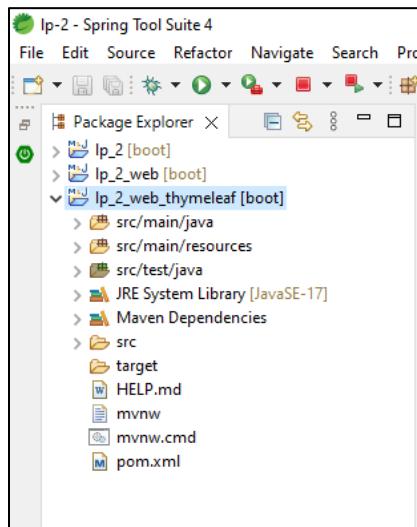
Paso 2. En el siguiente paso, seleccionar las dependencias (Web y Thymeleaf)

Figura 160: Asignamos las dependencias



Nota: Elaboración Propia

Figura 161: Estructura del proyecto



Nota: Elaboración Propia

Paso 3. Agregar una clase controladora, para lo cual se creará, de ser necesario, el paquete respectivo y se realizará la edición. En este paso, tomar la plantilla que proporciona la página oficial de Spring.

Figura 162: Página Spring, con la plantilla del controlador Web

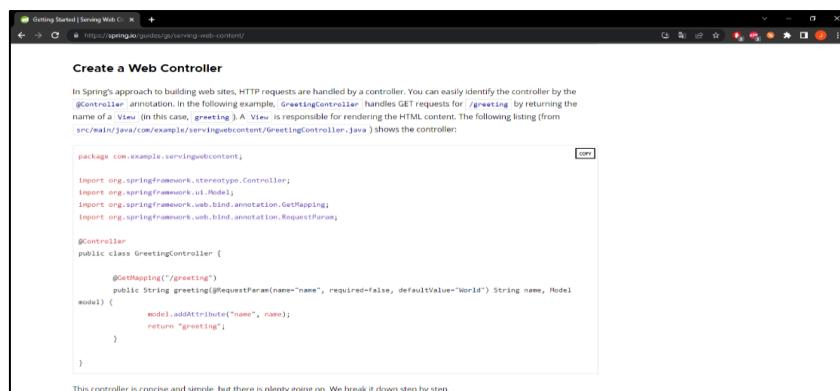
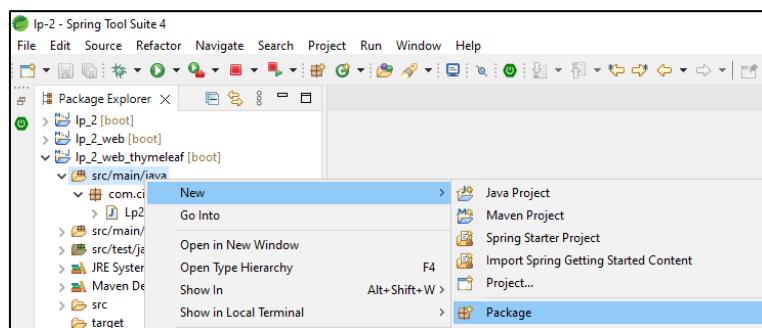
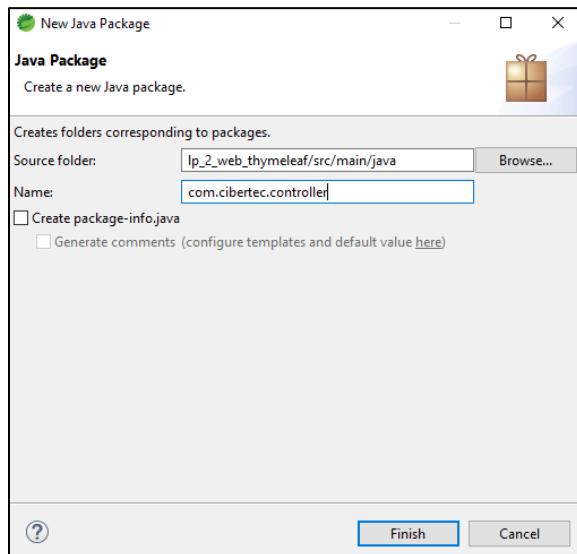
Nota: Adaptado de Spring, s.f., <https://spring.io/guides/gs/serving-web-content/>

Figura 163: Crear nuevos paquetes



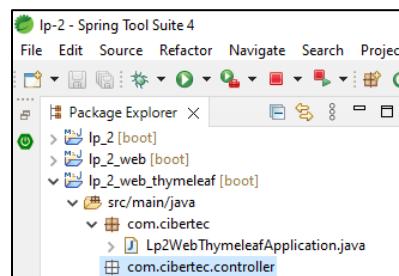
Nota: Elaboración Propia

Figura 164: Crear nuevos paquetes



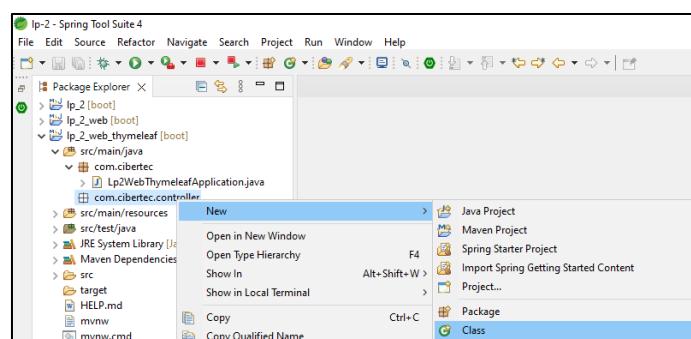
Nota: Elaboración Propia

Figura 165: Crear nuevos paquetes



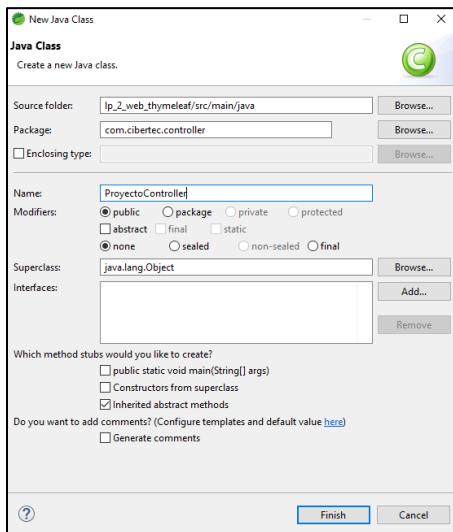
Nota: Elaboración Propia

Figura 166: Crear nuevas clases



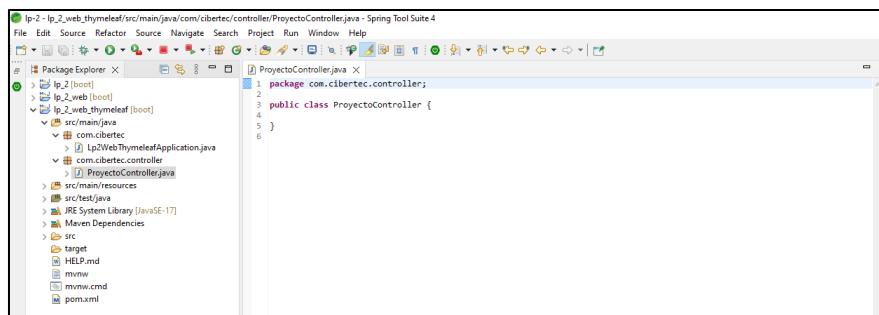
Nota: Elaboración Propia

Figura 167: Crear nuevos paquetes



Nota: Elaboración Propia

Figura 168: Clase controladora



Nota: Elaboración Propia

Utilizar el código plantilla de la página oficial.

Figura 169: Clase controladora

```

1 package com.cibertec.controller;
2
3
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestParam;
8
9 @Controller
10 public class ProyectoController {
11
12     @GetMapping("/greeting")
13     public String greeting(@RequestParam(name="name", required=false, defaultValue="World") String name, Model model) {
14         model.addAttribute("name", name);
15         return "greeting";
16     }
17
18 }
```

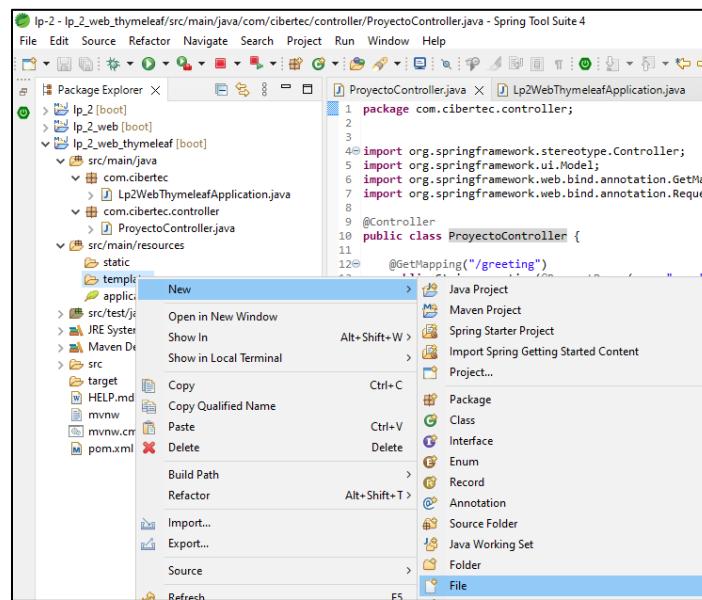
Nota: Elaboración Propia

Donde:

- `@RequestParam`, hace referencia al parámetro de entrada **name** enviado al alias que al no ser obligatorio tiene un valor por default.
- El parámetro se captura en la variable del mismo nombre **name** y se asocia al modelo del controlador.
- `model.addAttribute`, genera un **atributo** de salida que será enviado a la página de respuesta.

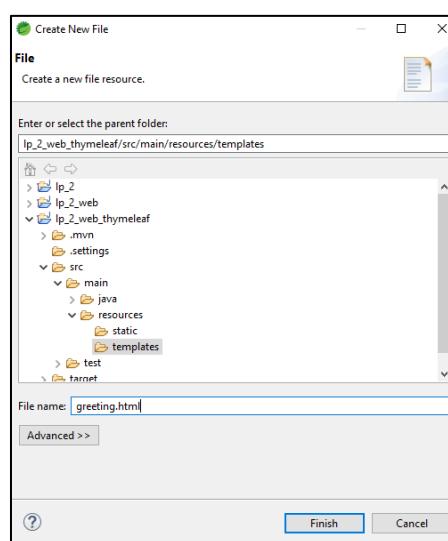
Paso 4. Generar el archivo html en la carpeta **template** del proyecto.

Figura 170: Creación nuevo archivo



Nota: Elaboración Propia

Figura 171: Creación nuevo archivo



Noya: Elaboración Propia

Figura 172: Clase controladora y archivo html plantilla

```

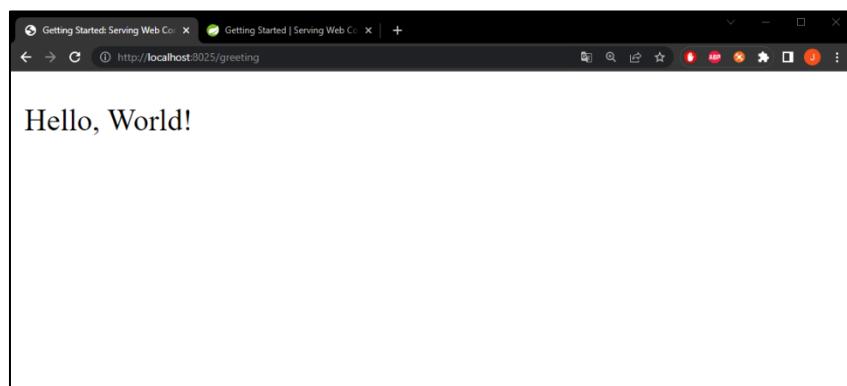
1 <!DOCTYPE HTML>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4   <title>Getting Started: Serving Web Content</title>
5   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
6 </head>
7 <body>
8   <p th:text="Hello, ' + ${name} + '!">
9 </body>
10 </html>

```

Nota: Elaboración Propia

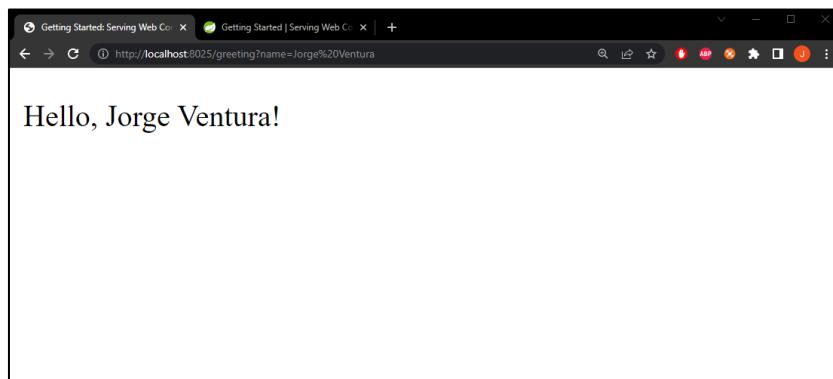
Paso 5. Ejecutar el proyecto y observar su ejecución.

Figura 173: Página ejecutándose



Nota: Elaboración Propia

Figura 174: Ejemplo de ejecución sin parámetros y con parámetros



Nota: Elaboración Propia

Es importante mencionar que en el controlador se agregará la lógica de programación o acceso a datos.

2.1.4. Spring DATA (JPA - Hibernate)

Spring Data es uno de los módulos del Framework Spring, cuyo objetivo es simplificar al desarrollador la persistencia de datos contra distintos repositorios de información.

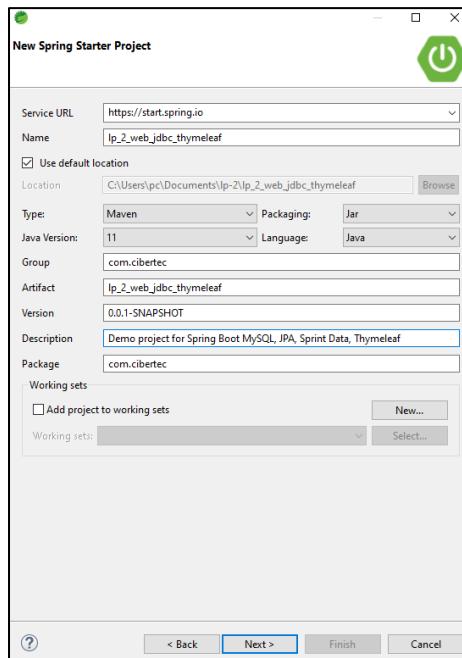
Características:

- Soporte sofisticado para construir repositorios basados en Spring y JPA.
- Soporte para predicados QueryDSL y por lo tanto, consultas JPA con seguridad de tipos.

- Auditoría transparente de la clase de dominio.
- Soporte de paginación, ejecución de consultas dinámicas, capacidad para integrar código de acceso a datos personalizados.
- Validación de @Queryconsultas anotadas en el momento del arranque.
- Soporte para mapeo de entidades basado en XML.
- Configuración de repositorio basada en JavaConfig introduciendo @EnableJpaRepositories.

Paso 1. Nuevo proyecto web con JPA, MySQL, Thymeleaf.

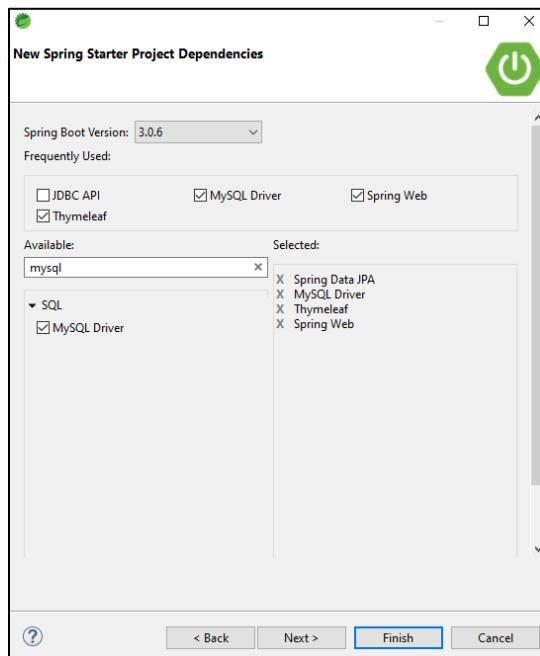
Figura 175: Creando nuevo proyecto



Nota: Elaboración Propia

Paso 2. Dependencias.

Figura 176: Agregando dependencias



Nota: Elaboración Propia

Paso 3. Configurar el archivo de propiedades.

Spring Boot da valores predeterminados en todas las cosas. En consecuencia, cuando se deseé utilizar cualquier base de datos, se deben definir los atributos de conexión en los archivos **application.properties**.

Figura 177: Archivo properties

```
application.properties X
1 server.port= 8025
2
3 #hibernate
4 spring.jpa.hibernate.ddl-auto=update
5
6 #mysql
7 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
8 spring.datasource.url=jdbc:mysql://localhost:3307/dbcursos?useSSL=false&serverTimezone=UTC
9 spring.datasource.username=root
10 spring.datasource.password=123456|
```

Nota: Elaboración Propia

En este caso, **spring.jpa.hibernate.ddl-auto** puede ser **none**, **update**, **create**, o **create-drop**.

