

# **PILL-EXPRESS FCT**

**Oscar Arjona Ecija  
Pablo Galván Tejada**

**JOSÉ MANUEL  
RAYO**

**Desarrollo de Aplicaciones Multiplataforma  
Curso 2023-2024**

**iFP Julián Camarillo  
Convocatoria de Presentación: Junio 2024**

# 1. INDICE

2. Resumen.....	2
3. Introducción.....	3
4. Motivación del Proyecto.....	5
5. Objetivos.....	7
6. Marco Teórico.....	8
7. Marco Práctico.....	10
7.1 Inicio.....	10
7.2 Registro.....	12
7.3 Ingreso.....	16
7.4 Menú.....	18
7.5 Administrador-Usuarios.....	20
7.6 Administrador-Roles.....	27
7.7 Administrador-Ubicaciones.....	33
7.8 Administrador-Productos.....	38
7.9 Administrador-Almacenes.....	41
7.10 Administrador-Pedidos.....	43
7.11 Administrador-Detalles Pedidos.....	45
7.12 Tienda.....	48
7.13 Repartidores-Mapa.....	55
7.14 Repartidores-Pedidos Pendientes.....	58
7.15 Pom.xml y application.properties.....	59
7.16 Base de Datos.....	61
8. Conclusiones.....	62
9. Bibliografía.....	65

## 2. Resumen.

Nuestro proyecto de trabajo de fin de grado se enfoca en el desarrollo de una página web utilizando Eclipse y Java, respaldada por una base de datos MYSQL. Principalmente, hemos empleado el Framework Spring, el cual ha simplificado y acelerado la construcción de nuestra aplicación. Gracias a su enfoque modular y flexibilidad, hemos podido destacar en áreas clave como la inyección de dependencias, la gestión de bases de datos, la configuración de seguridad y la encriptación, entre otros aspectos. Además, hemos integrado lenguajes complementarios como HTML, CSS, Bootstrap y JavaScript para una experiencia de usuario completa.

La esencia de nuestra aplicación reside en la facilitación de la venta y entrega de productos farmacéuticos a domicilio. Esta plataforma cuenta con tres roles principales: clientes, administradores y repartidores. Los clientes tienen la capacidad de realizar pedidos y dejar reseñas, mientras que los administradores se encargan de la gestión de los procesos de la página. Por último, los repartidores se dedican al transporte de los pedidos desde los almacenes hasta los domicilios de los clientes.

En resumen, nuestro objetivo es aplicar los conocimientos adquiridos en el grado de Desarrollo de Aplicaciones Multiplataforma para crear una página web con una aplicación práctica y relevante en el mundo laboral, respondiendo a la creciente demanda de servicios de entrega a domicilio y aprovechando las tecnologías más avanzadas en el desarrollo web.

### 3. Introducción.

Nuestro proyecto de trabajo de fin de grado se centra en la creación de una empresa dedicada a la comercialización y distribución de productos farmacéuticos, utilizando principalmente el lenguaje de programación Java.

A lo largo de esta introducción, detallaremos los diversos elementos que conforman nuestro proyecto de fin de grado. En primer lugar, exploraremos las motivaciones que nos llevaron a enfocarnos en la actividad principal de nuestra página web, así como el contexto que dio origen a esta idea y el análisis de empresas relacionadas con el sector de la distribución y venta electrónica.

En el siguiente apartado, nos concentraremos en establecer los objetivos que buscamos alcanzar con nuestra página web, abarcando aspectos tanto estudiantiles como laborales y económicos.

Posteriormente, presentaremos nuestras hipótesis sobre las finalidades y metas que esperamos lograr con este proyecto, aunque estas se plantean antes de su finalización.

El apartado siguiente se centrará en el aspecto teórico del trabajo, detallando los diversos lenguajes de programación, Frameworks y herramientas empleadas, así como sus características y funcionamiento.

A continuación, exploraremos el apartado práctico del proyecto, destacando cómo hemos implementado las diferentes funciones de la página web, la interacción con la base de datos MYSQL, la distribución de lenguajes y herramientas utilizadas en distintas secciones del proyecto, y complementaremos esta información con capturas de pantalla para una comprensión más clara del funcionamiento y procesos de la página web.

Esta parte práctica se subdividirá en las siguientes secciones:

- **Sección de Clientes:** Presentaremos una página web con una amplia gama de medicamentos disponibles, diferenciando entre aquellos que requieren receta y los que no. Los usuarios podrán buscar medicamentos, visualizar detalles de productos, añadir productos a la cesta de compra, adjuntar recetas si es necesario, seleccionar el método de pago y seguir el estado del pedido a través de un sistema de GPS.
- **Sección de Valoraciones:** Integrada en la sección de Clientes, los usuarios podrán encontrar feedback sobre los productos, incluyendo valoraciones numéricas, efectos secundarios, facilidad de uso, eficacia y opiniones generales.

- Sección de Administrador: Esta área incluirá funciones de ERP para la gestión integral de la aplicación, como la administración de usuarios, productos, trabajadores, pedidos, recetas, compras, ventas, facturas, y servirá como intermediario entre clientes y repartidores.
- Sección de Repartidores: Aquí se detallarán las funciones asignadas a los repartidores, quienes recibirán notificaciones de nuevos pedidos, accederán a los datos del cliente y del pedido, y seguirán la ruta hasta el domicilio del cliente utilizando un sistema de GPS.

