Ingeniería en Desarrollo de Software

Actividad | 3 | Backend para el sistema web

Nombre del curso

Ingeniería en Desarrollo de Software

TUTOR: Aaron Salazar.

ALUMNO: Uziel de Jesús López Ornelas.

FECHA: 20 de febrero de 2026.

Tabla de contenido

Introducción

Para dar inicio a la actividad número 3 continuando con el desarrollo del proyecto anterior, la tienda Sara ha aprobado el diseño del sitio web, ahora solicita darle funcionalidad, esto a través de API REST, el objetivo es que desarrollemos una página web funcional especializada o centralizada en el carrito de compras, para ello utilizaremos visual Studio Code Java y Spring Boot. De acuerdo a esto haremos primero una tabla investigando todas las peticiones HTTP en una REST API, después de que tengamos la tabla completamente hecha desarrollaremos una serie de pasos que nos llevarán a la funcionalidad total de nuestro carrito de compras iniciando primero para crear archivos dentro de carpetas, modificando clases, agregando modelos, agregando códigos, métodos, vistas y sobre todo utilizaremos lo que es SQL Server, realizaremos dentro de las tablas de categoría y marca cinco modificaciones para que estas tengan un impacto a la hora de que nosotros queramos agregar algo a nuestro carrito de compras, editando o eliminando ciertos accesorios, por último ejecutaremos la aplicación que es por medio de un servidor en línea y tomaremos captura de las evidencias que hemos realizado.