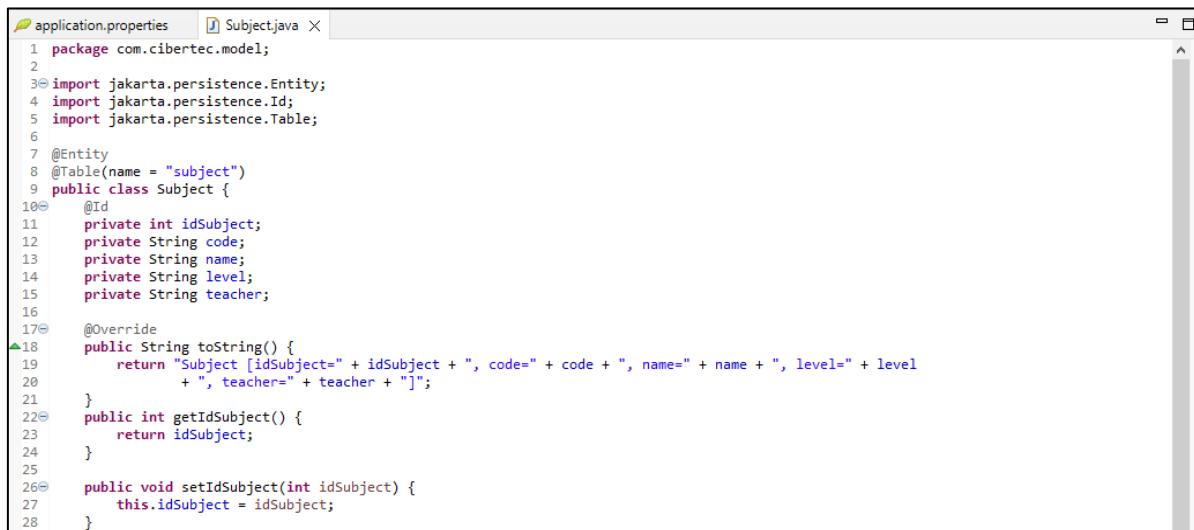
Donde:

- **none**: el predeterminado para MySQL. No se realiza ningún cambio en la estructura de la base de datos.
- **update**: Hibernate cambia en la base de datos, de acuerdo con las estructuras de entidades dadas.
- **create**: crea la base de datos cada vez, pero no la cierra.
- **create-drop**: crea la base de datos y la suelta cuando se cierra SessionFactory.

Debe comenzar con create o update, porque aún no tiene la estructura de la base de datos. Después de la primera ejecución, puede cambiarlo a update o none, según los requisitos del programa.

Use update cuando desee realizar algún cambio en la estructura de la base de datos.

Paso 4. Entidades. Al igual que cuando se desarrolló JPA, definir las entidades y sus anotaciones, así como los métodos de acceso get y set de todos los campos y opcionalmente el método `toString` para realizar un control.



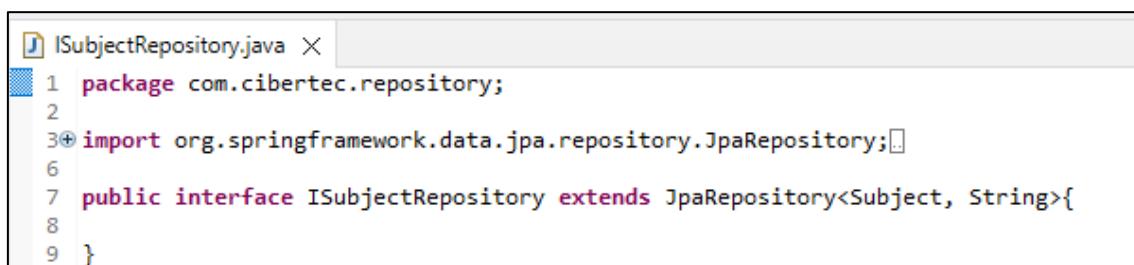
```

application.properties Subject.java
1 package com.cibertec.model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.Id;
5 import jakarta.persistence.Table;
6
7 @Entity
8 @Table(name = "subject")
9 public class Subject {
10     @Id
11     private int idSubject;
12     private String code;
13     private String name;
14     private String level;
15     private String teacher;
16
17     @Override
18     public String toString() {
19         return "Subject [idSubject=" + idSubject + ", code=" + code + ", name=" + name + ", level=" + level
20             + ", teacher=" + teacher + "]";
21     }
22     public int getIdSubject() {
23         return idSubject;
24     }
25
26     public void setIdSubject(int idSubject) {
27         this.idSubject = idSubject;
28     }
}

```

Tener presente que se pueden agregar más anotaciones y valores de control.

Paso 5. Interface de repositorio que se encargará de los métodos del proceso del CRUD; para ello, extender de clases como **JpaRepository** o **CrudRepository**, en donde indica el nombre de la entidad a controlar y el tipo del campo clave.



```

ISubjectRepository.java
1 package com.cibertec.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface ISubjectRepository extends JpaRepository<Subject, String>{
6
7 }

```

Paso 6. Clase controladora, donde se definirán las acciones y los alias de los procesos. En este ejemplo, se realizan los procesos de registro de datos fijos a la tabla cursos y un listado de toda la tabla.

```

1 package com.cibertec.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestParam;
8
9 import com.cibertec.model.Subject;
10 import com.cibertec.repository.ISubjectRepository;
11
12 @Controller
13 public class ProyectoController {
14
15     @Autowired
16     private ISubjectRepository repos;
17
18     @GetMapping("/registrar")
19     public String registrarSubject(@RequestParam(name= "name", required = false, defaultValue = "Subject") String name, Model model) {
20
21         Subject subj = new Subject();
22         subj.setIdSubject(1);
23         subj.setCode("CODE0102");
24         subj.setName("Lenguaje de Programación II");
25         subj.setLevel("básico");
26         subj.setTeacher("Jorge Ventura");
27
28         subj = repos.save(subj);
29
30         model.addAttribute("name", subj);
31         return "greeting";
32     }
33
34     @GetMapping("/listar")
35     public String listarSubject(Model model) {
36
37         model.addAttribute("lstSubject", repos.findAll());
38
39         return "listado";
40     }
41 }

```

Paso 7. Páginas, se diseñan las páginas registro.html y listado.html, en la carpeta de **templates** del **src/main/resources**,

```

1 <!DOCTYPE HTML>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <title>Getting Started: Serving Web Content</title>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
6 </head>
7 <body>
8     <p th:text="'Registro, ' + ${name.idsubject} + '!'></p>
9 </body>
10 </html>

```

En la página listado.html, se utiliza un foreach para leer el atributo enviado desde el controlador y recorrerlo para mostrar cada elemento.

```

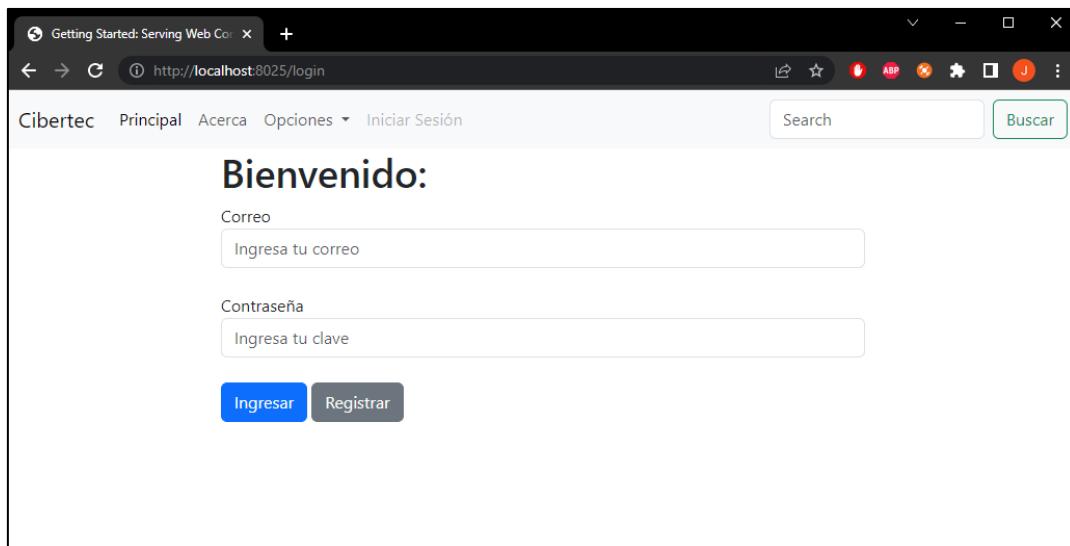
1 <!DOCTYPE HTML>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <title>Getting Started: Serving Web Content</title>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
6 </head>
7 <body>
8     <table>
9         <th:block th:each="p: ${lstCursos}">
10             <tr>
11                 <td th:text="${p.idSubject}"></td>
12                 <td th:text="${p.name}"></td>
13             </tr>
14         </th:block>
15     </table>
16 </body>
17 </html>

```

2.1.5. Web

Para trabajar con formularios, usar th:action="@{/url}" para indicar la URL a la que se enviarán los datos del formulario, al presionar el botón Submit. th:object="\${atributo}" Permite establecer a que atributo del modelo estará enlazado el formulario, th:field="*{campo}" permite enlazar una propiedad del objeto al que esté vinculado el formulario a un elemento de este formulario, por ejemplo, enlazar un campo de texto a la propiedad nombre.

Figura 178: Ejemplo de página web con estilos



Nota: Elaboración Propia

La cual será diseñada de la siguiente forma:

```

<div style="margin-left: 20%; width: 60%;">
  <h1>Bienvenido:</h1>
  <form th:action="@{/login}" method="post">
    <p th:if="${errors}" th:text="Usuario o clave incorrecto"></p>
    <div class="form-group">
      <label for="correo">Correo</label>
      <input type="email" name="usuario" placeholder="Ingresa tu correo" class="form-control">
    </div>
    <br>
    <div class="form-group">
      <label for="contraseña">Contraseña</label>
      <input type="password" name="clave" placeholder="Ingresa tu clave" class="form-control">
    </div>
    <br>
    <button class="btn btn-primary" type="submit">Ingresar</button>
    <button type="button" class="btn btn-secondary">Registrar</button>
    <br>
  </form>
  <br>
</div>
  
```

2.1.6. Transacciones

"Las manipulaciones (las escrituras) de datos en una base de datos se hacen dentro de una transacción. Una transacción permite agrupar operaciones elementales en una única operación más grande. Si todas las operaciones elementales tienen éxito, la transacción se valida (se habla de commit). Si alguna de las operaciones elementales fracasa, la transacción considera que la operación global no se ha completado y se anulará todas las operaciones elementales (se habla de rollback)" (Déléchamp, 2018)

Entendemos entonces que en una aplicación empresarial una transacción se le denomina a la serie de pasos que se realizan en un sistema para ejecutar una operación.

Transacciones con Spring:

"Claramente, tiene opciones que tomar: si usar transacciones locales o transacciones globales y, si está usando transacciones globales. ¿Cuál deberías usar?

Spring proporciona gestión de transacciones programática y declarativa. El enfoque programático expone la implementación subyacente de ITA o no ITA mediante la interfaz:

- org.springframework.transaction.PlatformTransactionManager

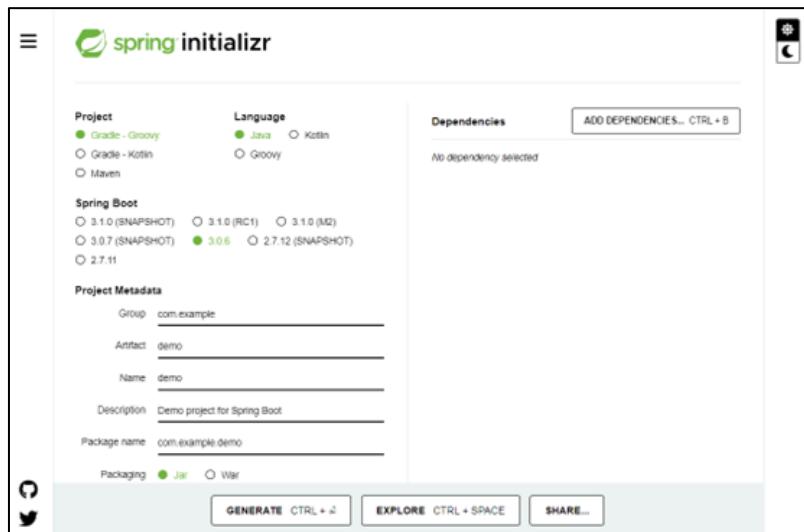
y las subclases relacionadas. Esto proporciona una API consistente para manejar todas las transacciones de manera similar, indiferente a cómo la API subyacente expone la transacción. Las transacciones declarativas se aplican utilizando Programación Orientada a Aspectos (AOP) para definir los límites de la transacción. Este enfoque permite un control de transacciones mucho menos invasivo, ya que parte de la codificación directamente en la API subyacente". (Lui, 2011)

Implementación con Spring:

Paso 1: Cree un proyecto Spring Boot

En este paso, crearemos un proyecto de arranque de primavera. Para esto, usaremos Spring Initializr. Para crear un proyecto Spring Boot, consulte [¿Cómo crear un proyecto Spring Boot?](#)

Figura 179: Creación de Proyecto



Nota: Elaboración Propia

Paso 2: agregar dependencias

Agregaremos las dependencias requeridas para nuestra aplicación Spring Boot.

Paso 3: configurar la base de datos

Implementaremos usando las siguientes configuraciones y las agregaremos a nuestro archivo **application.properties**.

Figura 180: Configuración de Proyecto

```

server.port = 9090

#mysql
spring.datasource.url=jdbc:mysql://localhost:3306/employee_db
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect

#ddl-auto:update: creará el esquema de entidad y lo asignará a la db automáticamente
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

```

Nota: Elaboración Propia

Paso 4: Crear Clase Modelo

Empleado.java

```

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructorConstructor;
import lombok.Getter;
import lombok.NoArgsConstructorConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@NoArgsConstructorConstructor
@AllArgsConstructorConstructor
@ToString
@Entity
@Table(name="EMP_INFO")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String name;

}

```

Dirección.java

```

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
import lombok.AllArgsConstructorConstructor;
import lombok.Getter;
import lombok.NoArgsConstructorConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@NoArgsConstructorConstructor
@AllArgsConstructorConstructor
@ToString
@Entity
@Table(name="ADD_INFO")
public class Address {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String address;

    // one to one mapping means,
    // one employee stays at one address only
    @OneToOne
    private Employee employee;

}

```

Paso 5: crear una capa de base de datos

```

import org.springframework.data.jpa.repository.JpaRepository;
import com.geeksforgeeks.transactionmanagement.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {
}

```

AddressRepository.java

```

import org.springframework.data.jpa.repository.JpaRepository;
import com.geeksforgeeks.transactionmanagement.model.Address;

public interface AddressRepository extends JpaRepository<Address, Integer> {
}

```

Paso 6: crear una capa de servicio

EmployeeService.java

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.geeksforgeeks.transactionmanagement.model.Address;
import com.geeksforgeeks.transactionmanagement.model.Employee;
import com.geeksforgeeks.transactionmanagement.repository.EmployeeRepository;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;

    @Autowired
    private AddressService addressService;

    @Transactional
    public Employee addEmployee(Employee employee) throws Exception {
        Employee employeeSavedToDB = this.employeeRepository.save(employee);

        Address address = new Address();
        address.setId(123L);
        address.setAddress("Varanasi");
        address.setEmployee(employee);

        // calling addAddress() method
        // of AddressService class
        this.addressService.addAddress(address);
        return employeeSavedToDB;
    }
}

```

AddressService.java

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.geeksforgeeks.transactionmanagement.model.Address;
import com.geeksforgeeks.transactionmanagement.repository.AddressRepository;

@Service
public class AddressService {

    @Autowired
    private AddressRepository addressRepository;

    public Address addAddress(Address address) {
        Address addressSavedToDB = this.addressRepository.save(address);
        return addressSavedToDB;
    }

}

```

Paso 7: Crear controlador

Controlador.java

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.geeksforgeeks.transactionmanagement.model.Employee;
import com.geeksforgeeks.transactionmanagement.service.EmployeeService;

@RestController
@RequestMapping("/api/employee")
public class Controller {

    @Autowired
    private EmployeeService employeeService;

    @PostMapping("/add")
    public ResponseEntity<Employee> saveEmployee(@RequestBody Employee employee)
            throws Exception{
        Employee employeeSavedToDB = this.employeeService.addEmployee(employee);
        return new ResponseEntity<Employee>(employeeSavedToDB,
                HttpStatus.CREATED);
    }
}

```

Paso 8: Ejecutar nuestra aplicación

```

Console X Problems Debug Shell Terminal Servers Gradle Tasks Gradle Executions Git Staging
TransactionManagementApplication [Java Application] C:\Users\palan\p2pool\plugins\org.eclipse.jdt.openjdk.hotspot.re.full.win32.x86_64_17.0.2.v20220201-1208\re\bin\java.exe (15-Jan-2023, 20:51 pm)
2023-01-15T14:10:00.072+05:30 INFO 12048 --- [ restartedMain] org.hibernate.Version                    : HHH000412: Hibernate Core ver
2023-01-15T14:10:00.359+05:30  WARN 12048 --- [ restartedMain] org.hibernate.orm.deprecation       : HHH0000021: Encountered deprecate
2023-01-15T14:10:01.016+05:30  INFO 12048 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource      : HikariPool-1 - Starting...
2023-01-15T14:10:01.016+05:30  INFO 12048 --- [ restartedMain] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection c
2023-01-15T14:10:01.048+05:30  INFO 12048 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource      : HHH0000400: Using dialect: org.hib
2023-01-15T14:10:01.048+05:30  WARN 12048 --- [ restartedMain] org.hibernate.orm.deprecation       : HHH0000026: MySQLDialect has b
Hibernate: create table add_info (id bigint not null, address varchar(255), employee_id integer, primary key (id)) engine=InnoDB
Hibernate: create table add_info_seq (next_val bigint) engine=InnoDB
Hibernate: insert into add_info_seq values ( 1 )
Hibernate: create table emp_info (id integer not null, name varchar(255), primary key (id)) engine=InnoDB
Hibernate: create table emp_info_seq (next_val bigint) engine=InnoDB
Hibernate: insert into emp_info_seq values ( 1 )
Hibernate: alter table add_info add constraint FKm3qr2uljsypowcg8lj6vhesii foreign key (employee_id) references emp_info (id)
2023-01-15T14:10:02.834+05:30  INFO 12048 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform impl
2023-01-15T14:10:02.845+05:30  INFO 12048 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean: Initialized JPA EntityManagerFact
2023-01-15T14:10:03.303+05:30  WARN 12048 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enable
2023-01-15T14:10:03.905+05:30  INFO 12048 --- [ restartedMain] o.s.b.d.a.OptionalLocalReloadServer      : LiveReload server is running on p
2023-01-15T14:10:03.993+05:30  INFO 12048 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9090 ( ...
2023-01-15T14:10:04.008+05:30  INFO 12048 --- [ restartedMain] c.g.t.TransactionManagementApplication   : Started TransactionManagementAppl

```

Paso 9: Interrumpir la transacción

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.geeksforgeeks.transactionmanagement.model.Address;
import com.geeksforgeeks.transactionmanagement.model.Employee;
import com.geeksforgeeks.transactionmanagement.repository.EmployeeRepository;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;

    @Autowired
    private AddressService addressService;

    @Transactional
    public Employee addEmployee(Employee employee) throws Exception {
        Employee employeeSavedToDB = this.employeeRepository.save(employee);

        // we will initialize the
        // address object as null
        Address address = null;
        address.setId(123L);
        address.setAddress("Varanasi");
        address.setEmployee(employee);

        // calling addAddress() method
        // of AddressService class
        this.addressService.addAddress(address);
        return employeeSavedToDB;
    }
}