Finalmente, en el último apartado, se presentarán las conclusiones del proyecto, destacando el proceso de desarrollo, posibles mejoras futuras y las ideas surgidas durante su realización.

## 4. Motivación del Proyecto.

La concepción de Pill-Express surge en respuesta a un conjunto de circunstancias que han transformado drásticamente la forma en que interactuamos con el mundo que nos rodea. El brote de la pandemia de COVID-19 ha sido un catalizador sin precedentes, desafiando nuestras rutinas cotidianas y exponiendo las vulnerabilidades de nuestra sociedad. En medio de este panorama, la necesidad de adaptarnos y encontrar soluciones innovadoras se ha vuelto más urgente que nunca.

El impacto del COVID-19 ha sido multifacético. Uno de los aspectos más evidentes ha sido la aceleración del uso de la tecnología como herramienta para facilitar nuestras vidas diarias. Las aplicaciones de compra y entrega de productos se han vuelto omnipresentes, ofreciendo comodidad y seguridad a millones de personas en todo el mundo. Esta tendencia ha señalado un cambio fundamental en nuestros hábitos de consumo, y Pill-Express surge como una respuesta específica a esta demanda emergente.

Sin embargo, más allá de ser una simple reacción a una tendencia del mercado, Pill-Express encuentra su verdadera motivación en el deseo de servir a aquellos que enfrentan desafíos únicos en su acceso a los servicios de salud. La pandemia ha puesto de relieve la vulnerabilidad de ciertos grupos de la población, como las personas mayores, aquellos con problemas de movilidad y aquellos que viven solos y se enfrentan a dificultades para salir de casa.

Pill-Express se inspira en la necesidad de proporcionar una solución integral para estas personas, garantizando que tengan acceso oportuno y seguro a los productos farmacéuticos que necesitan. Ya sea que se trate de medicamentos recetados esenciales, productos de venta libre o suministros médicos especializados, nuestro objetivo es brindar una plataforma confiable y conveniente que aborde las necesidades de salud de aquellos que enfrentan barreras físicas o sociales para acceder a ellas.

Al reconocer la importancia de la conexión humana y la atención personalizada, Pill-Express no solo se trata de facilitar transacciones comerciales, sino también de proporcionar un servicio empático y solidario. Nos comprometemos a establecer relaciones de confianza con nuestros usuarios, brindando apoyo y asistencia en cada paso del proceso. Entendemos que detrás de cada pedido hay una historia única y una necesidad genuina, y nos esforzamos por abordarlas con la atención y el cuidado que merecen.

En resumen, la motivación detrás del proyecto Pill-Express es una combinación de factores externos y una profunda empatía por aquellos que enfrentan desafíos en su acceso a la atención médica. A través de la innovación tecnológica y un enfoque centrado en el cliente, aspiramos a hacer una diferencia significativa en la vida de aquellos que más lo necesitan, proporcionando una vía de acceso segura, conveniente y confiable a los productos farmacéuticos esenciales.

## 5. Objetivos.

El objetivo principal de nuestro proyecto es desarrollar una página web funcional dedicada a la entrega de medicamentos a domicilio. Utilizaremos una variedad de lenguajes de programación y herramientas para lograr este propósito, con un enfoque específico en el uso de Spring para mejorar nuestras habilidades en el desarrollo de páginas web.

En primer lugar, nos comprometemos a crear una plataforma robusta y eficiente que permita a los usuarios realizar pedidos de medicamentos desde la comodidad de sus hogares y recibirlos en su puerta de manera segura y oportuna. Esto implica la implementación de funciones como búsqueda de productos, gestión de carritos de compra, procesamiento de pagos y seguimiento de pedidos mediante un sistema de GPS, entre otros aspectos.

Además, consideramos este proyecto como una oportunidad invaluable para aprender y mejorar en la programación de páginas web utilizando el Framework Spring. Nos enfocaremos en comprender y aplicar los principios y conceptos fundamentales de Spring, aprovechando su modularidad, flexibilidad y potencia para optimizar el desarrollo de nuestra aplicación.

Otro objetivo clave es demostrar los conocimientos adquiridos a lo largo del curso de Desarrollo de Aplicaciones Multiplataforma. Este proyecto servirá como una prueba tangible de nuestras habilidades en el diseño, implementación y despliegue de una aplicación web compleja, mostrando nuestra capacidad para enfrentar desafíos técnicos y resolver problemas de manera efectiva.

Finalmente, aspiramos a que este proyecto tenga un impacto significativo en nuestro futuro profesional, ya que la experiencia adquirida y la calidad del producto final podrían abrirnos puertas en el mundo laboral. Al demostrar nuestras habilidades en el desarrollo de aplicaciones web utilizando tecnologías modernas y demandadas, esperamos posicionarnos como candidatos atractivos para oportunidades de empleo relacionadas con el desarrollo de software y la tecnología web.

En resumen, nuestro objetivo es crear una página web funcional de entrega de medicamentos a domicilio, utilizando Spring y otras herramientas de programación web, con el propósito de aprender, demostrar nuestros conocimientos y habilidades, y potenciar nuestras oportunidades laborales en el campo de la tecnología.



## 6. Marco Teórico.

En esta sección, nos adentraremos en los componentes teóricos fundamentales que constituyen las herramientas empleadas en el desarrollo del proyecto de fin de grado. Nuestro objetivo es proporcionar una comprensión integral de su funcionamiento y su relevancia en el contexto del desarrollo de aplicaciones web.