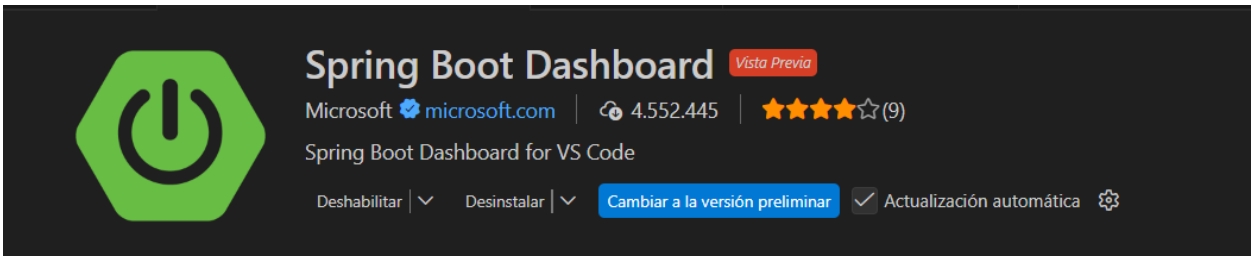
Investigación de la actividad

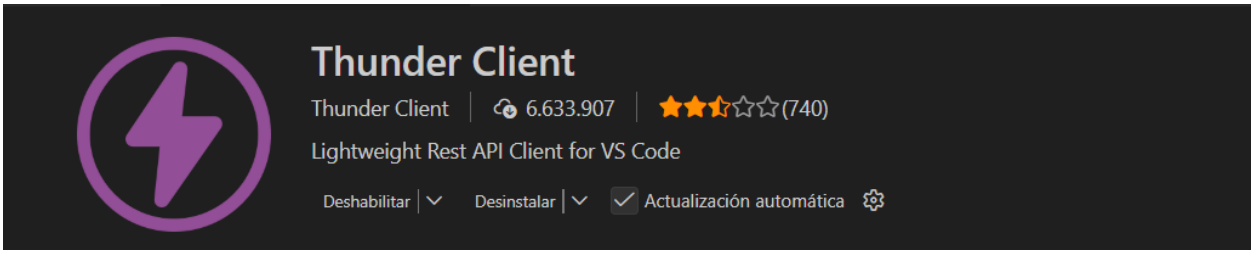
Tabla de API REST

Método	Función	Descripción	Ejemplo de uso
GET	Es un tipo de solicitud HTTP que utilizan los clientes, como los navegadores web, esto sirven para recuperar información de un servidor web.	Se caracteriza principalmente por solicitar datos a un servidor, este obviamente funciona con una URL lo que envía los parámetros en la misma y por último ese tipo de método no modifica el servidor.	Obtener usuarios de algún contexto, obtener datos de algún usuario en específico.
POST	Enviar datos al servidor para su posterior procesamiento, lo que conlleva la creación de un nuevo recurso, eso se ve reflejado mucho a la hora de crear o añadir algún producto en un sitio web.	Los datos enviados mediante este método se almacenan en el cuerpo del mensaje de la solicitud.	Usuario registra datos en un formulario y el servidor analiza y procesa los datos enviados, crear un nuevo usuario o un nuevo producto.
PUT	Sirve para actualizar un recurso en un servidor.	Es idem potente, lo que significa que varias solicitudes de estas mismas te dan impacto de solo una. Ayuda bastante a que la solicitud de repetidas no generen algún error.	Actualizar por completo la información de un usuario o reemplazar algún producto.
PATCH	Aplicar modificaciones parciales a algún recurso.	A diferencia del del método PUT, el método PATCH no es idem potente, esto causa que si hay peticiones idénticas sucesivamente puede resultar en errores.	Actualizar solamente el correo de algún usuario o cambiar el estado de un pedido.
DELETE	Eliminar recursos de un servidor.	Está diseñado para acciones destructivas, se especifica en la URL, encabezados y el cuerpo.	Eliminación de cuentas de usuario, eliminar registros o borrar caché y datos temporales.