```

Paso 10: Gestión de transacciones

TransactionManagementApplication.java

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@SpringBootApplication
@EnableTransactionManagement
public class TransactionManagementApplication {

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

EmployeeService.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.geeksforgeeks.transactionmanagement.model.Address;
import com.geeksforgeeks.transactionmanagement.model.Employee;
import com.geeksforgeeks.transactionmanagement.repository.EmployeeRepository;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;

    @Autowired
    private AddressService addressService;

    @Transactional
    public Employee addEmployee(Employee employee) throws Exception {
        Employee employeeSavedToDB = this.employeeRepository.save(employee);

        // we will initialize the
        // address object as null
        Address address = null;
        address.setId(123L);
        address.setAddress("Varanasi");
        address.setEmployee(employee);

        // calling addAddress() method
        // of AddressService class
        this.addressService.addAddress(address);
        return employeeSavedToDB;
    }
}
```

Resumen

1. **Spring**, es un framework web MVC Open Source para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java, que facilita la integración con java, kotlin y groovy.
2. **Spring Boot**, es la herramienta para crear aplicaciones web y también aplicaciones microservicios de una manera rápida y fácil.
3. **SpringInitializr**, es una herramienta que nos ayuda en la creación de aplicaciones Spring, desde cero, configurando sus dependencias de una manera simple y todo desde la web. Podemos descargar el proyecto e importarlo en nuestro IDE.
4. **Thymeleaf**, es un motor de plantillas xml, xhtml, html5 que nos permite poder implementar aplicaciones en java. Se integra fácilmente con aplicaciones MVC.

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://start.spring.io/>
- <https://spring.io/guides/gs/serving-web-content/>
- <https://spring.io/projects/spring-data-jdbc>
- <https://spring.io/projects/spring-data-jpa>
- <https://www.arquitecturajava.com/spring-jdbctemplate-y-el-principio-dry>



Reportes con Jasper y despliegue con Azure

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno despliega una aplicación web que implementa reportes.

TEMARIO

3.1 Tema 5 : JasperReports

- 3.1.1 : Diseño e implementación de reportes con Jasper Studio
- 3.1.2 : Integración JasperReports
- 3.1.3 : Reportes gráficos y resúmenes

3.2 Tema 6 : Azure

- 3.2.1 : Creación de cuenta educativa en Azure
- 3.2.2 : Configuración de entorno Azure
- 3.2.3 : Despliegue de aplicaciones web en Azure

ACTIVIDADES PROPUESTAS

- Los alumnos diseñan reportes usando la herramienta Jaspersoft Studio.
- Los alumnos integran en su aplicativo web la opción de generar reportes usando JasperReports.
- Los alumnos despliegan sus aplicaciones usando Azure.

3.1. JASPERREPORTS

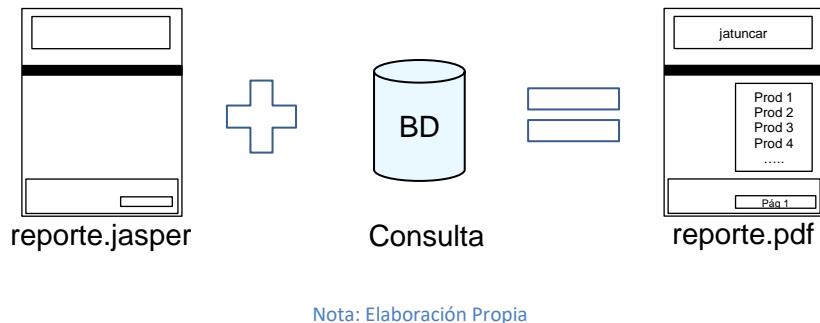
“JasperReports es una biblioteca de procesamiento de contenido, no una aplicación independiente. No puede ejecutarse por sí solo y debe integrarse en otra aplicación Java del lado del cliente o del servidor. JasperReports es una librería pura de Java y se puede usar en cualquier plataforma que admita Java. Al ser una biblioteca, JasperReports es completamente independiente del entorno en el que se utiliza para generar informes.” (Teodor, 2007)

Jasper Reports es un lenguaje para generación de reportes en Java, trabaja en forma similar a un compilador y a un intérprete.

Este archivo fuente XML debe ser compilado para obtener un reporte real. La versión compilada de fuente es nombrada con extensión “Jasper”.

Para ejecutar reportes en Java, se necesita considerar la aplicación del archivo compilado (.jasper) que recibirá los datos a visualizar de la aplicación. Dichos datos pueden ser recuperados desde varias fuentes como bases de datos, JavaBeans, archivos XML, etc.

Figura 181: Conversión de Jasper a pdf

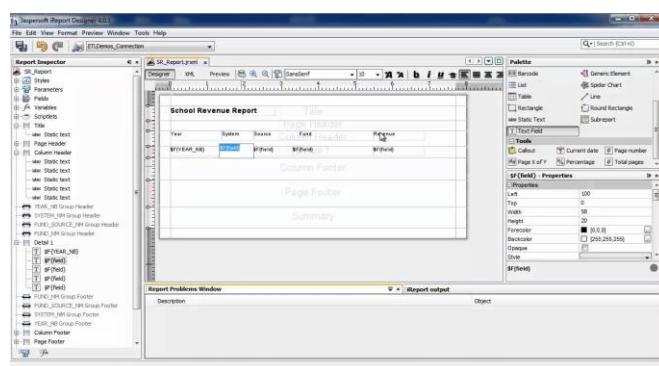


Nota: Elaboración Propia

3.1.1. Diseño e implementación de reportes con Jasper Studio

“La biblioteca JasperReports es una herramienta de generación de informes muy potente y flexible que ofrece contenido Enriquecido en la pantalla, la impresora o el archivo en formato PDF, HTML, RTE, XLS, ODT, CSV o XML.” (Teodor, 2007)

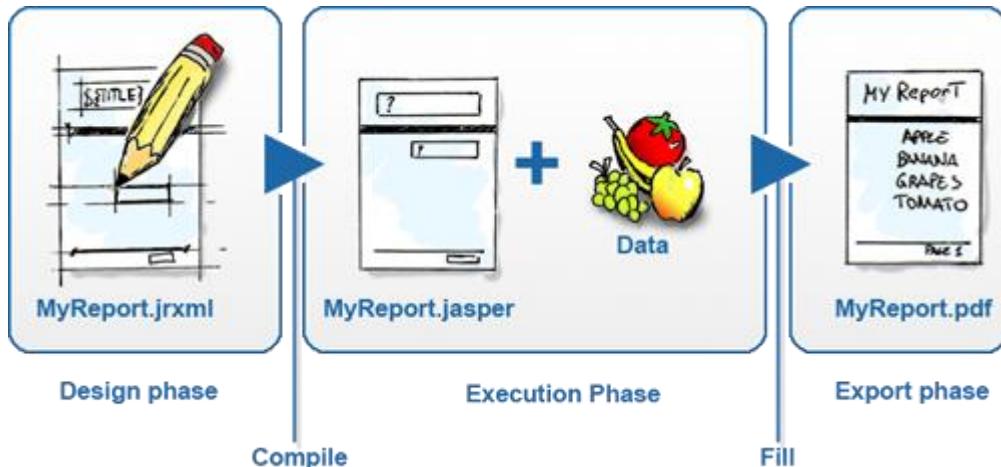
Figura 182: Crear proyecto Jasper



Nota: Elaboración Propia

El software Jaspersoft Studio es un implementador - diseñador de reportes basado en el IDE Eclipse. Es un software mejorado de iReport. Este software permite implementar diseños especializados que contienen gráficos, imágenes, subinformes, tablas cruzadas y mucho más. Se puede integrar con datos mediante varios orígenes de datos.

Figura 183: Proceso de uso de reportes



Nota: Adaptado de tibco

Instalar JasperStudio

Visitar la página <https://community.jaspersoft.com/project/jaspersoft-studio/releases> para descargar (Versión 6.9)

Figura 184: Página oficial de JasperStudio

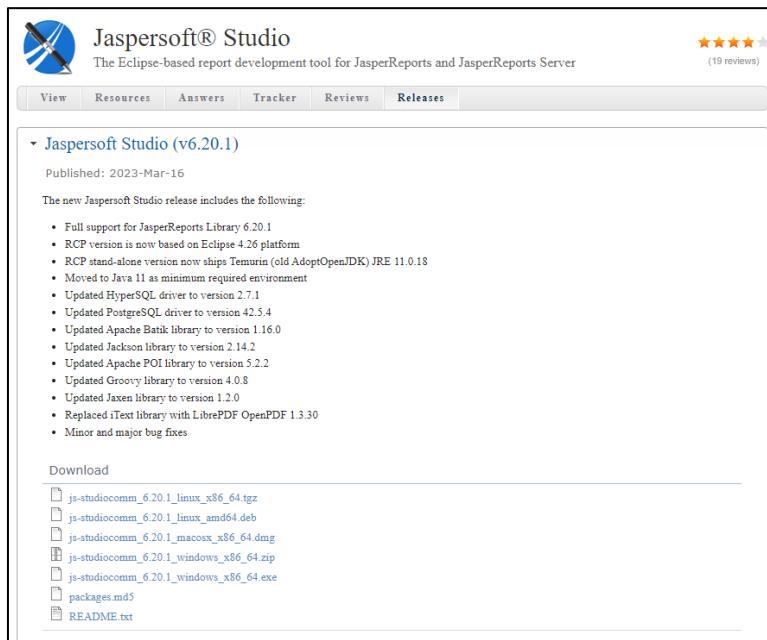
La captura de pantalla muestra la página web de Jaspersoft Community. En la parte superior, hay una barra de menú con enlaces para JASPERSOFT.COM, SUPPORT, CONTACT, SEARCH, LOGIN/REGISTER, PRODUCTS, SOLUTIONS, SERVICES, RESOURCES y un botón verde "Download Now".

En el centro, se muestra la sección "Jaspersoft® Studio" con una descripción: "The Eclipse-based report development tool for JasperReports and JasperReports Server". Una lista desplegable muestra las versiones disponibles: Jaspersoft Studio (v6.20.1), Jaspersoft Studio (v6.20.0), Jaspersoft Studio (v6.19.1), Jaspersoft Studio (v6.19.0), Jaspersoft Studio (v6.18.1), Jaspersoft Studio (v6.17.0), Jaspersoft Studio (v6.16.0), Jaspersoft Studio (v6.15.0), Jaspersoft Studio (v6.14.0) y Jaspersoft Studio (v6.13.0).

A la derecha, hay un cuadro con información sobre la descarga: "Download Jaspersoft® Studio", "Browse Source Code", "Provider: Jaspersoft Corporation", "License: Eclipse", "Issues for Jaspersoft® Studio:", "All Issues: 86 open, 1638 total" y "Bug Reports: 75 open, 1386 total".

Nota: Adaptado de Jaspersoft® Studio, s.f., <https://community.jaspersoft.com/project/jaspersoft-studio/releases>

Figura 185: Página oficial de JasperStudio

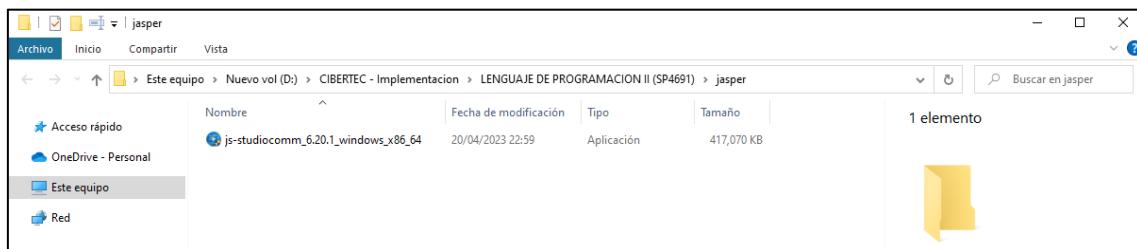


Nota: Adaptado de Jaspersoft® Studio, s.f., <https://community.jaspersoft.com/project/jaspersoft-studio/releases>

Mostrará una página para iniciar sesión o inscribir al sitio de la comunidad de Jaspersoft, para luego poder descargar la versión.

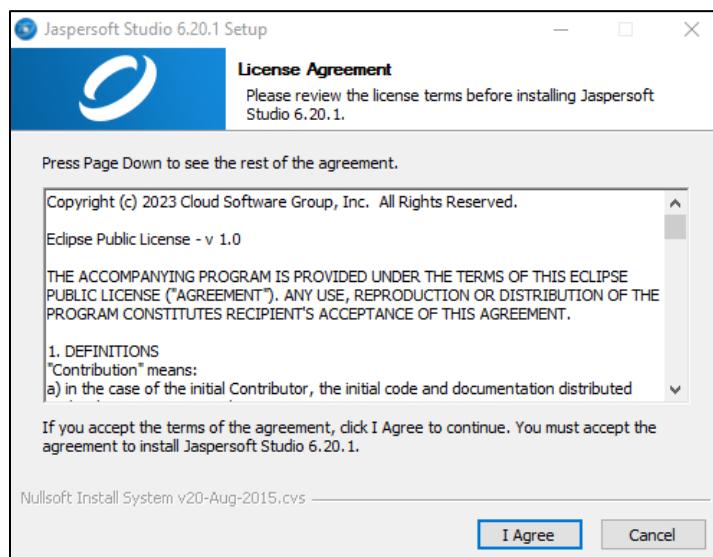
Forma 1: iniciar el instalador y continuar con los pasos de instalación.

Figura 186: Instalador de JasperSoft



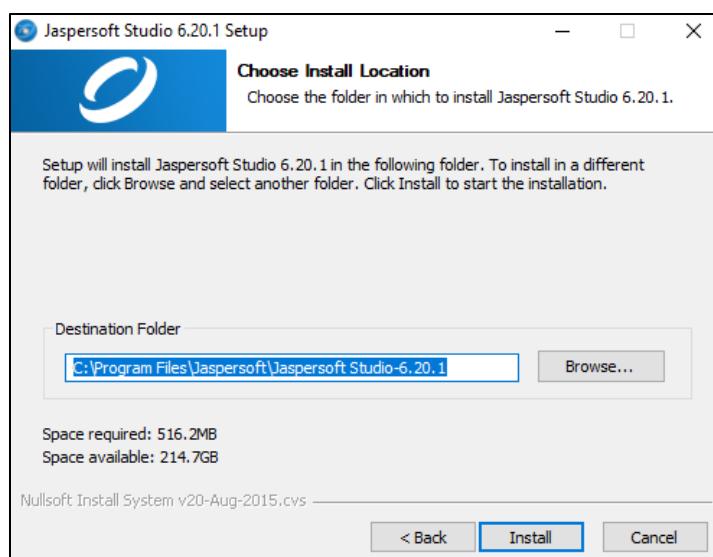
Nota: Elaboración Propia

Figura 187: Proceso instalación



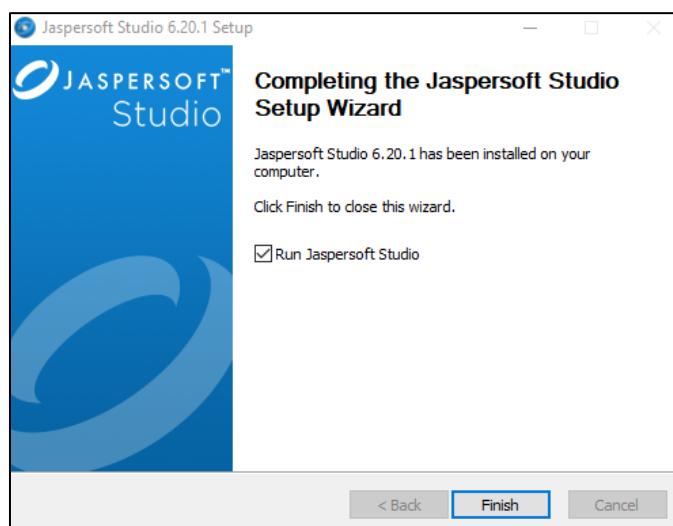
Nota: Elaboración Propia

Figura 188: Proceso instalación



Nota: Elaboración Propia

Figura 189: Terminamos instalación de JasperStudio



Nota: Elaboración Propia

Figura 190: Iniciamos JasperStudio

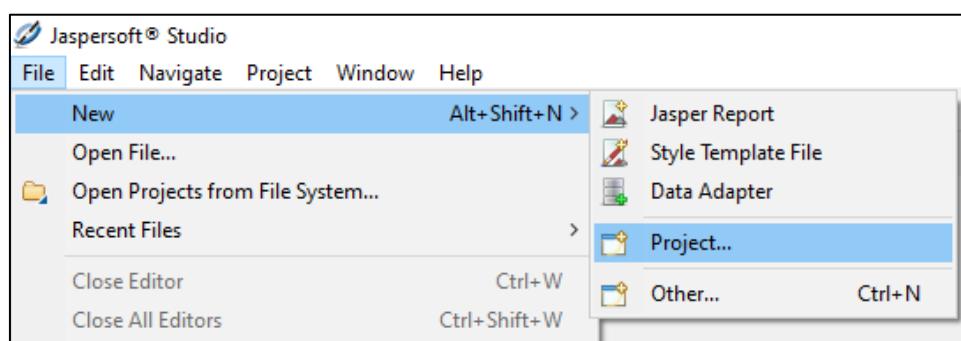


Nota: Elaboración Propia

Creando plantilla de reporte

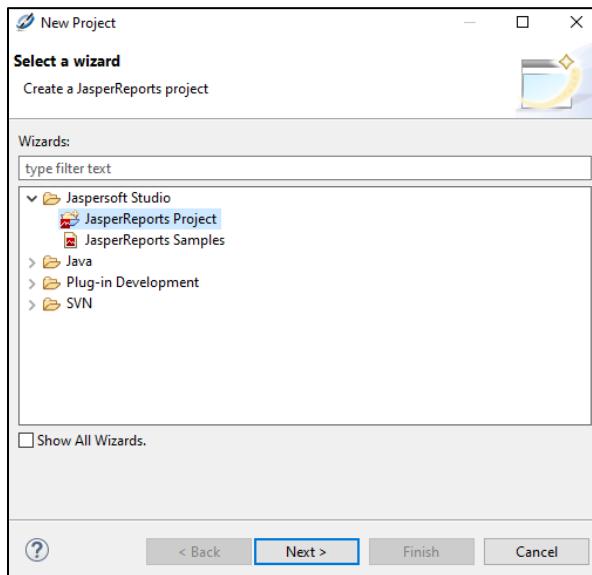
Paso 1. Crear el proyecto.

Figura 191: Creamos proyecto en JasperStudio



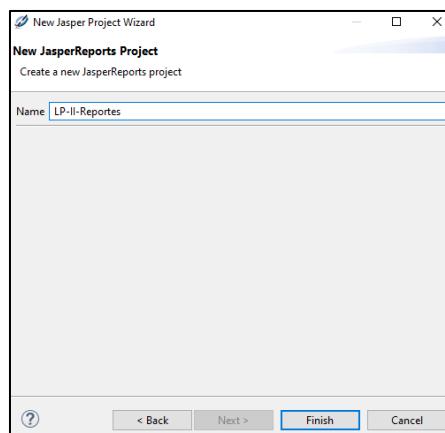
Nota: Elaboración Propia

Figura 192: Creamos proyecto en JasperStudio



Nota: Elaboración Propia

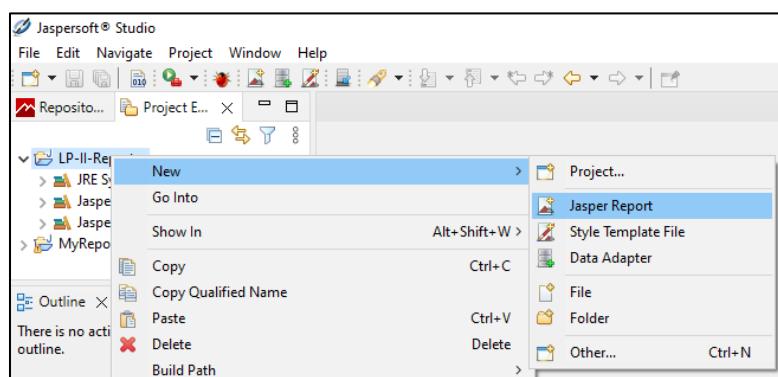
Figura 193: Creando proyecto en JasperStudio



Nota: Elaboración Propia

Paso 2. Crear el reporte.

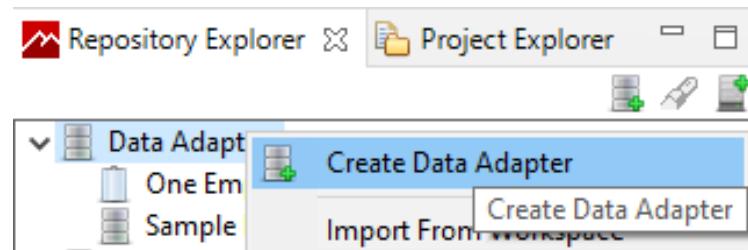
Figura 194: Nuevo reporte



Nota: Elaboración Propia

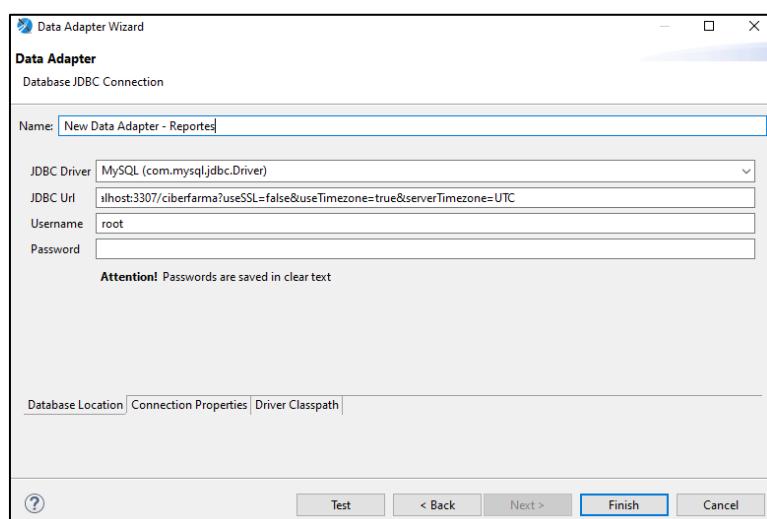
Paso 3. Consulta de datos. Opcionalmente, se puede conectar con la base de datos para generar los campos que servirán al diseñar el reporte.

Figura 195: Nuevo origen de datos



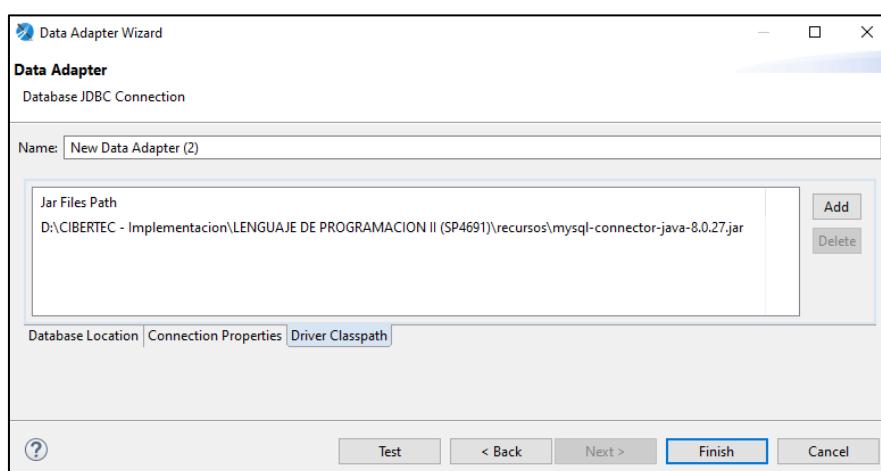
Nota: Elaboración Propia

Figura 196: Nuevo origen de datos



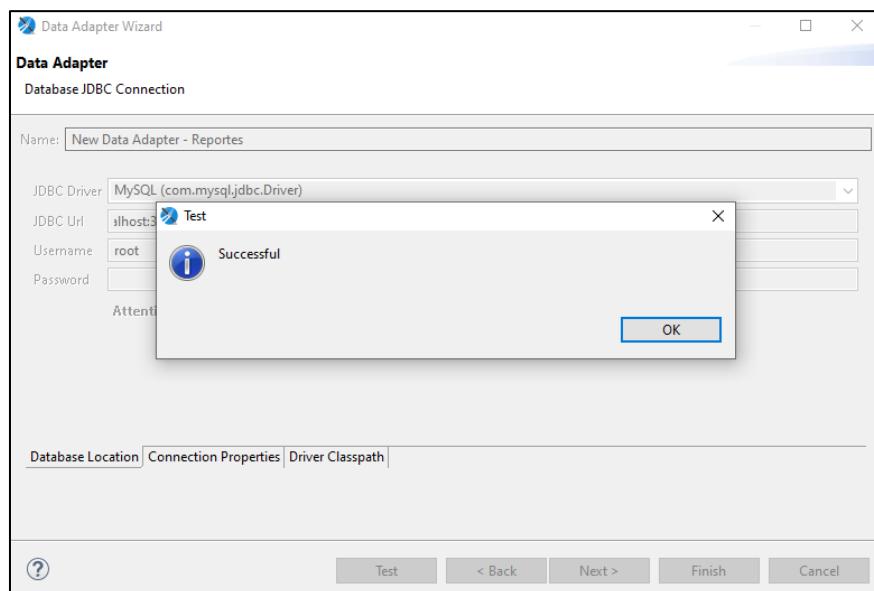
Nota: Elaboración Propia

Figura 197: Nuevo origen de datos



Nota: Elaboración Propia

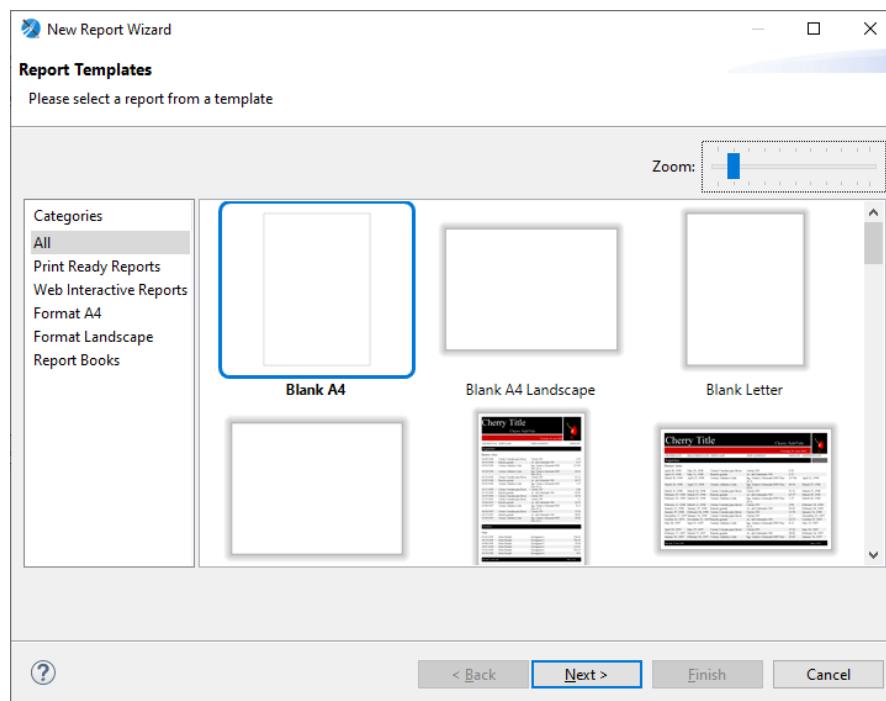
Figura 198: Validación de Conexión



Nota: Elaboración Propia

Paso 4. Definiendo el tipo de plantilla.

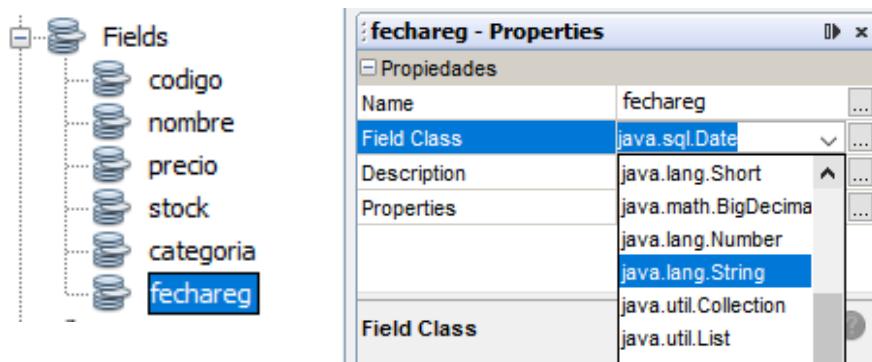
Figura 199: Seleccionando plantilla



Nota: Elaboración Propia

Paso 5. JasperStudio, crea los campos considerando los valores de las tablas, por lo que, si es necesario, habrá que cambiar sus propiedades:

Figura 200: Campos y propiedades



Nota: Elaboración Propia

Paso 6. Arrastrar los campos a la sección respectiva, como el detalle.

Figura 201: Ejemplo de detalle

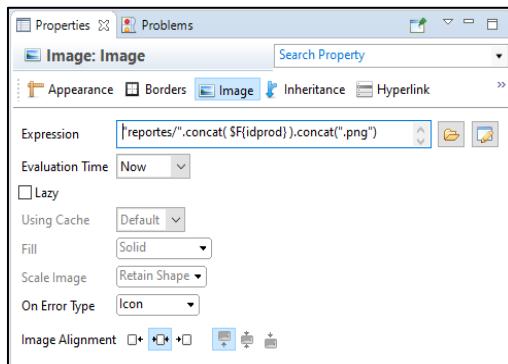
	Código	Nombre del producto	Stock disponible	Categoría	Precio Unitario
	\$F{idprod}	\$F{descripción}	\$F{stock}	\$F{idcategoria}	\$F{precio}

Detail 1

Nota: Elaboración Propia

Si se necesitan emplear elementos como imágenes que cambien, de acuerdo con los registros, se puede emplear **expresiones**. En este ejemplo, las imágenes se ubicarán en la carpeta **reportes** y en las propiedades, se activará la opción de mostrar ícono en caso de error.

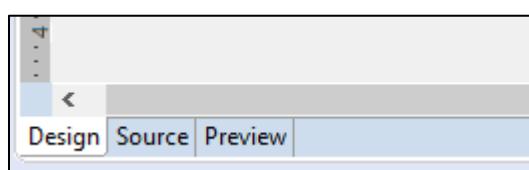
Figura 202: Propiedades de la imagen



Nota: Elaboración Propia

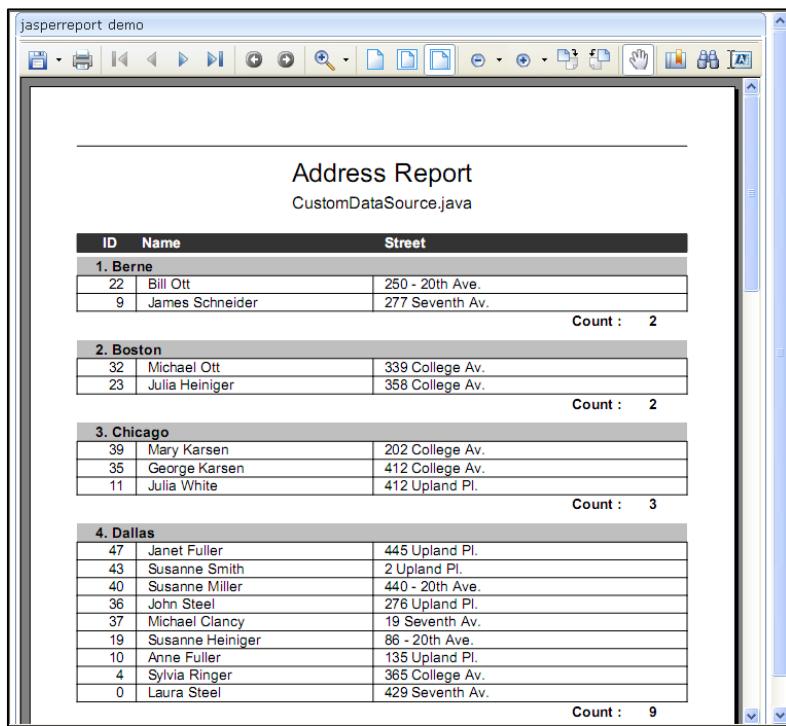
Paso 6. Se puede cambiar la visualización del diseño usando las pestañas:

Figura 203: Pestañas o vistas



Nota: Elaboración Propia

Figura 204: Ejemplo de las vistas



Nota: Elaboración Propia

3.1.2. Integración JasperReports

Paso 1. Agregar las dependencias a nuestro proyecto en el archivo **pom.xml**, para ello buscar en sitios como MVN Repository.

Figura 205: Página del repositorio

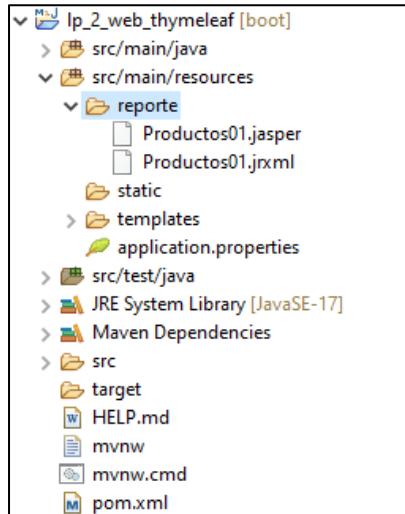
Version	Vulnerabilities	Repository	Usages	Date
6.20.2		Central	6	Apr 19, 2023
6.20.1		Central	8	Mar 14, 2023
6.20.0		Central	20	Jul 14, 2022
6.19.1		Central	11	Mar 29, 2022
6.19.0		Central	11	Feb 22, 2022
6.18.1		Central	15	Oct 29, 2021
6.18.0		Central	5	Oct 25, 2021
6.17.0		Central	25	May 11, 2021
6.16.0		Central	17	Nov 06, 2020
6.15.0		Central	10	Oct 06, 2020
6.14.0		Central	10	Jul 31, 2020

Nota: Adaptado de JasperReports Library, s.f., <https://mvnrepository.com/artifact/net.sf.jasperreports/jasperreports>