**Lenguaje de Programación Java:** Java, como lenguaje de programación orientado a objetos, se destaca por su portabilidad, seguridad y versatilidad. Su robusta sintaxis y amplia gama de librerías lo convierten en una elección adecuada tanto para proyectos web como para aplicaciones.

**Entorno de Desarrollo Eclipse:** Eclipse, un entorno de desarrollo integrado (IDE), provee herramientas avanzadas para escribir, compilar y depurar código Java. Destaca por su interfaz intuitiva y sus numerosas funcionalidades, como el resaltado de sintaxis, el completado automático de código y la depuración paso a paso, es el principal entorno de desarrollo que hemos empleado a lo largo del curso y con el cual tenemos una mayor experiencia.

**Base de Datos MySQL:** MySQL es un sistema de gestión de bases de datos relacional, destaca por ser ampliamente utilizado en aplicaciones web. Utiliza el modelo cliente-servidor y el lenguaje SQL para almacenar, gestionar y recuperar datos de manera eficiente. Su capacidad para manejar grandes volúmenes de información y su rendimiento optimizado lo hacen ideal para nuestra página web, ya que requiere de un almacenamiento y recuperación de datos de manera confiable.

**Maven:** Maven es una herramienta de gestión de proyectos de software que facilita la construcción, el manejo de dependencias y la generación de informes. Una de las características más destacadas de Maven es su capacidad para automatizar el proceso de gestión de dependencias. A través de su archivo de configuración centralizado, conocido como pom.xml, Maven permite especificar las dependencias del proyecto, así como sus versiones correspondientes. Esto simplifica enormemente la tarea de gestionar las bibliotecas y frameworks utilizados en el proyecto, ya que Maven se encarga de descargar automáticamente las dependencias necesarias del repositorio central de Maven o de repositorios personalizados.

Framework Spring: Spring, un Framework de aplicación Java, proporciona una amplia gama de herramientas y características para simplificar el desarrollo de aplicaciones empresariales. Componentes clave incluyen:

- Spring MVC (Modelo-Vista-Controlador): Facilita la creación de aplicaciones web basadas en el patrón de diseño Modelo-Vista-Controlador, separando la lógica de negocio de la presentación de datos.
- Spring Data JPA: Simplifica el acceso a datos en aplicaciones Java mediante el uso de repositorios, eliminando la necesidad de escribir consultas SQL manualmente.
- Spring Security: Se encarga de la seguridad de la aplicación, proporcionando funciones de autenticación y autorización, protección contra ataques comunes y gestión de sesiones de usuario.
- Spring Boot: Simplifica el desarrollo de aplicaciones Spring al proporcionar una configuración predeterminada y autoconfigurada, permitiendo crear rápidamente aplicaciones listas para producción.

Tecnologías Frontend: En el Frontend de las aplicaciones web hemos utilizado diversas tecnologías para crear interfaces de usuario atractivas y funcionales. HTML proporciona la estructura básica de las páginas web, CSS se encarga de la presentación y el estilo visual, Bootstrap facilita el diseño responsivo de las interfaces de usuario y JavaScript añade interactividad y dinamismo a nuestro proyecto.

En conclusión, las herramientas teóricas utilizadas en el proyecto proporcionan una base sólida y eficiente para el desarrollo de aplicaciones web modernas y efectivas, como lo es nuestra página web de Pill-Express. Desde el lenguaje de programación Java hasta el framework Spring y las tecnologías frontend, cada componente desempeña un papel crucial en la creación y operación exitosa de nuestro proyecto.

## 7. Marco Práctico.

### 7.1 Inicio.

El código es una página HTML que utiliza Thymeleaf como motor de plantillas, indicado por los atributos "th" en el HTML. La página está diseñada para ser una interfaz de ingreso al sistema "Pill-Express".

```
<!doctype html>
<html xmlns:th="http://www.thymeleaf.org">
```

En la sección <head>, se incluyen metadatos y enlaces a archivos CSS para estilos, además del propio estilo de CSS en <style>.

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">
  <link rel="icon" href="favicon.ico">

  <title>Pill-Express | Ingresar al Sistema</title>
  <!-- Bootstrap core CSS -->
  <link th:href="@{/bootstrap/css/bootstrap.min.css}" rel="stylesheet">
  <!-- Custom styles for this template -->
  <link th:href="@{/bootstrap/css/jumbotron.css}" rel="stylesheet">
  <link th:href="@{/bootstrap/css/sticky-footer-navbar.css}" rel="stylesheet">

  <style>
    /* Estilos personalizados */
    body {
      background-color: #e83b6c; /* Color de fondo */
    }