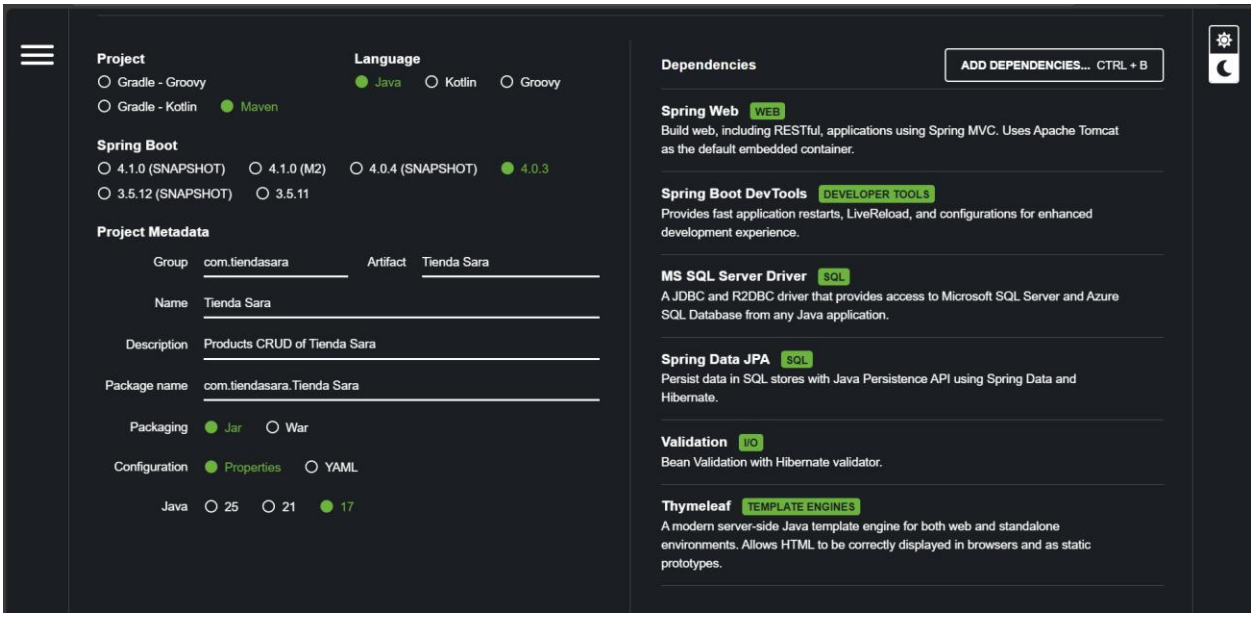
Desarrollo

Agregación de los puglings en VSCode

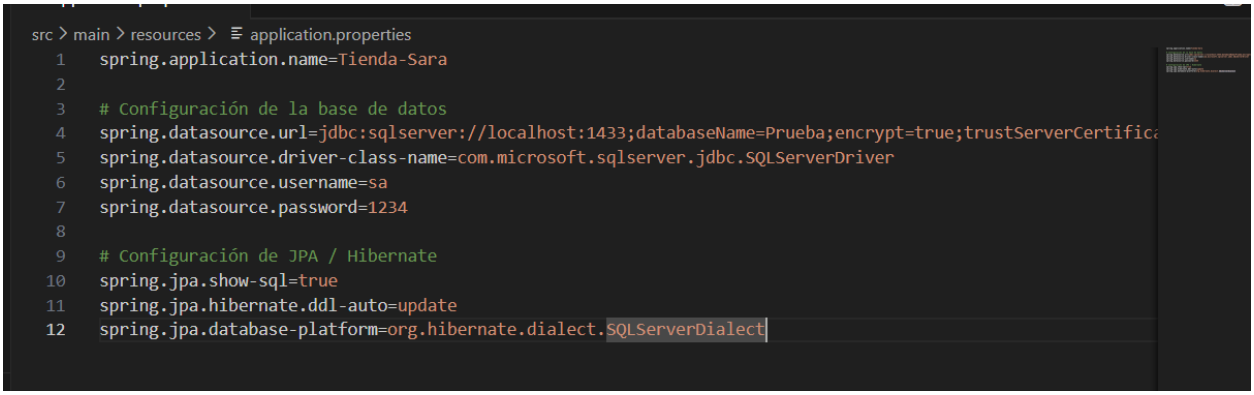




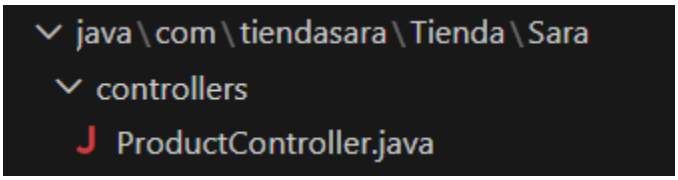
Creación del proyecto Java Spring Boot con dependencias



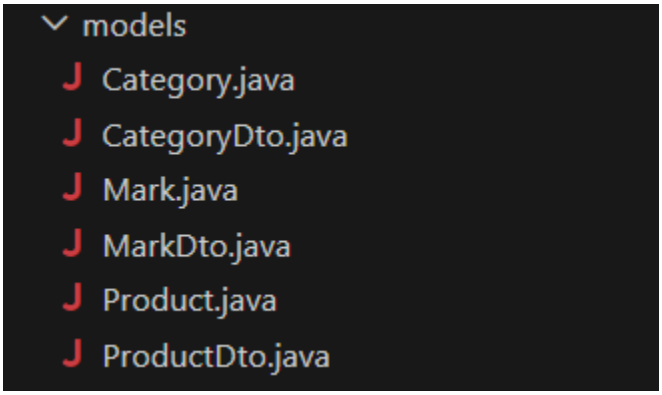
Modificación del archivo application.properties



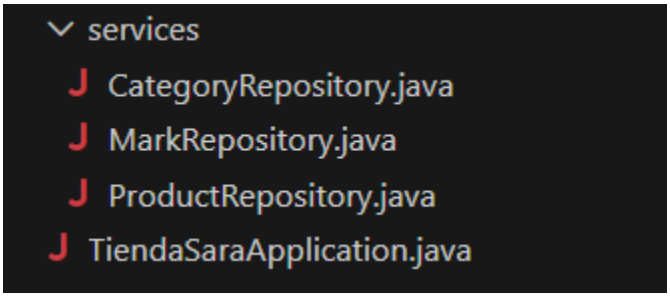
Creación de archivos y carpetas dentro de src/main/java/tiendasara/:controllers



Models/



Services/



Definir los getters y setters

application.properties | Category.java

src > main > java > com > tiendasara > Tienda > Sara > models > Category.java > ...

```java
package com.tiendasara.Tienda.Sara.models;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
```

application.properties | Category.java | CategoryDto.java

src > main > java > com > tiendasara > Tienda > Sara > models > CategoryDto.java > ...

```java
package com.tiendasara.Tienda.Sara.models;

import jakarta.validation.constraints.*;

public class CategoryDto {
    @Size(min = 10, message = "The description should be at least 10 characters")
    @Size(max = 2000, message = "The description cannot 2000 characters")
    private String description;
```

application.properties | Mark.java

src > main > java > com > tiendasara > Tienda > Sara > models > Mark.java > ...

```java
package com.tiendasara.Tienda.Sara.models;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="Marcas")
public class Mark {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
```