```
<!-- https://mvnrepository.com/artifact/net.sf.jasperreports/jasperreports -->
<dependency>
    <groupId>net.sf.jasperreports</groupId>
    <artifactId>jasperreports</artifactId>
    <version>6.9.0</version>
</dependency>
```

Paso 3. En los recursos del proyecto, crear una carpeta para colocar los **reportes** diseñados con JasperStudio (.jasper y .jrxml):

Figura 206: Estructura del proyecto con la carpeta de reportes



Nota: Elaboración Propia

Paso 4. Escribir el código para generar el PDF.

Ejemplo:

```
 @RequestMapping(value = "/ProductosReport", method = RequestMethod.GET)
 @ResponseBody
 public void ProductsReport(HttpServletRequest response) throws JRException, IOException, SQLException {

    Connection conn = jdbcTemplate.getDataSource().getConnection();
    InputStream jasperStream = this.getClass().getResourceAsStream("/report/Productos01.jasper");
    Map<String, Object> params = new HashMap<String, Object>();
    JasperReport jasperReport = (JasperReport) JRLoader.loadObject(jasperStream);
    JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, params, conn);
    response.setContentType("application/x-pdf");
    response.setHeader("Content-disposition", "inline; filename=productos_report.pdf");
    final OutputStream outputStream = response.getOutputStream();
    JasperExportManager.exportReportToPdfStream(jasperPrint, outputStream);
    conn.close();
}
```

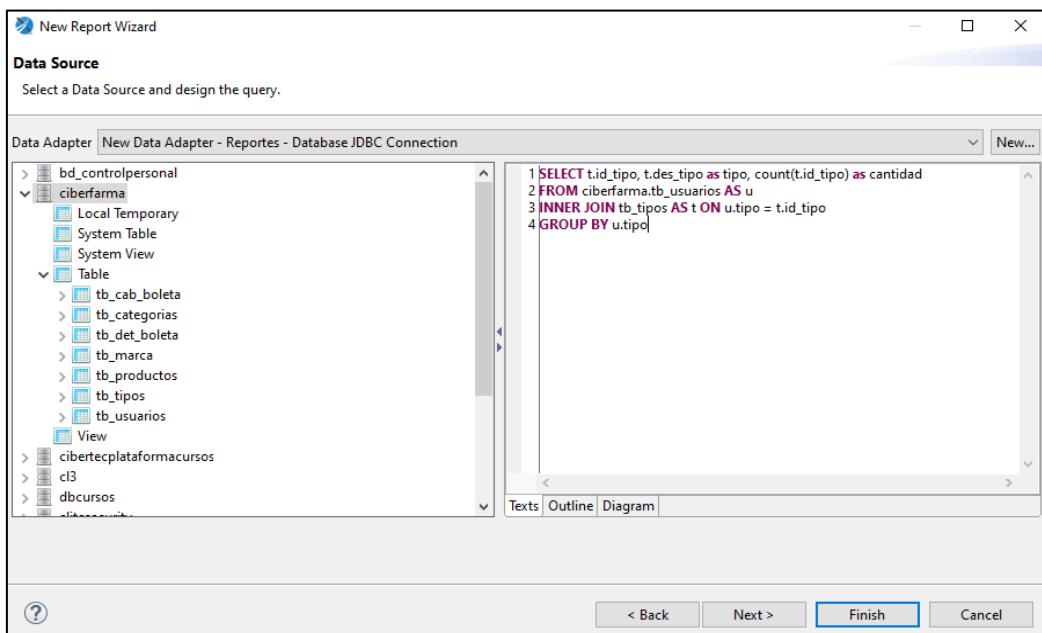
3.1.3. Reportes gráficos y resúmenes

En esta parte para a crear un reporte que utilice como elemento visual un gráfico de tipo Chart. Estos reportes es posible mostrar los datos desde un conjunto de datos principal. Esto facilita la inclusión de diversos gráficos en un reporte.

Seguiremos los siguientes pasos para construir ese reporte:

Paso 1. Creación del reporte y elaborando la sentencia

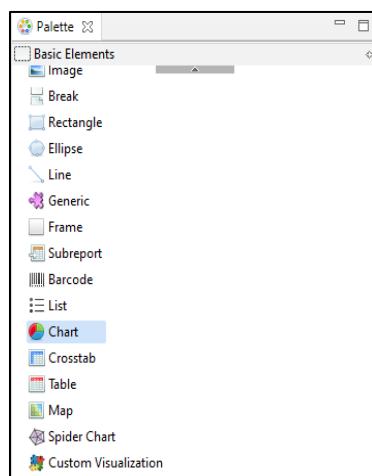
Figura 207: Ejemplo de consulta para gráfico



Nota: Elaboración Propia

Paso 2. Seleccionar de la paleta de herramientas, la opción **Chart** y arrastrar de preferencia a la sección **resumen (Summary)**, o siempre que sólo se necesite un gráfico.

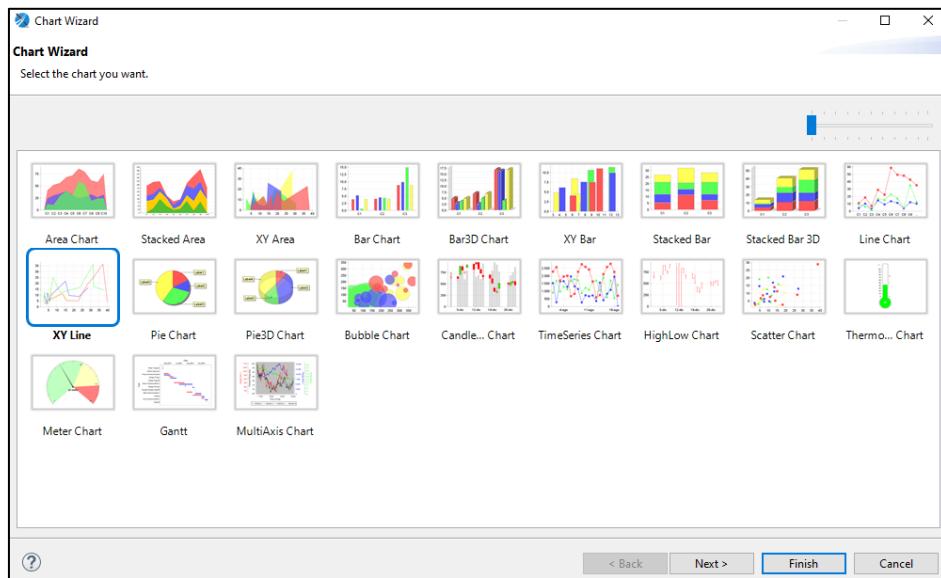
Figura 208: Herramienta Chart



Nota: Elaboración Propia

Paso 3. Seleccionar el tipo de gráfico: XY Line.

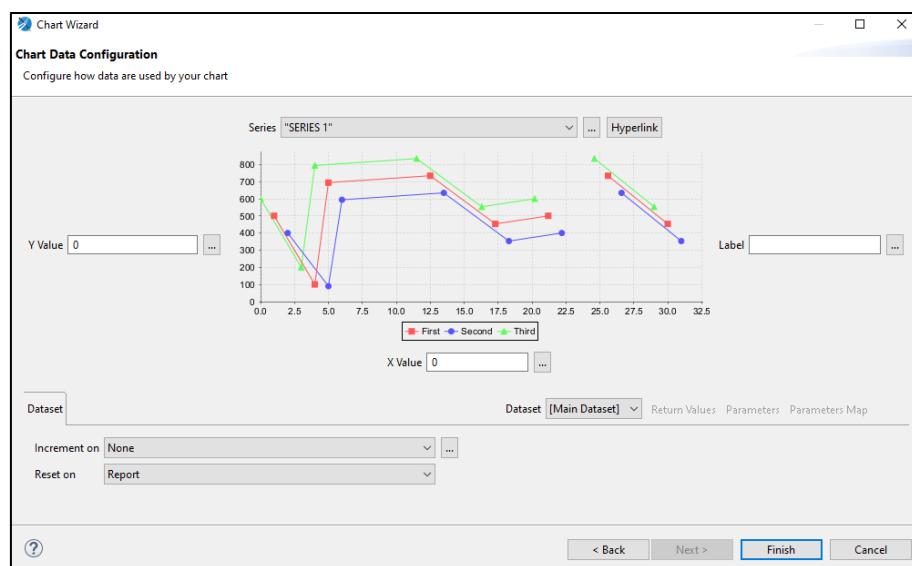
Figura 209: Asistente gráfico



Nota: Elaboración Propia

Paso 4. Dependiendo del tipo de gráfico, seleccionar las opciones para cada “serie” o grupo de valores.

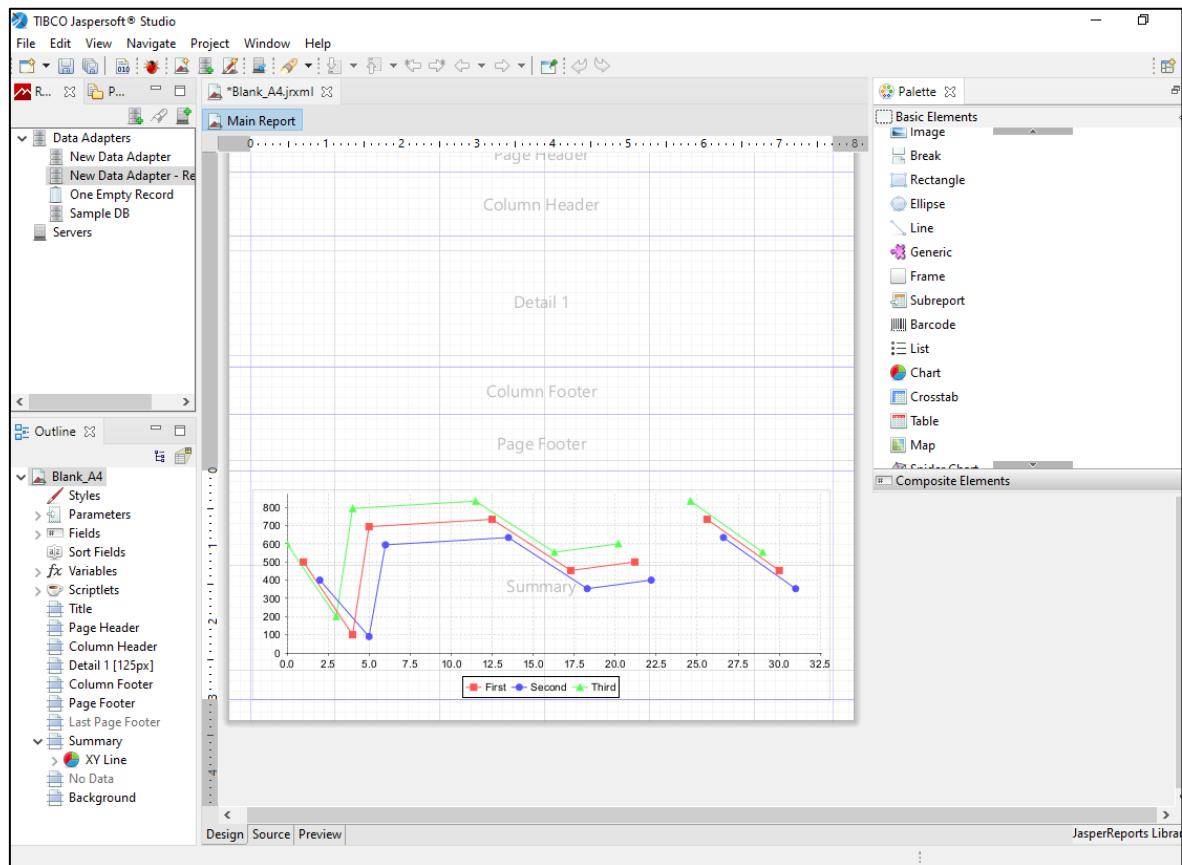
Figura 210: Asistente gráfico



Nota: Elaboración Propia

Paso 5. Finalmente, quedará lo siguiente:

Figura 211: Asistente gráfico



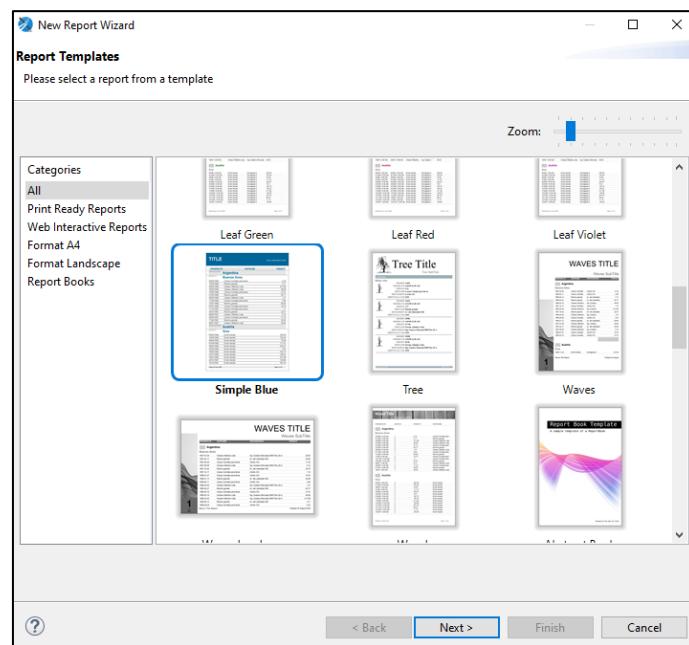
Nota: Elaboración Propia

Reportes de resumen

Estos permiten generar gráficos a manera de cursores de SQL, es decir, por cada elemento o coincidencia, se muestran los valores agrupando los resultados hasta el siguiente grupo de valores.

Paso 1. Realizar el diseño de reporte.

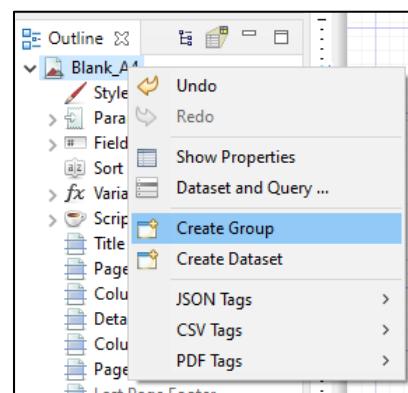
Figura 212: Creamos el reporte y seleccionamos su diseño



Nota: Elaboración Propia

Paso 2. Crear los campos de agrupamiento.

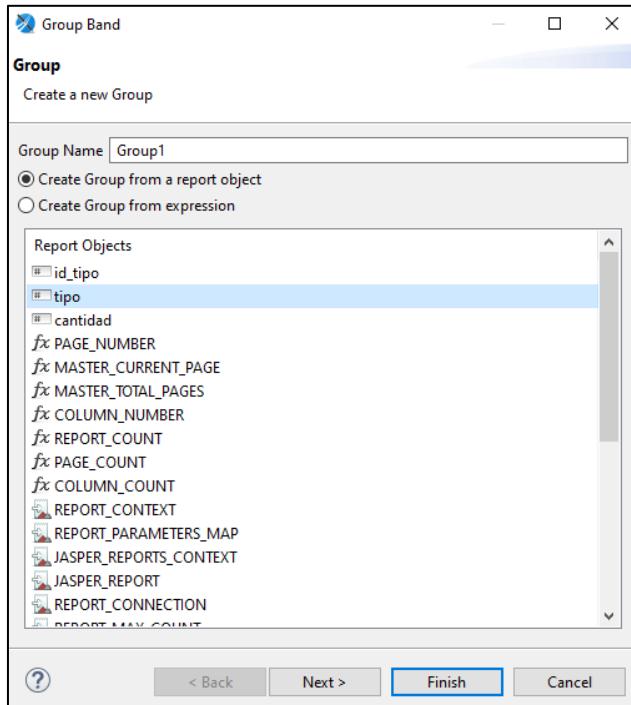
Figura 213: Creando los agrupamientos



Nota: Elaboración Propia

Paso 3. Seleccionar el campo de agrupamiento y escribir un nombre identificador:

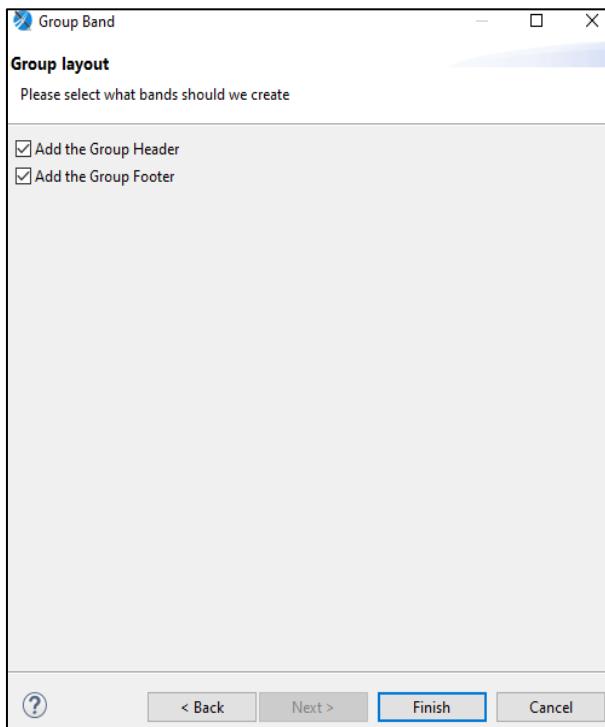
Figura 214: Creando los agrupamientos



Nota: Elaboración Propia

Paso 4. Agregar las secciones de cabecera y pie de grupo:

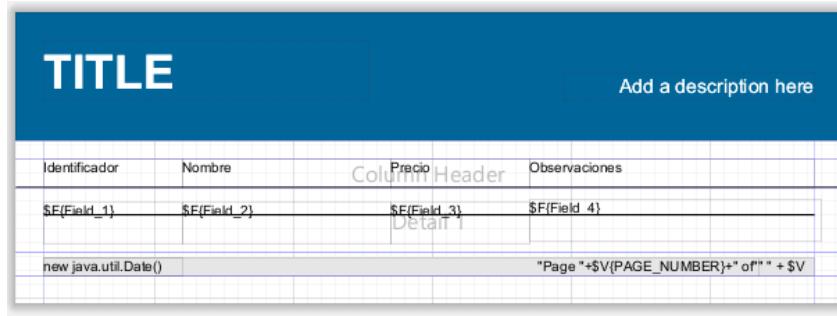
Figura 215: Creando los agrupamientos



Nota: Elaboración Propia

Paso 5. Aparecerán las secciones, donde se colocarán los elementos respectivos de agrupamiento y en el detalle, los campos en común.

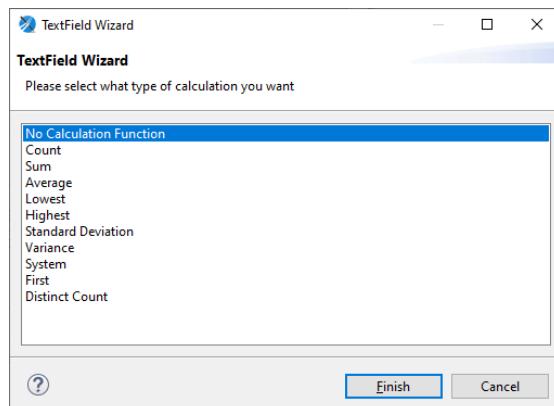
Figura 216: Creando los agrupamientos



Nota: Elaboración Propia

También se pueden utilizar estas secciones como elementos de estadística:

Figura 217: Creando los agrupamientos



Nota: Elaboración Propia

Resumen

1. Jaspersoft Studio, es una herramienta que nos permite crear diseños para un reporte simplificando la creación del archivo XML y generando el archivo .Jasper.
2. Tiene dependencias, que permiten una integración sencilla con eclipse o spring, facilitando la implementación de reportes Jasper en nuestros proyectos.
3. Existen diversos tipos de gráficos que se emplean en un reporte; sin embargo, no todos se pueden diseñar de la misma manera la cantidad de grupos de valores dependerán del tipo de gráfico.

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://community.jaspersoft.com/project/jaspersoft-studio>
- <https://community.jaspersoft.com/project/jaspersoft-studio/releases>
- <https://mvnrepository.com/artifact/net.sf.jasperreports/jasperreports>

3.2. AZURE

Azure es una plataforma de nube pública de Microsoft que nos permite compilar, implementar y administrar de una manera rápida aplicaciones en los centros de datos de Microsoft.

“Microsoft brinda soporte para nubes públicas, privadas e híbridas. Windows Azure Pack es un complemento gratuito de Microsoft System Center que le permite alojar muchos de los servicios principales de Azure en su propio centro de datos y brindar una experiencia de portal de autoservicio a sus usuarios. Puede integrarlos en una nube híbrida mediante el uso de una nube virtual” (Robin, 2015)

Figura 218: Portal de Microsoft Azure en español



Nota: Adaptado de Azure, 2023, <https://azure.microsoft.com/es-es>

3.2.1. Creación de cuenta educativa en Azure

En Microsoft Azure todos los estudiantes de Cibertec tienen acceso a los diferentes recursos en la nube usando sus credenciales institucionales. Como estudiante te deberás registrar en el portal de Microsoft Azure for Student y verificar tu identidad.

Figura 219: Portal de Azure Education



Nota: Adaptado de Azure, s.f., <https://azure.microsoft.com/es-es/resources/students>

Procedimiento:

- Ingresá al portal de Azure (<https://azure.microsoft.com/es-es/resources/students>). En la parte superior ingresa presionando “Cuenta gratuita”.

Figura 220: Portal de Azure Education

The screenshot shows the Azure Education landing page. At the top, there's a navigation bar with links like 'Explorar', 'Productos', 'Soluciones', 'Precios', 'Asociados', 'Recursos', 'Búsqueda', 'Aprender', 'Soporte', 'Hable con ventas', a green 'Cuenta gratuita' button, and 'Iniciar sesión'. Below the navigation, a large black header box contains the title 'Recursos para estudiantes de desarrollo' and a subtext: 'Adquiere aptitudes que impulsen tu carrera y cause un impacto positivo en todo el mundo.' A green 'Empezar gratis' button is visible. The main content area has tabs at the top: 'Información general', 'Aprender', 'Crear', 'Participar', and 'Oportunidades'. One section is titled 'Impulsar una carrera tecnológica' with a subtext about finding resources for programming languages and job placement. Another section at the bottom is titled 'Nota: Adaptado de Azure, s.f., https://azure.microsoft.com/es-es/resources/students'.

- En el portal, presiona “Empiece gratis”. El usar esta herramienta tendrás un crédito de \$100. No se pedirá tarjeta de crédito y se tendrá acceso a las diferentes herramientas de Azure.

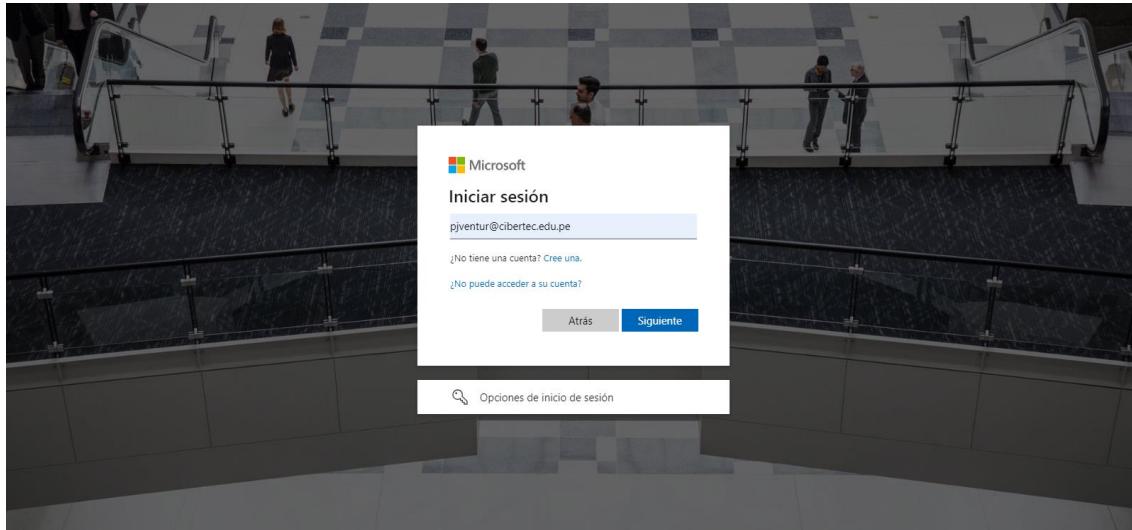
Figura 221: Portal de Azure Education

The screenshot shows the Azure Education landing page with a dark header. It features a central section titled 'Compilación gratuita en la nube con Azure for Students' with a subtext: 'Use el correo electrónico de su universidad o escuela para registrarse y renovar cada año que sea estudiante'. Below this are two buttons: 'Empiece gratis' and 'Más información acerca de la idoneidad'. At the bottom, there are two boxes: one on the left saying 'Comience con \$100 crédito de Azure' and one on the right saying 'No se requiere tarjeta de crédito'.