    /* Contenedor para centrar el contenido */
    .centered-content {
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      min-height: 100vh; /* Ajusta la altura para ocupar la pantalla completa */
      margin-top: -30px; /* Espacio en la parte superior */
    }
  </style>
</head>
```

En el <body> encontramos los siguientes elementos:

- <header th:insert="fragments/menú :: menú-principal"></header>: Inserta un fragmento de Thymeleaf llamado menú-principal desde fragments/menú. Esto se usa para incluir un menú de navegación común.
- <div class="centered-content">: Crea un contenedor centrado para el contenido principal.
- <main role="main">: Define el contenido principal de la página.
- <a th:href="@{/menú}">: Un enlace que redirige a la ruta /menú.

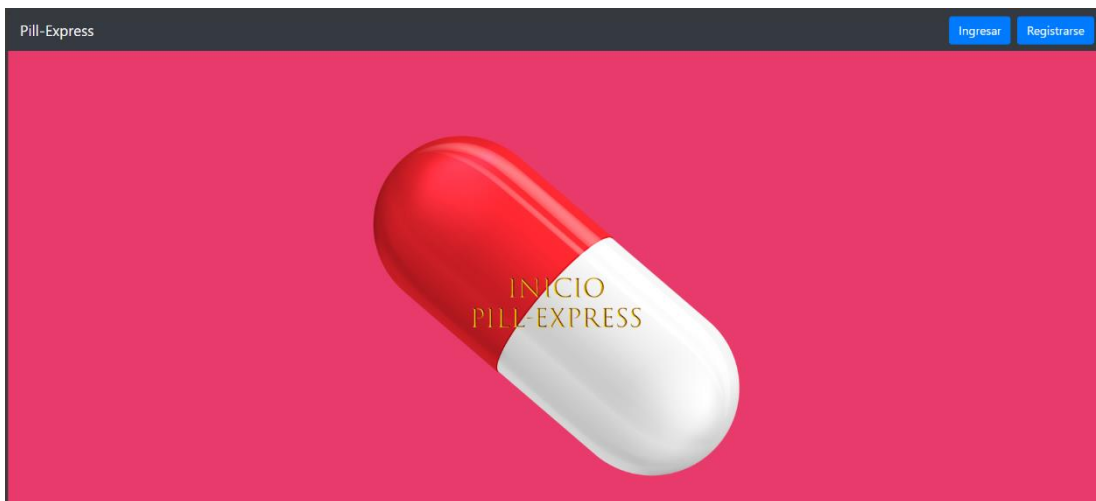
- ``: Muestra una imagen con un enlace.
- `<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>`: Incluye la librería jQuery.
- `<script th:src="@{/bootstrap/js/bootstrap.min.js}"></script>`: Incluye el JavaScript de Bootstrap mediante Thymeleaf.

```
<body>

<header th:insert="fragments/menu :: menu-principal"></header>

<!-- Contenedor para centrar el contenido -->
<div class="centered-content">
    <main role="main">
        <!-- Aquí incluyo la imagen con el enlace -->
        <a th:href="@{/menu}">
            
    </main>
</div>
```

Con la aplicación activada se muestra el logo de la empresa y como anteriormente hemos indicado, al pulsar lleva al menú de la página web, pero antes se necesita ingresar o registrarse para poder acceder, en la parte superior se observa el navbar que ha sido implementado con las opciones: Ingresar y Registrarse.



Todo esto se muestra gracias al HomeController.java que emplea estos métodos:

- Controlador con `@GetMapping`: Mapea solicitudes GET a la ruta `"/`.
- Método: `mostrarHome` que devuelve el nombre de la vista `"home"`.
- Vista: Spring renderiza la vista correspondiente.

```
@GetMapping("/")
public String mostrarHome () {
    return "home";
}
```

## 7.2 Registro.

El Registro emplea un método en HomeController.java para mostrarse:

```
@GetMapping("/signup")  
public String registrarse(Usuario usuario) {  
    return "formRegistro";  
}
```

Y utiliza el siguiente método para hacer el registro:

1. Anotación @PostMapping("/signup"):
  - Esta anotación mapea solicitudes HTTP POST a la ruta /signup a este método. Es decir, este método se ejecuta cuando un formulario de registro se envía a /signup.
  -
2. Parámetros del Método:
  - Usuario usuario: Este parámetro se vincula automáticamente a los datos del formulario enviados. Spring lo convierte en un objeto Usuario.
  - RedirectAttributes attributes: Utilizado para pasar atributos a la vista redirigida después de que el método se complete.
  -
3. Encriptación de la Contraseña:
  - String pwdPlano = usuario.getPassword();: Recupera la contraseña en texto plano del objeto Usuario.
  - String pwdEncriptado = passwordEncoder.encode(pwdPlano);: Encripta la contraseña usando BCryptPasswordEncoder.
  - Usuario.setPassword(pwdEncriptado);: Actualiza la contraseña del usuario con la versión encriptada.
4. Configuración del Usuario:
  - Usuario.setEstatus(1);: Establece el estado del usuario como activo.
  - Usuario.setFechaRegistro(new Date());: Establece la fecha de registro del usuario a la fecha y hora actual del servidor.
5. Asignación de Perfil:

- Perfil perfil = new Perfil();: Crea un nuevo objeto Perfil.
- Perfil.setId(3);: Establece el ID del perfil a 3, que representa el perfil de usuario estándar.
- Usuario.agregar(perfil);: Agrega el perfil al usuario. Se asume que agregar es un método en la clase Usuario que añade perfiles a una lista o colección dentro del usuario.

#### 6. Guardado en la Base de Datos:

- ServiceUsuarios.guardar(usuario);: Guarda el usuario en la base de datos. Se asume que serviceUsuarios es un servicio que maneja la persistencia de usuarios y que este método también guarda automáticamente el perfil del usuario.

#### 7. Redirección y Mensaje de Confirmación:

- Attributes.addFlashAttribute("msg", "Has sido registrado. ¡Ahora puedes ingresar al sistema!");: Añade un mensaje flash que se muestra en la vista redirigida.
- Return "redirect:/login";: Redirige al usuario a la página de inicio de sesión.