application.properties | Mark.java | MarkDto.java

src > main > java > com > tiendasara > Tienda > Sara > models > MarkDto.java > ...

```java
package com.tiendasara.Tienda.Sara.models;

import jakarta.validation.constraints.*;

public class MarkDto {
    @Size(min = 10, message = "The description should be at least 10 characters")
    @Size(max = 2000, message = "The description cannot 2000 characters")
    private String description;
```

application.properties | Product.java

src > main > java > com > tiendasara > Tienda > Sara > models > Product.java > ...

```java
package com.tiendasara.Tienda.Sara.models;

import java.util.Date;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
```

application.properties | Product.java | ProductDto.java

src > main > java > com > tiendasara > Tienda > Sara > models > ProductDto.java > ...

```java
package com.tiendasara.Tienda.Sara.models;

import org.springframework.web.multipart.MultipartFile;

import jakarta.validation.constraints.*;

public class ProductDto {
    @Size(min = 10, message = "The description should be at least 10 characters")
    @Size(max = 2000, message = "The description cannot 2000 characters")
    private String description;
```

Definir códigos correspondientes a los servicios

application.properties | Product.java | ProductDto.java | CategoryRepository.java

src > main > java > com > tiendasara > Tienda > Sara > services > CategoryRepository.java > ...

```java
package com.tiendasara.Tienda.Sara.services;

import org.springframework.data.jpa.repository.JpaRepository;
import com.tiendasara.Tienda.Sara.models.Category;

public interface CategoryRepository extends JpaRepository<Category, Integer>{

}
```

application.properties | Product.java | ProductDto.java | CategoryRepository.java | MarkRepository.java

src > main > java > com > tiendasara > Tienda > Sara > services > MarkRepository.java > ...

```java
package com.tiendasara.Tienda.Sara.services;

import org.springframework.data.jpa.repository.JpaRepository;
import com.tiendasara.Tienda.Sara.models.Mark;

public interface MarkRepository extends JpaRepository<Mark, Integer>{

}
```

application.properties ProductRepository.java

src > main > java > com > tiendasara > Tienda > Sara > services > ProductRepository.java > ...

```java
package com.tiendasara.Tienda.Sara.services;

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

import com.tiendasara.Tienda.Sara.models.Product;

public interface ProductRepository extends JpaRepository<Product, Integer> {

}

```

Definir los métodos

application.properties ProductRepository.java ProductController.java

src > main > java > com > tiendasara > Tienda > Sara > controllers > ProductController.java > ...

```java
package com.tiendasara.Tienda.Sara.controllers;

import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.Date;
import java.util.List;

```

Definir las vistas

application.properties index.html

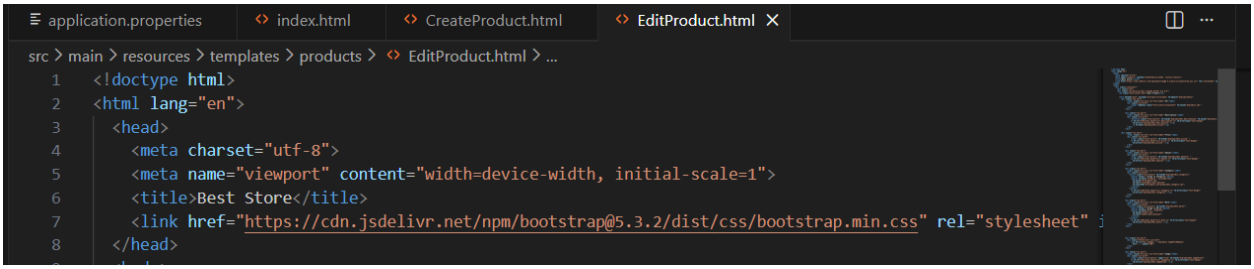
src > main > resources > templates > products > index.html > ...