Nota: Adaptado de Azure, s.f., <https://azure.microsoft.com/es-es/resources/students>

- Ingresa usando correo institucional Cibertec y contraseña del portal Office365:

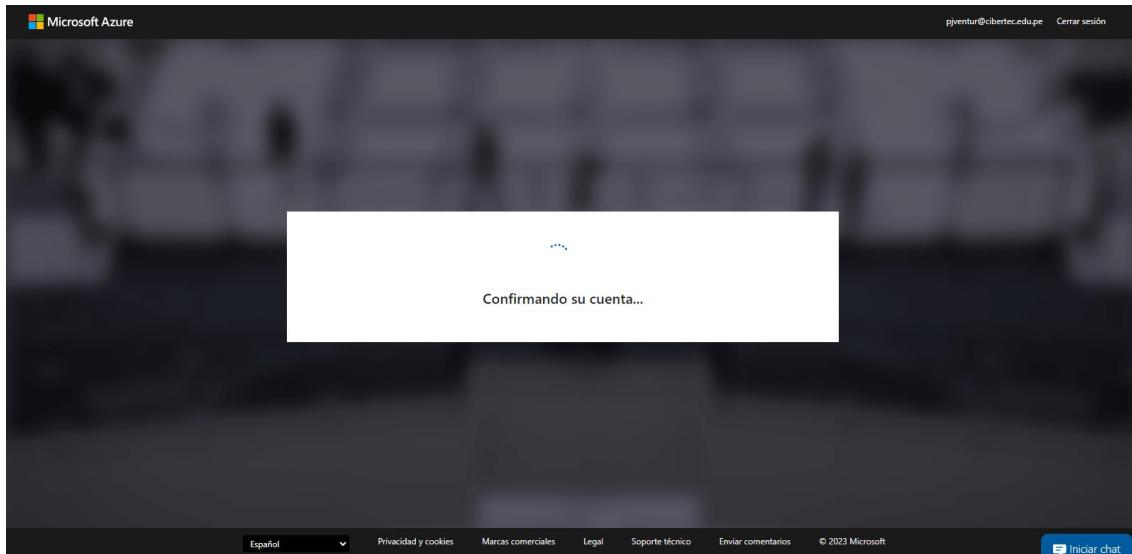
Figura 222: Portal de Azure Education



Nota: Adaptado de Azure, s.f., <https://azure.microsoft.com/es-es/resources/students>

- Se verificará la identidad:

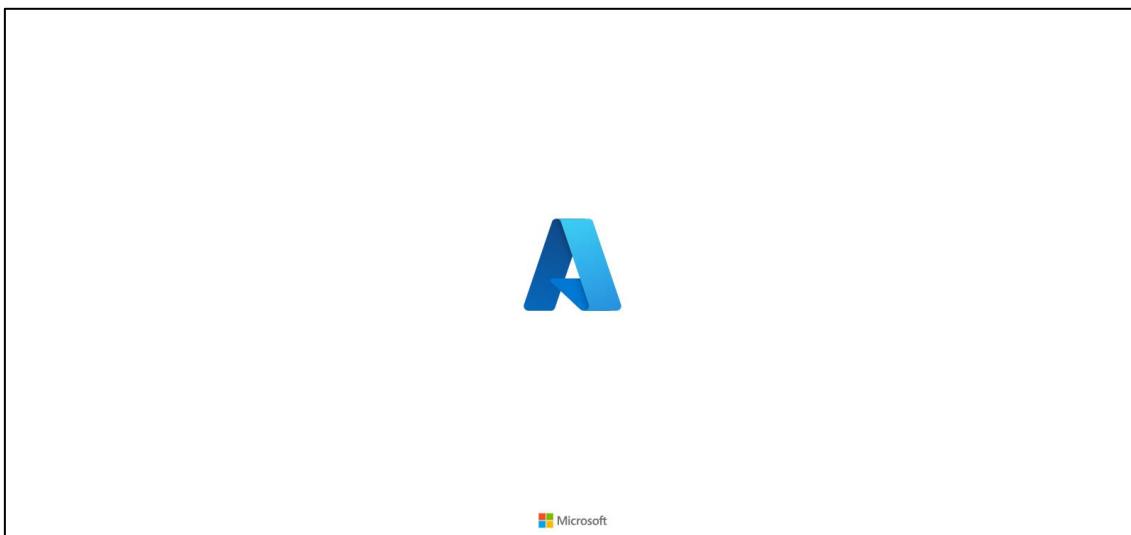
Figura 223: Portal de Azure Education



Nota: Adaptado de Azure, s.f., <https://azure.microsoft.com/es-es/resources/students>

- Esperamos que el portal de Azure inicie:

Figura 224: Portal de Azure Education



Nota: Adaptado de Azure, s.f., <https://azure.microsoft.com/es-es/resources/students>

- Pantalla de información general de la cuenta. Podremos visualizar el crédito que Azure nos da como cuenta educativa:

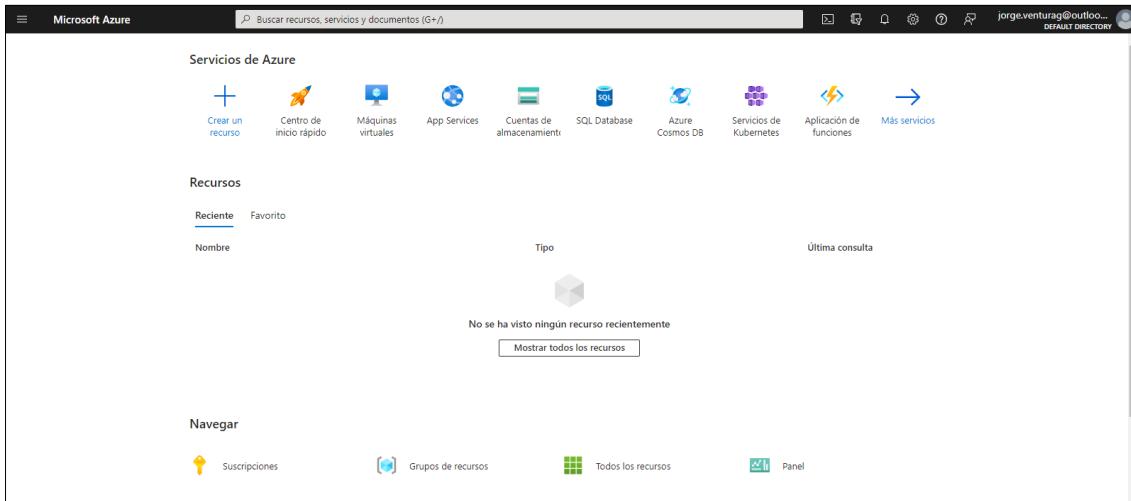
Figura 225: Portal de Azure Education

Nota: Adaptado de Azure, s.f., <https://azure.microsoft.com/es-es/resources/students>

3.2.2. Configuración de entorno Azure

Pasos 1. Abrimos el panel de Azure y seleccionamos “Crear un recurso”:

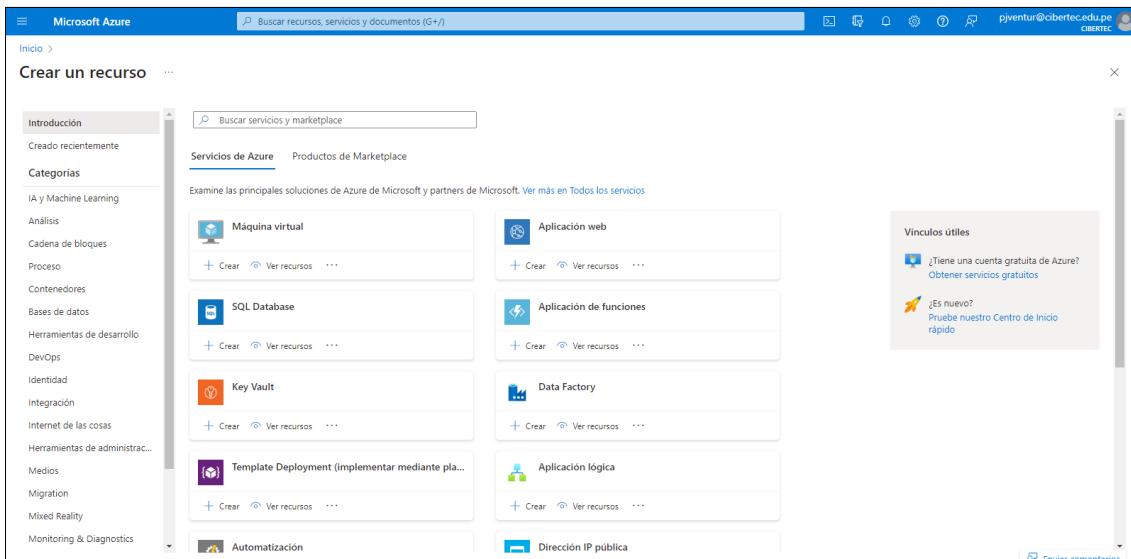
Figura 226: Portal de Azure



Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

Paso 2. Ahora, buscamos el recurso “Aplicación Web”, y presionamos la opción “Crear”:

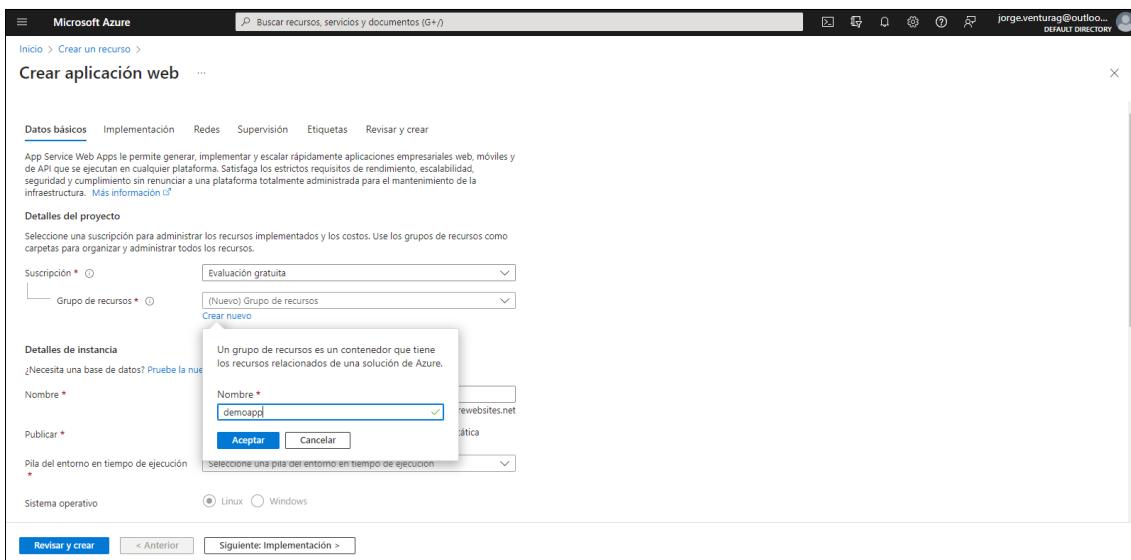
Figura 227: Creación de recurso web



Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

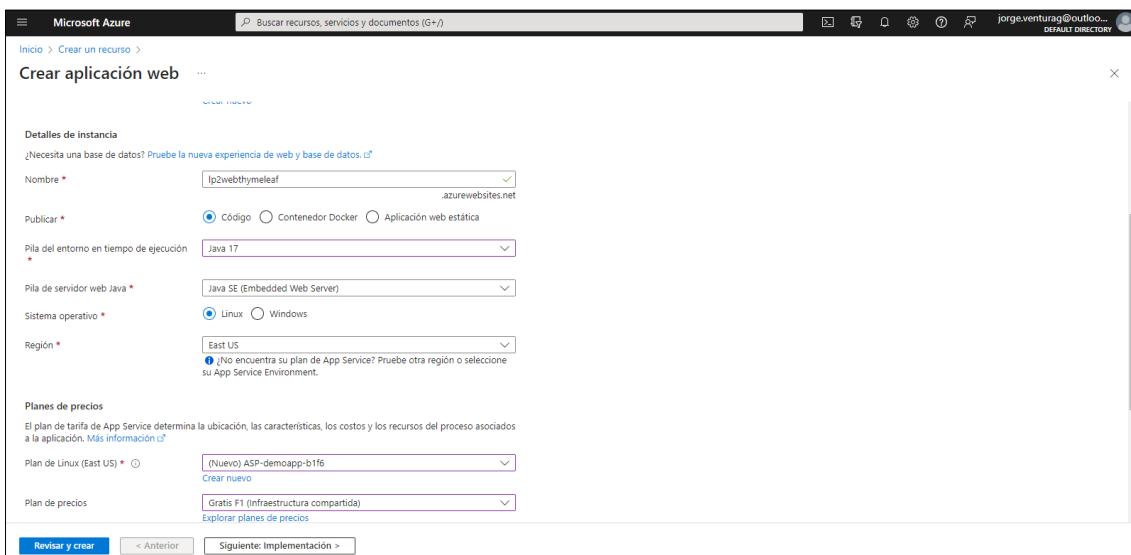
Paso 3. Completamos los datos solicitados para crear la aplicación web y presionamos “Revisar y crear”:

Figura 228: Creación de recurso web



Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

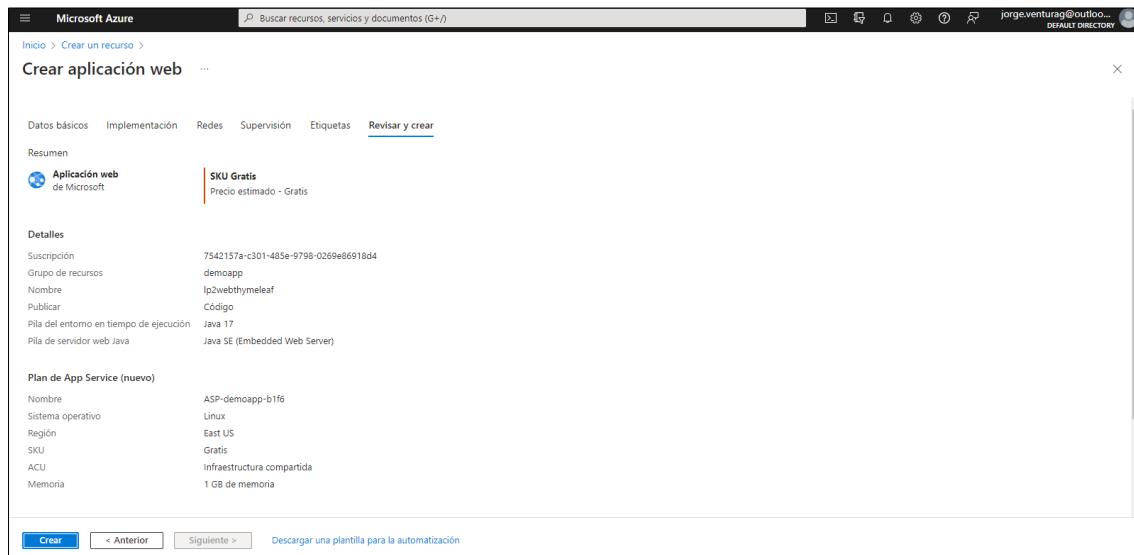
Figura 229: Creación de recurso web



Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

Paso 4. Para finalizar la creación del recurso, presionamos “Crear”:

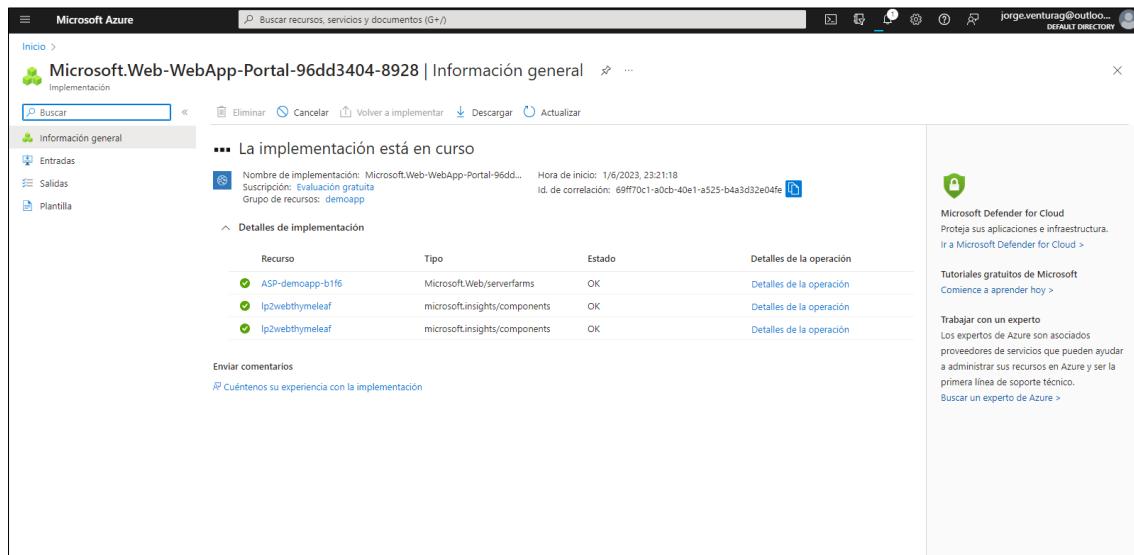
Figura 230: Creación de recurso web



Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

Paso 5. Esperamos que el aplicativo web se termine de crear:

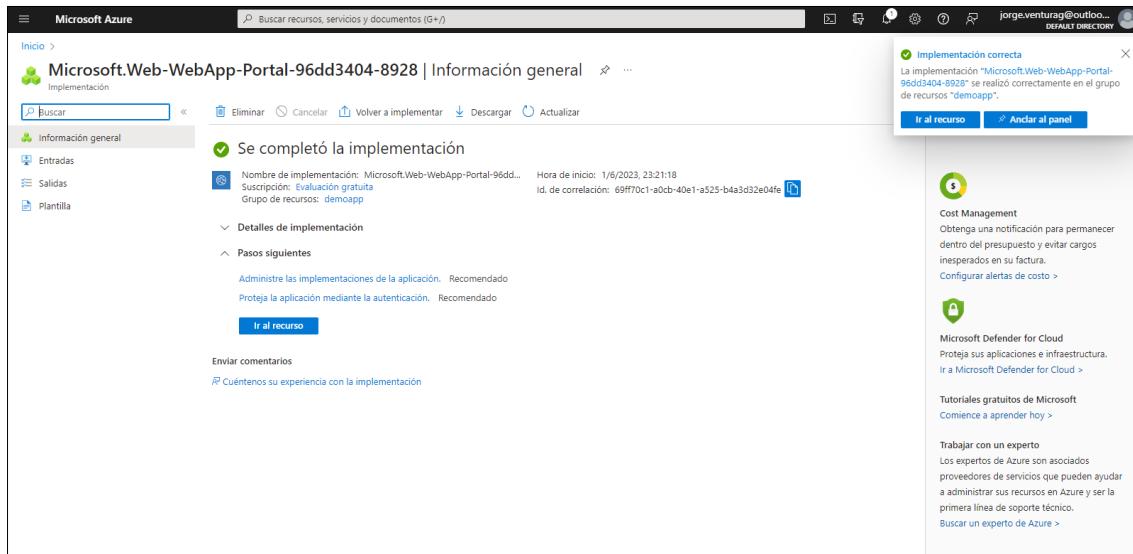
Figura 231: Creación de recurso web



Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

Paso 6. Confirmación de aplicativo web creado y presionamos el botón inferior “Ir al recurso”:

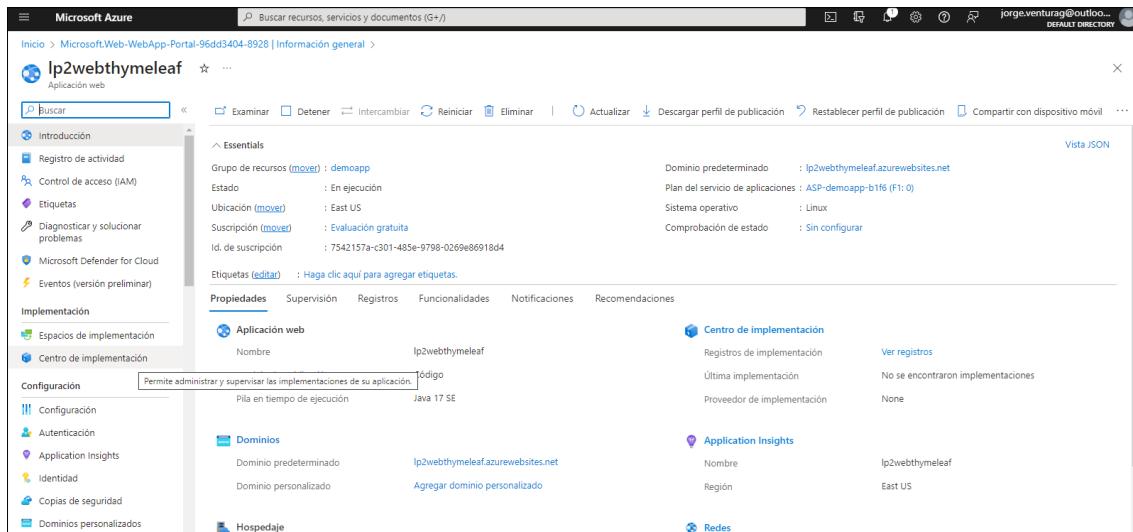
Figura 232: Creación de recurso web



Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

Paso 6. Confirmación de aplicativo web creado:

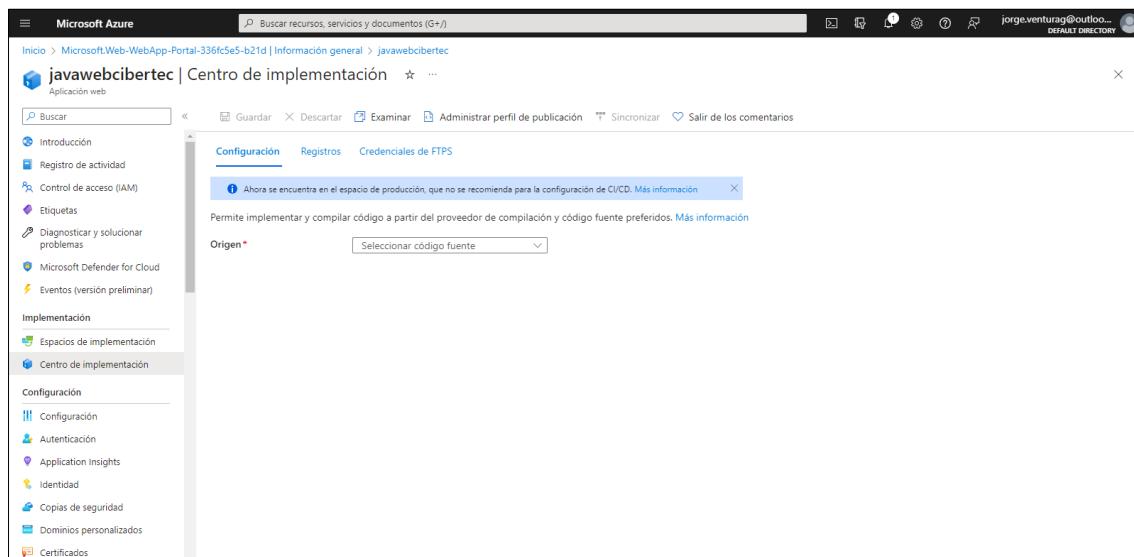
Figura 233: Recursos creado



Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

Paso 7. Seguimos en la pantalla anterior. Ingresemos a la opción “Centro de implementación” y esperamos que termine de cargar la información.

Figura 234: Recursos creado



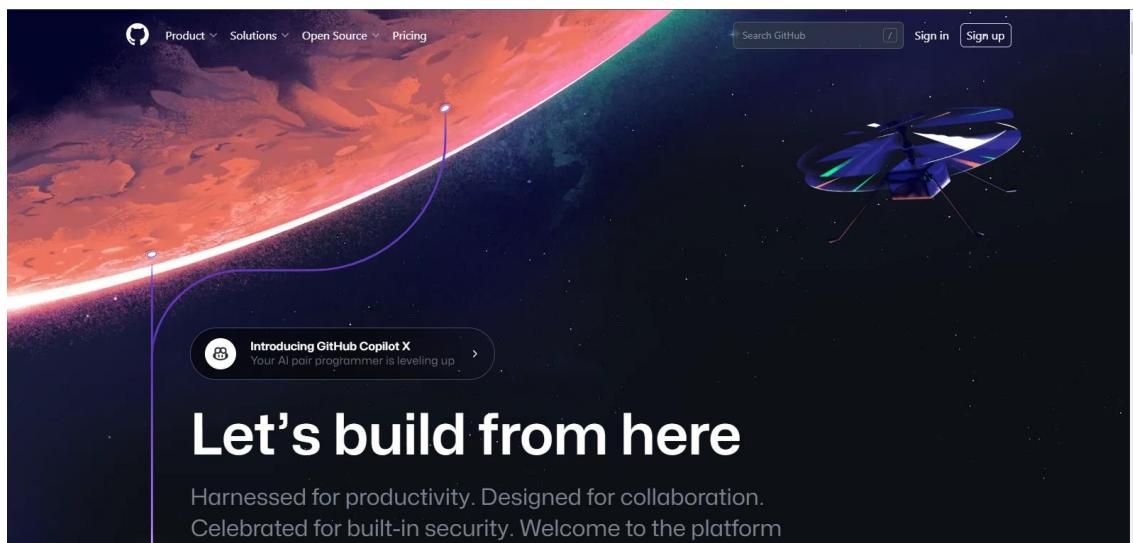
Nota: Adaptado de Azure, 2023, <https://portal.azure.com/#home>

Configuración de GitHub:

Paso 1. El proyecto creado en eclipse se deberá subir a un repositorio de GitHub que nos permita poder escogerlo desde Azure.

Vamos a GitHub e ingresen sus credenciales:

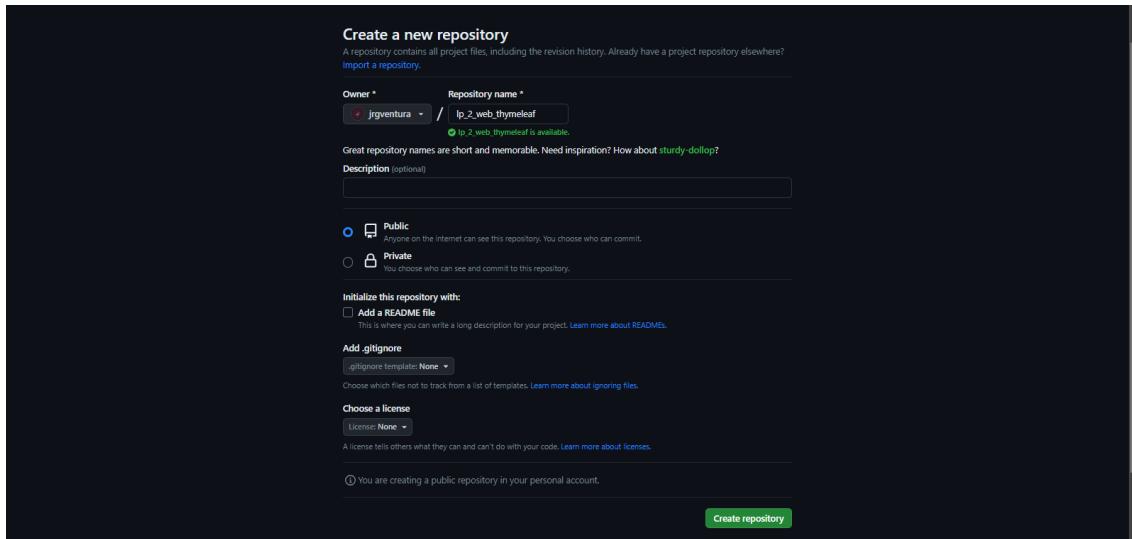
Figura 235: Portada de GitHub



Nota: Adaptado de Github, s.f., <https://github.com/>

Paso 2. Creamos un nuevo repositorio en GitHub. Prioricemos la configuración indicada en la imagen:

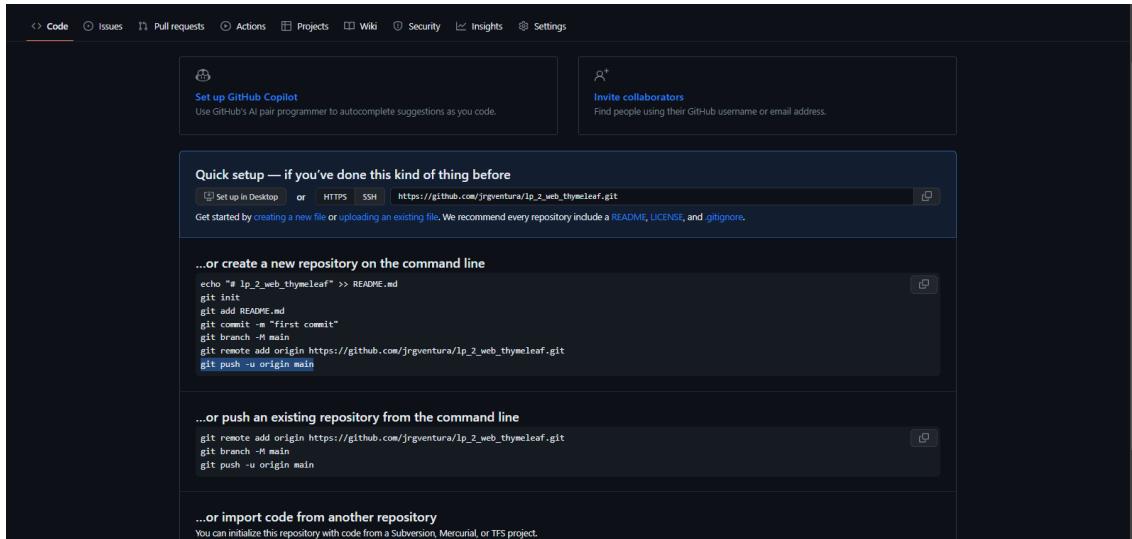
Figura 236: Portada de GitHub



Nota: Adaptado de Github, s.f., <https://github.com/>

Paso 3. Listo. Con el repositorio creado, subimos los cambios del proyecto local siguiendo la secuencia de comandos indicados por GitHub:

Figura 237: Portada de GitHub



Nota: Adaptado de Github, s.f., <https://github.com/>

Paso 4. Siguiendo los pasos que nos indica GitHub, subimos nuestro proyecto al repositorio creado. Para esto debemos abrir la terminal en la ruta del proyecto y seguir la secuencia de comandos:

- `git init`

Figura 238: Terminal Eclipse