```
@PostMapping("/signup")
public String guardarRegistro(Usuario usuario, RedirectAttributes attributes) {
    // Recuperamos el password en texto plano
    String pwdPlano = usuario.getPassword();
    // Encriptamos el pwd BCryptPasswordEncoder
    String pwdEncriptado = passwordEncoder.encode(pwdPlano);
    // Hacemos un set al atributo password (ya viene encriptado)
    usuario.setPassword(pwdEncriptado);
    usuario.setEstatus(1); // Activado por defecto
    usuario.setFechaRegistro(new Date()); // Fecha de Registro, la fecha actual del servidor

    // Creamos el Perfil que le asignaremos al usuario nuevo
    Perfil perfil = new Perfil();
    perfil.setId(3); // Perfil USUARIO
    usuario.agregar(perfil);

    /**
     * Guardamos el usuario en la base de datos. El Perfil se guarda automáticamente
     */
    serviceUsuarios.guardar(usuario);

    attributes.addFlashAttribute("msg", "Has sido registrado. ¡Ahora puedes ingresar al sistema!");
    return "redirect:/login";
}
```

A continuación, el HTML está estructurado de la siguiente manera:

- El <head> con los metadatos, la implementación del CSS, el título, etcétera.

```
<!doctype html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">
  <link rel="icon" href="favicon.ico">

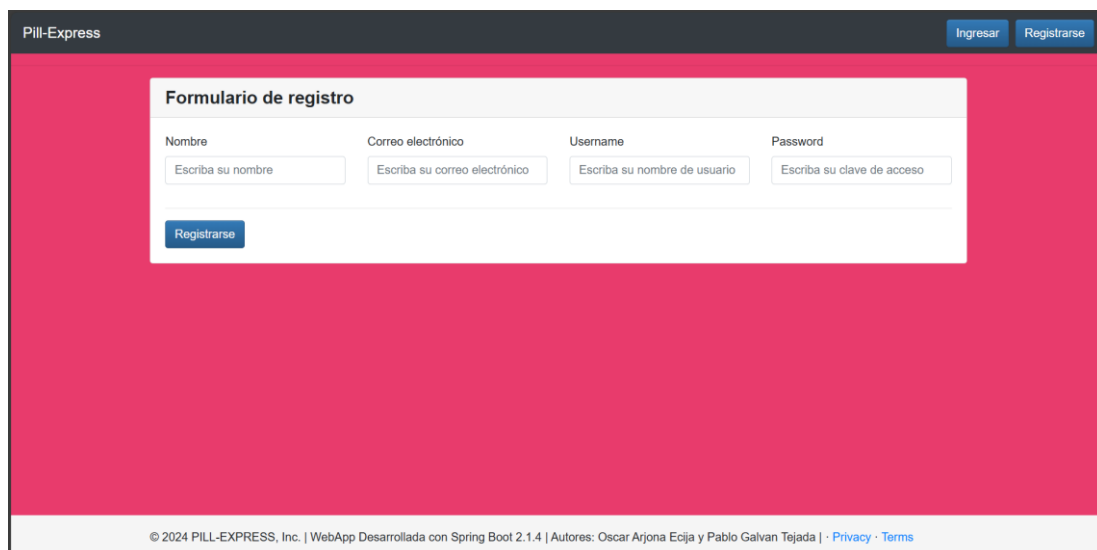
  <title>EmpleosApp | Formulario de registro</title>
  <link th:href="@{/bootstrap/css/bootstrap.min.css}" rel="stylesheet">
  <!-- Custom styles for this template -->
  <link th:href="@{/bootstrap/css/jumbotron.css}" rel="stylesheet">
  <link th:href="@{/bootstrap/css/sticky-footer-navbar.css}" rel="stylesheet">
  <link href="https://use.fontawesome.com/releases/v5.5.0/css/all.css" rel="stylesheet">
  <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css"
    integrity="sha384-fLW2N01lMgjakBkx3l/M9EahuwpsfEhNvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r"
    crossorigin="anonymous">
</head>
<script
  src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"
  integrity="sha384-0mSbJDEHialfmuBBQP6A4Qrprq5OVfW37PRR3j5ELqxxslyVqOtnepnHVP9aJ7xS"
  crossorigin="anonymous"></script>
</head>
<style>
body {
  font-family: Helvetica;
  -webkit-font-smoothing: antialiased;
  background: #e83b6c;
}
</style>
```

En el <body> encontramos los siguientes elementos:

1. <header>:
  - Inserta un fragmento Thymeleaf para el menú principal.
2. El contenido principal:
  - Estructura del formulario: El formulario utiliza Bootstrap para un diseño responsivo.
  - Utiliza th:action="@{/signup}" para definir la acción del formulario, redirigiendo a /signup en el servidor.
  - Th:object="{usuario}" enlaza el formulario con un objeto usuario.
  - Los campos del formulario están definidos con th:field="\*{campo}" para enlazar con los atributos del objeto usuario.
  - Cada campo está dentro de un div con clase col-md-3 para una distribución en una fila.
3. <footer>:
  - Inserta un fragmento Thymeleaf para el pie de página.