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Best Store</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" j
  </head>
  <body>
    <div class="container">
        <h1 class="text-center my-4">Products</h1>

        <a class="btn btn-primary" href="/products/create">Create Product</a>

```

application.properties index.html CreateProduct.html

src > main > resources > templates > products > CreateProduct.html > ...

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Best Store</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" j
  </head>
  <body>
    <div class="container">
      <div class="row">
      <div class="col-md-8 mx-auto rounded border p-4 m-4">
```

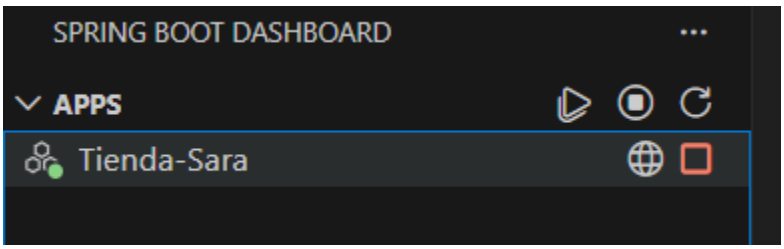


Realizar cinco inserciones a cada tabla de base de datos

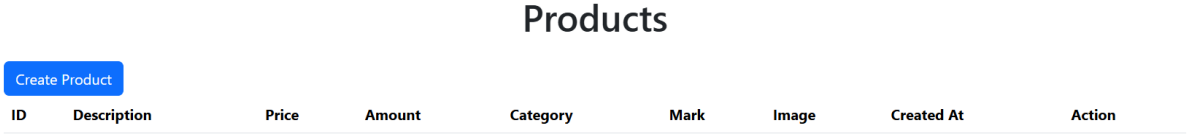
	id	description
1	1	Electronica
2	2	Movilidad
3	3	Linea blanca
4	4	Decoración
5	5	Ropa

	id	description
1	1	Asus
2	2	HP
3	3	Mabe
4	4	KTM
5	5	Zara

Ejecuta la aplicación y prueba el CRUD

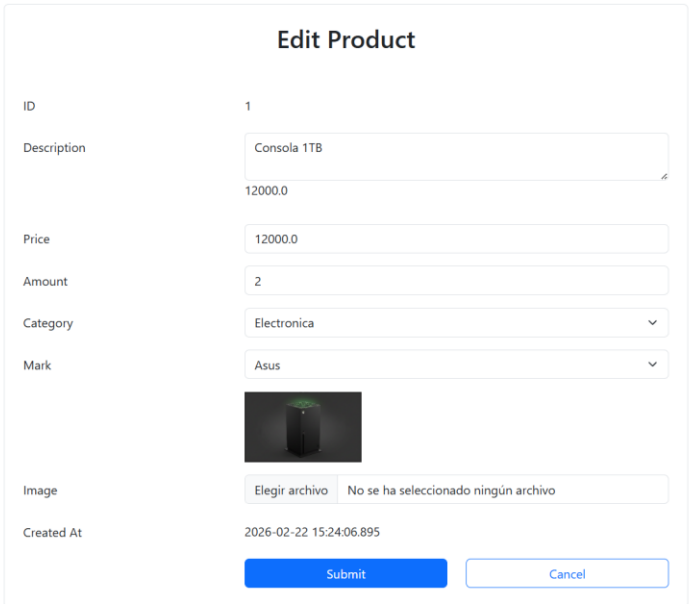


Página de carrito de compras



Create Product

ID	Description	Price	Amount	Category	Mark	Image	Created At	Action
1	Consola 1TB	$12000.0	2	Electronica	Asus		2026-02-22	Edit Delete



Link de GitHub

https://github.com/UZLOP984/Sistemas_Web_II.git

Conclusión

Sin duda alguna esta es una de las actividades más laboriosas que hemos hecho a lo largo de esta materia, no solamente es conocer conceptos sino que también es emplearlos y utilizarlos para resolver la problemática, en este caso la problemática daba en hacer funcionar una página web, o como ya se mencionó anteriormente el carrito de compras en especial, dentro de él nosotros podemos agregar productos, editar los detalles del mismo producto artículo, agregarlo alguna clase que es categoría o marca y por supuesto también agregando el valor alguna imagen representativa y que esta se vea reflejada en la decisión final del consumidor. En un inicio tuve detalles, tuve problemas para poder realizar esta actividad ya que algunos conceptos eran nuevos para mí y no tenía muy bien configurada lo que era mi aplicación de visual Studio Code pero gracias a guías y materiales de estudio junto con los maestros puede realizar actividad y presentarla como es debido.

Referencias

GET Method. (s. f.). F5, Inc. https://www.f5.com/glossary/get-method

PATCH - HTTP | MDN. (2025, 7 julio). https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Methods/PATCH

DELETE Method in APIs - Abstract API. (2025, 24 febrero). https://www.abstractapi.com/guides/api-glossary/delete-method-in-apis