```
C:\WINDOWS\system32\cmd.exe X
Microsoft Windows [Versión 10.0.19045.2965]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pc\Documents\lp-2\lp_2_web_thymeleaf>git init
Initialized empty Git repository in C:/Users/pc/Documents/lp-2/lp_2_web_thymeleaf/.git/
C:\Users\pc\Documents\lp-2\lp_2_web_thymeleaf>
```

Nota: Elaboración propia

Agregamos los archivos necesarios a subir al repositorio:

- `git add .\pom.xml`
- `git add .\src\`

Figura 239: Terminal Eclipse

```
C:\WINDOWS\system32\cmd.exe X
Microsoft Windows [Versión 10.0.19045.2965]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pc\Documents\lp-2\lp_2_web_thymeleaf>git init
Initialized empty Git repository in C:/Users/pc/Documents/lp-2/lp_2_web_thymeleaf/.git/
C:\Users\pc\Documents\lp-2\lp_2_web_thymeleaf>
```

Nota: Elaboración propia

Subimos los cambios al repositorio:

- `git commit -m "first commit"`
- `git push origin main`

Figura 240: Terminal Eclipse

```
C:\WINDOWS\system32\cmd.exe X
Microsoft Windows [Versión 10.0.19045.2965]
(c) Microsoft Corporation. Todos los derechos reservados.