```
<main role="main">
<hr>
<div class="container">
<div class="card">
<h4 class="card-header"><strong>Formulario de registro</strong></h4>
<div class="card-body">
<form th:action="@{/signup}" th:object="${usuario}" method="post">
<div class="row">
<div class="col-md-3">
<div class="form-group">
<label for="nombre">Nombre</label>
<input type="text" class="form-control" th:field="**{nombre}" id="nombre" name="nombre" placeholder="Nombre" />
</div>
</div>
<div class="col-md-3">
<div class="form-group">
<label for="email">Correo electrónico</label>
<input type="email" class="form-control" th:field="**{email}" id="email" name="email" placeholder="Correo electrónico" />
</div>
</div>
<div class="col-md-3">
<div class="form-group">
<label for="username">Username</label>
<input type="text" class="form-control" th:field="**{username}" id="username" name="username" placeholder="Username" />
</div>
</div>
<div class="col-md-3">
<div class="form-group">
<label for="password">Password</label>
<input type="password" class="form-control" th:field="**{password}" id="password" name="password" placeholder="Password" />
</div>
</div>
</div>
<hr>
<button type="submit" title="Quiero registrarme." class="btn btn-primary">Registrarse</button>
</form>
</div>
</div>
</div> <!-- /container -->
</main>
```

Con la aplicación activada se muestra en la parte superior el navbar, en el centro el formulario para realizar el registro con los diferentes campos, además del botón de registrarse que realiza el método /signup y, por último, el footer que ha sido implementado.



The screenshot shows the Pill-Express web application. At the top, there is a dark blue header with the text "Pill-Express" on the left and two buttons, "Ingresar" and "Registrarse", on the right. Below the header is a large pink area containing a white registration form titled "Formulario de registro". The form has four input fields: "Nombre" (with placeholder "Escriba su nombre"), "Correo electrónico" (with placeholder "Escriba su correo electrónico"), "Username" (with placeholder "Escriba su nombre de usuario"), and "Password" (with placeholder "Escriba su clave de acceso"). Below these fields is a blue "Registrarse" button. At the bottom of the page, there is a footer with the text: "© 2024 PILL-EXPRESS, Inc. | WebApp Desarrollada con Spring Boot 2.1.4 | Autores: Oscar Arjona Ecija y Pablo Galvan Tejada | · Privacy · Terms".



## 7.3 Ingreso.

El HomeController.java contiene el método para mostrarlo:

```

    @GetMapping("/login")
    public String mostrarLogin() {
        return "formLogin";
    }

```

A continuación, los principales elementos del HTML en orden son los siguientes:

### 1. Formulario de Inicio de Sesión:

- Acción del formulario: `th:action="@{/login}"` establece la acción del formulario, dirigiendo la solicitud POST al endpoint `/login` en el servidor.
- Estructura del formulario: La clase `form-signin` y la disposición de los elementos dentro de un `div` con clase `card` proporcionan un diseño responsivo y atractivo utilizando Bootstrap.

```

<div class="card">
  <div class="card-body">
    <form th:action="@{/login}" method="post" class="form-signin">
      <div class="row">
        <div class="col-md-4">

```

### 2. Imagen y Título:

- Estos elementos proporcionan una imagen y un título centrados y estilizados para la página de inicio de sesión.

```

<div class="col-md-4">
  
  <h4 class="h3 mb-3 font-weight-normal">Ingresar</h4>

```

### 3. Mensajes de Error y Éxito:

- Estos elementos muestran mensajes de error si las credenciales son incorrectas y mensajes de éxito si hay algún mensaje (por ejemplo, después de un registro exitoso).

```

<div th:if="${param.error}">
  <span class="text-danger">Usuario / Password incorrectos!</span>
</div>

<div th:if="${msg != null}" class="alert alert-success" th:text="${msg}" role="alert"></div>

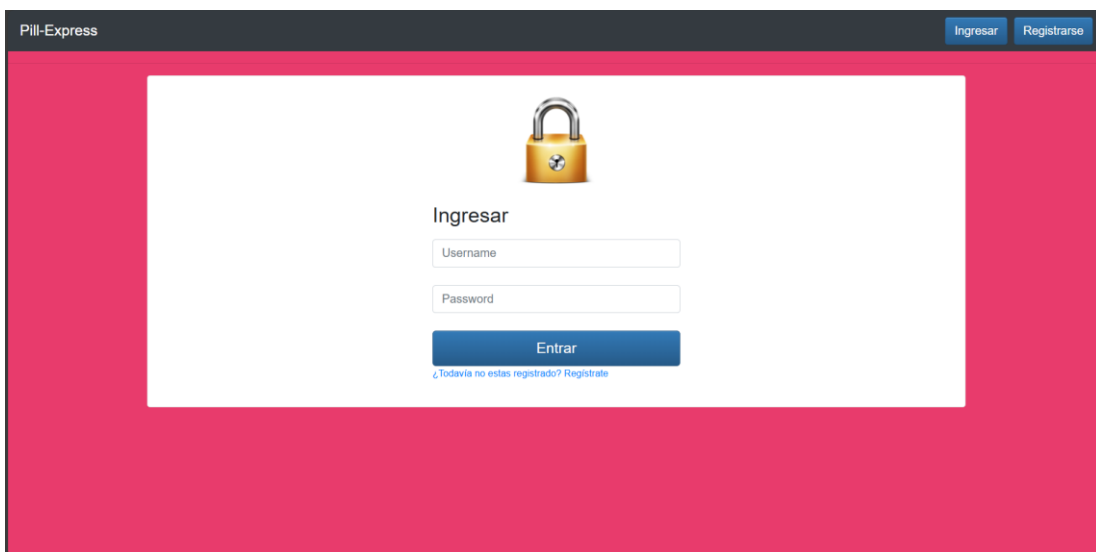
```

### 4. Campos de Entrada y Botón de Envío:

- Estos campos permiten al usuario ingresar su nombre de usuario y contraseña. El botón de envío envía el formulario. El párrafo final incluye un enlace a la página de registro si el usuario no está registrado.


```
<label for="username" class="sr-only">Username</label>
<input type="text" name="username" id="username" class="form-control" placeholder="Username" required autofocus>
<label for="password" class="sr-only">Password</label>
<input type="password" name="password" id="password" class="form-control" placeholder="Password" required></input>
<button class="btn btn-lg btn-primary btn-block" type="submit">Entrar</button>
<p class="text-primary" style="font-size: small">¿Todavía no estás registrado? <a href="@{/signup}">Regístrate</a></p>
</div>
<div class="col-md=4">
```

En el HTML se muestran los diferentes campos a rellenar y el botón de ingreso que llama al método /login.



Pill-Express

Ingresar Registrarse



Ingresar

Username

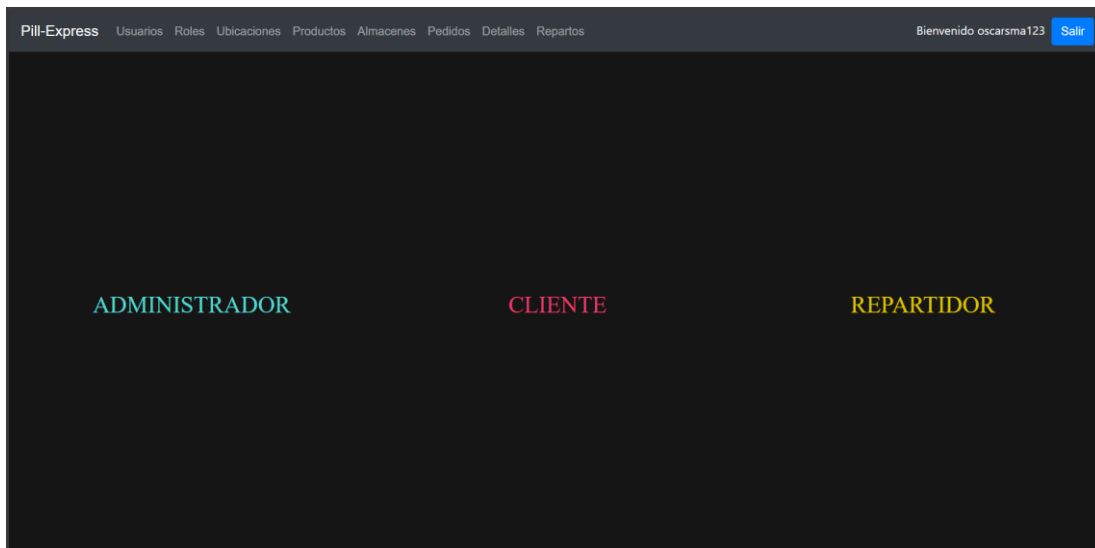
Password

Entrar

¿Todavía no estás registrado? Regístrate

## 7.4 Menú.

Una vez ingresamos a la aplicación observamos el menú con tres opciones Administrador, Cliente y Repartidor; dependiendo del Rol que tenga el usuario solo podrá acceder a un determinado apartado, el usuario con rol de Administrador tiene acceso a todos los campos, el usuario con rol de Repartidor tiene acceso a Cliente y Repartidor y el usuario con rol Cliente solo tiene acceso a Cliente.



Las características del HTML son las siguientes:

1. CSS con las diferentes animaciones y colores de la página.

```
.cliente:hover {
  color: #111;
  transition: color 1s cubic-bezier(0.33, 1, 0.68, 1);
}

.cliente::before {
  content: "";
  position: absolute;
  z-index: -2;
  width: 100%;
  height: 100%;
  top: 0;
  right: 0;
  clip-path: polygon(0% -10%, 100% -20%, 100% -10%, 0% 0%, 0% 130%, 100% 120%, 100% 100%, 0% 110%);
  background-color: #e83b6c;
  transition: clip-path 1s cubic-bezier(0.25, 1, 0.5, 1);
}

.cliente:hover::before {
  clip-path: polygon(0% 10%, 100% 0%, 100% 20%, 0% 30%, 0% 100%, 100% 90%, 100% 70%, 0% 80%);
}

.cliente::after {
  content: "";
  position: absolute;
  z-index: -1;
  width: 5ch; /* Anchura ajustada */
  height: 12ch; /* Altura ajustada */
  top: 50%;
  right: 50%;
  transform: translate(50%, -50%) rotate(10deg) scale(0);
  transition: transform 1s ease;
  background-color: #e83b6c;
}

.cliente:hover::after {
```

2. El cuerpo contiene tres divisiones principales que representan diferentes secciones basadas en los roles de usuario:
  - Administrador: Solo accesible para usuarios con el rol de ADMINISTRADOR.
  - Cliente: Accesible para usuarios con los roles ADMINISTRADOR, REPARTIDOR y CLIENTE.
  - Repartidor: Accesible para usuarios con los roles ADMINISTRADOR y REPARTIDOR.