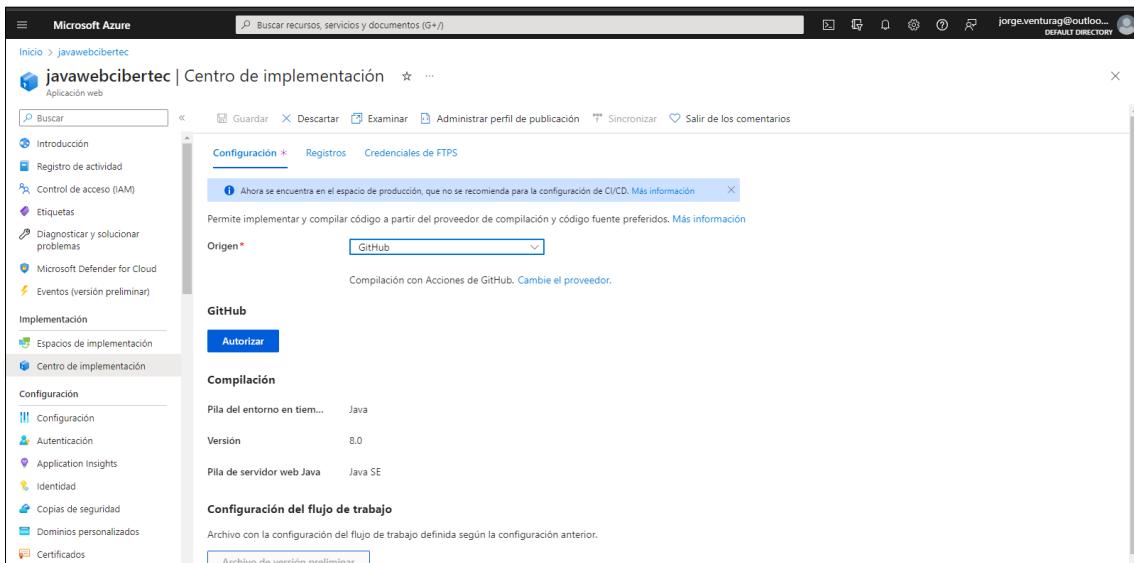
C:\Users\pc\Documents\lp-2\lp_2_web_thymeleaf>git init
Initialized empty Git repository in C:/Users/pc/Documents/lp-2/lp_2_web_thymeleaf/.git/
C:\Users\pc\Documents\lp-2\lp_2_web_thymeleaf>git add .\pom.xml
warning: in the working copy of 'pom.xml', LF will be replaced by CRLF the next time Git touches it
C:\Users\pc\Documents\lp-2\lp_2_web_thymeleaf>git add .\src\
warning: in the working copy of 'src', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/main/java/com/cibertec/Lp2WebThymeleafApplication.java', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/test/java/com/cibertec/Lp2WebThymeleafApplicationTests.java', LF will be replaced by CRLF the next time Git touches it
C:\Users\pc\Documents\lp-2\lp_2_web_thymeleaf>git commit -m "first commit"
[master (root-commit) 91fe5b2] first commit
 9 files changed, 179 insertions(+)
 create mode 100644 pom.xml
 create mode 100644 src/main/java/com/cibertec/Lp2WebThymeleafApplication.java
 create mode 100644 src/main/java/com/cibertec/controller/ProductController.java
 create mode 100644 src/main/resources/application.properties
 create mode 100644 src/main/resources/reporte/Productos01.jasper
 create mode 100644 src/main/resources/reporte/Productos01.jrxml
 create mode 100644 src/main/resources/templates/greeting.html
 create mode 100644 src/main/resources/templates/login.html
 create mode 100644 src/test/java/com/cibertec/Lp2WebThymeleafApplicationTests.java
```

Nota: Elaboración propia

3.2.3. Despliegue de aplicaciones web en Azure

Paso 1. En el portal de Azure seleccionar “Centro de Implementación” y completar la configuración. Para conectar con nuestra cuenta en GitHub debemos presionar el botón “Autorizar”:

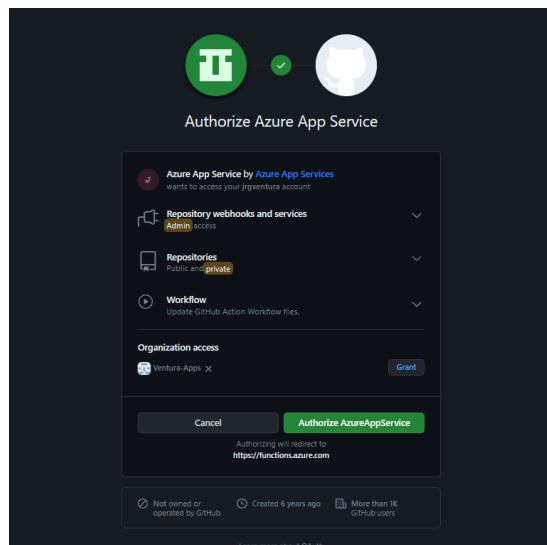
Figura 241: Configuración en Centro de Implementación



Nota: Adaptado de Azure

Paso 2. Aceptamos el permiso para conectar Azure con nuestra cuenta en GitHub.

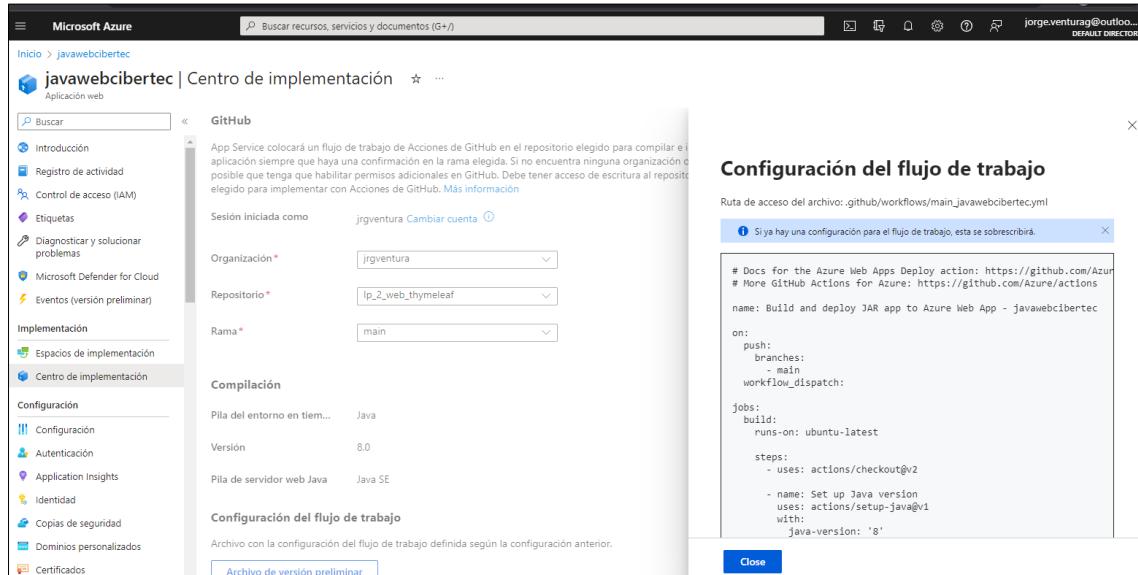
Figura 242: Permiso de configuración en Azure



Nota: Adaptado de Azure

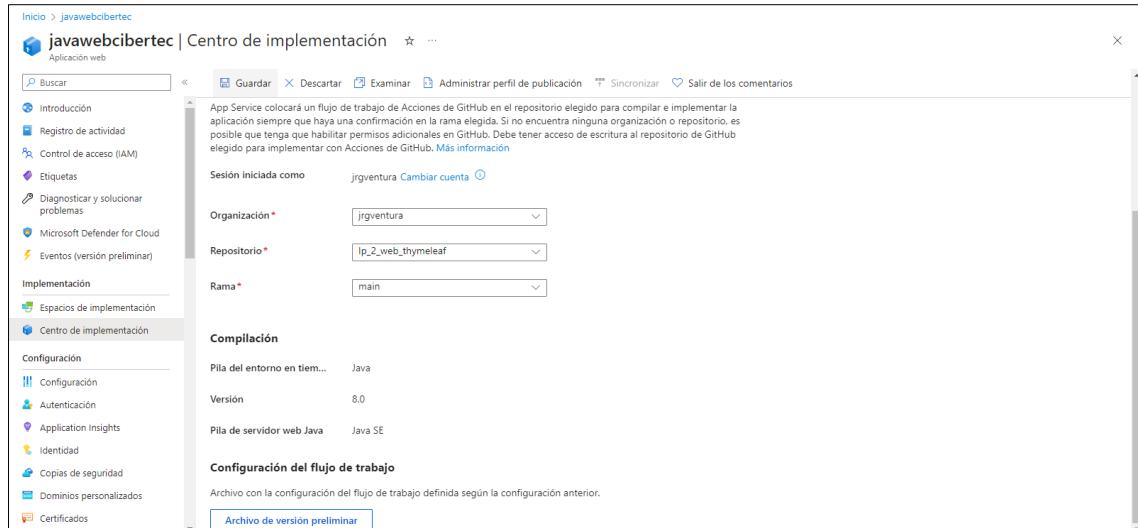
Paso 3. Terminamos con la configuración:

Figura 243: Configuración del Flujo de Trabajo



Nota: Adaptado de Azure

Figura 244: Configuración del Flujo de Trabajo



Nota: Adaptado de Azure

Paso 4. Selecciona la pestaña de Registros.

Figura 245: Centro de Implementación - Registros

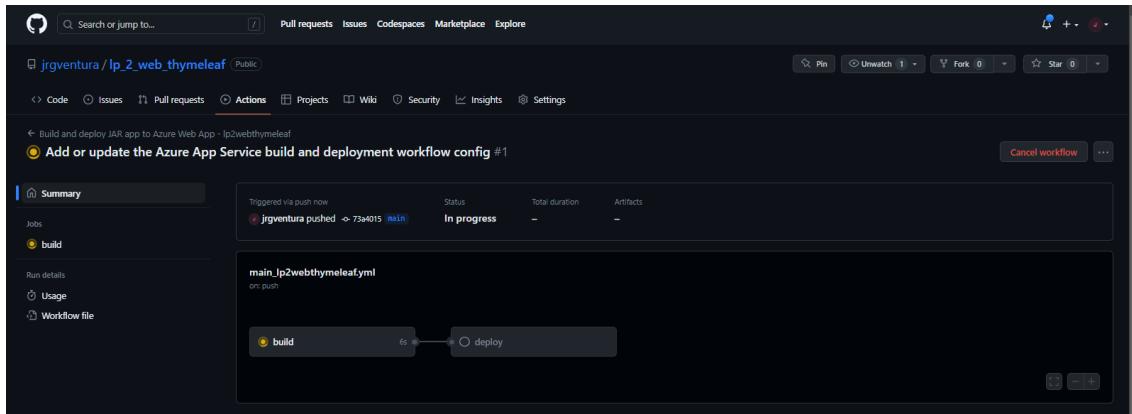
Nota: Adaptado de Azure

Figura 246: Centro de Implementación - Registros

Nota: Adaptado de Azure

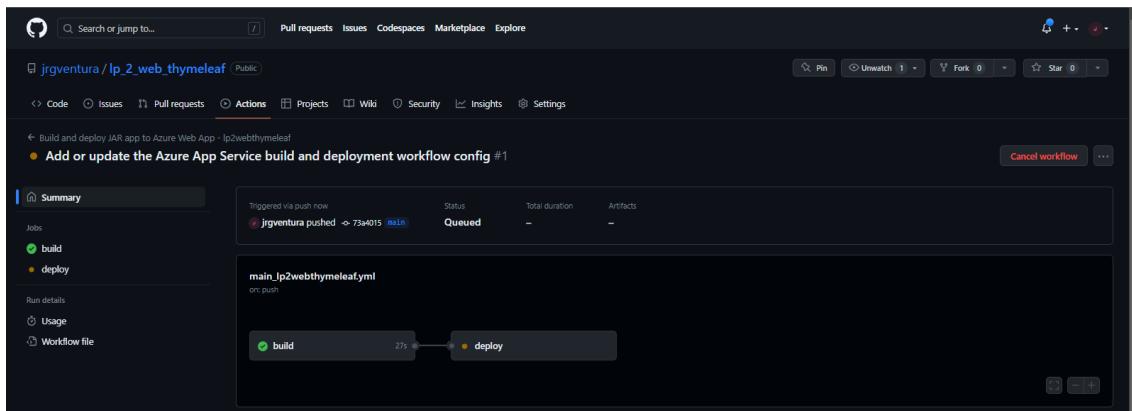
Ingresamos al enlace generado, y estaremos en **GitHub Actions** verificando el despliegue.

Figura 247: Despliegue con GitHub Actions



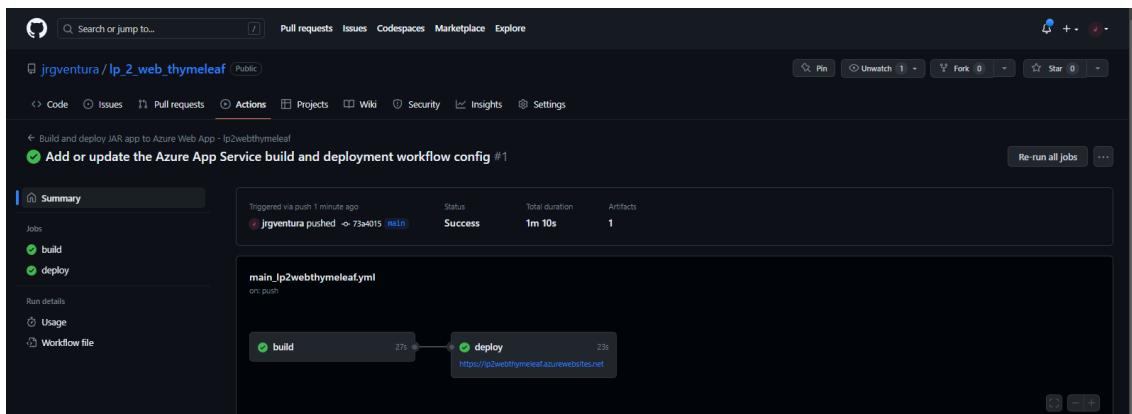
Nota: Adaptado de GitHub

Figura 248: Despliegue con GitHub Actions



Nota: Adaptado de GitHub

Figura 249: Despliegue con GitHub Actions

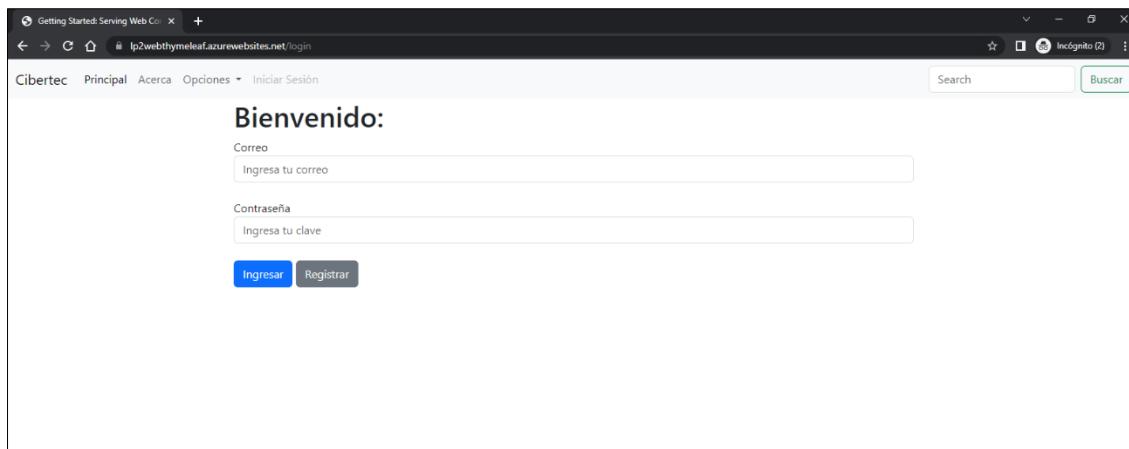


Nota: Adaptado de GitHub

Terminado el despliegue en GitHub Actions, ingresamos al enlace de la web creada en el portal Azure y nos mostrará la web desplegada y disponible.

En caso de presentar errores en el despliegue verificar que la versión de Java indicada en el proyecto sea la misma que la indicada al momento de crear la aplicación en Azure.

Figura 250: Portal desplegado correctamente



Nota: Adaptado de GitHub

Listo, la pantalla principal del sistema se visualiza.

Resumen

1. Azure es una plataforma de Microsoft que nos permite desplegar aplicaciones en la nube pública.
2. Azure nos brinda una experiencia de portal de autoservicio.
3. Azure permite la integración con una nube híbrida mediante el uso de una nube virtual.
4. La plataforma de Azure tiene un nivel gratuito para los estudiantes que nos brinda un crédito de \$100 en los servicios que ofrecen de nube pública.
5. Con la finalidad de agilizar el proceso de despliegue de la aplicación Azure nos permite poder conectar nuestro proyecto creado con GitHub y GitHub Actions.

Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- Página oficial: <https://azure.microsoft.com/es-es>
- Tutorial Publicar en Azure: <https://azure.microsoft.com/es-es/get-started/>

Bibliografía

- Auribox Training. (2018). *Creación de un proyecto Maven*. <https://blog.auriboxtraining.com/java/creacion-proyecto-maven/>
- Dauzon, S. (2018). *Git: Controle la gestión de sus versiones (conceptos, utilización y casos prácticos)*. ENI.
- Déléchamp, F. (2018). *Java y Eclipse: desarrolle una aplicación con Java y Eclipse*. ENI.
- Hibernate. (s.f.). *Hibernate ORM. Documentation - 5.4. Guides and such*. <https://hibernate.org/orm/documentation/5.4/>
- Jasper Community. (2023). *JasperReports Server*. Reporting and Analytics Server. Recuperado de <https://community.jaspersoft.com/>
- Jqueryvalidation (s.f.). *jQuery Validation Plugin*. <https://jqueryvalidation.org/>
- Lui, M., Gray M., Chan A., Long, J. (2011). *Pro Spring Integration*. Apress.
- Oracle. (2023). *Java Persistence API*. <http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>
- Oracle (2014). *Java Platform, Enterprise Edition: The Java EE Tutorial*. <https://docs.oracle.com/javaee/7/tutorial/index.html>
- Pérez Martínez, E. (2016). *Desarrollo de aplicaciones mediante el Framework de spring*. Ra-Ma.
- Kulkarni, R. (2018). *Java EE 8 Development with Eclipse: Develop, Test, and Troubleshoot Java Enterprise Applications Rapidly with Eclipse*, 3rd Edition. Packt Publishing.
- Sanderson, Dan (2010). *Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure (Animal Guide)*. O'Reilly Media.
- Siriwardena, P. (2015). *Maven Essentials: Get started with the essentials of Apache Maven and get your build automation system up and running quickly*. Packt Publishing.
- The Apache Software Foundation. (2023). *Apache Tomcat*. Recuperado de <http://tomcat.apache.org/>
- Teodor, D., Lucian, C. (2007). *The Definitive Guide to JasperReports*. Apress Berkeley.
- Walls, C. (2015). *Spring*. 4a ed. Anaya Multimedia.