```

</head>
<header th:insert="fragments/menu:: menu-principal"></header>
<body>
<div class="background-one">
  <div class="link-container" sec:authorize="hasAnyAuthority('ADMINISTRADOR')">
    <a class="admin" th:href="@{/usuarios/index}" style="font-size: 2em; font-family: serif;">ADMINISTRADOR</a>
  </div>
</div>
<div class="background-two link-container" sec:authorize="hasAnyAuthority('ADMINISTRADOR', 'SUPERVISOR', 'USUARIO')">
  <a class="cliente" th:href="@{/tienda}" style="font-size: 2em; font-family: serif;">CLIENTE</a>
</div>
<div class="background-three link-container " sec:authorize="hasAnyAuthority('ADMINISTRADOR', 'SUPERVISOR')">
  <a class="reparto" th:href="@{/repartidor}" style="font-size: 2em; font-family: serif;">REPARTIDOR</a>
</div>

```

## 7.5 Administrador-Usuarios.

UsuariosController.java:

```

4
5 @Controller
6 @RequestMapping("/usuarios")
7 public class UsuariosController {
8
9     // Inyectamos una instancia desde nuestro ApplicationContext
10    @Autowired
11    private IUsuariosService serviceUsuarios;
12
13    /**
14     * Método que muestra la lista de usuarios sin paginación
15     * @param model
16     * @param page
17     * @return
18     */
19    @GetMapping("/index")
20    public String mostrarIndex(Model model) {
21        List<Usuario> lista = serviceUsuarios.buscarRegistrados();
22        model.addAttribute("usuarios", lista);
23        return "usuarios/listUsuarios";
24    }
25
26    /**
27     * Método para eliminar un usuario de la base de datos.
28     * @param idUsuario
29     * @param attributes
30     * @return
31     */
32    @GetMapping("/delete/{id}")
33    public String eliminar(@PathVariable("id") int idUsuario, RedirectAttributes attributes) {
34
35        // Eliminamos el usuario
36        serviceUsuarios.eliminar(idUsuario);
37
38        attributes.addFlashAttribute("msg", "El usuario fue eliminado!.");
39        return "redirect:/usuarios/index";
40    }

```

```

    /**
     * Método para activar un usuario
     * @param idUsuario
     * @param attributes
     * @return
     */
    @GetMapping("/unlock/{id}")
    public String activar(@PathVariable("id") int idUsuario, RedirectAttributes attributes) {
        serviceUsuarios.activar(idUsuario);
        attributes.addFlashAttribute("msg", "El usuario fue activado y ahora tiene acceso al sistema");
        return "redirect:/usuarios/index";
    }

    /**
     * Método para bloquear un usuario
     * @param idUsuario
     * @param attributes
     * @return
     */
    @GetMapping("/lock/{id}")
    public String bloquear(@PathVariable("id") int idUsuario, RedirectAttributes attributes) {
        serviceUsuarios.bloquear(idUsuario);
        attributes.addFlashAttribute("msg", "El usuario fue bloqueado y no tendrá acceso al sistema");
        return "redirect:/usuarios/index";
    }
}

```

#### Modelo Usuarios:

1. Esta clase Usuario representa una entidad en la base de datos y contiene los siguientes atributos y métodos:

- Id: Identificador único del usuario, generado automáticamente.
- Username: Nombre de usuario.
- Nombre: Nombre completo del usuario.
- Email: Dirección de correo electrónico del usuario.
- Password: Contraseña del usuario.
- Estatus: Estado del usuario, que puede ser activo o bloqueado.
- FechaRegistro: Fecha de registro del usuario en la base de datos.
- Perfiles: Lista de perfiles asociados al usuario.

2. Métodos:

- getId(), setId(), getUsername(), setUsername(), getNombre(), setNombre(), getEmail(), setEmail(), getPassword(), setPassword(), getStatus(), setStatus(), getFechaRegistro(), setFechaRegistro(), getPerfiles(), setPerfiles(): Métodos de acceso para los atributos de la clase.
- Agregar(Perfil tempPerfil): Método para agregar un perfil a la lista de perfiles del usuario.
- ToString(): Método que devuelve una representación de cadena de texto del objeto usuario, útil para depuración y registro.

3. Relación ManyToMany:

- La clase Usuario tiene una relación ManyToMany con la clase Perfil, lo que significa que un usuario puede tener múltiples perfiles y viceversa.
- La anotación @ManyToMany especifica esta relación y define la tabla intermedia (UsuarioPerfil) y las claves foráneas (idUser e idPerfil).
- La propiedad fetch = FetchType.EAGER indica que los perfiles asociados al usuario deben cargarse de inmediato cuando se recupera un usuario.

```

18
19 @Entity
20 @Table(name = "Usuarios")
21 public class Usuario {
22
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY) // auto_increment MySQL
25     private Integer id;
26     private String username;
27     private String nombre;
28     private String email;
29     private String password;
30     private Integer estatus;
31     private Date fechaRegistro;
32
33     // Relacion ManyToMany (Un usuario tiene muchos perfiles)
34     // Por defecto Fetch es FetchType.LAZY
35     @ManyToMany(fetch = FetchType.EAGER)
36     @JoinTable(name = "UsuarioPerfil", // tabla intermedia
37         joinColumns = @JoinColumn(name = "idUsuario"), // foreignKey en la tabla de UsuarioPer
38         inverseJoinColumns = @JoinColumn(name = "idPerfil") // foreignKey en la tabla de Usuar
39     )
40     private List<Perfil> perfiles;
41
42     public Integer getId() {
43         return id;
44     }
45
46     public void setId(Integer id) {
47         this.id = id;
48     }
49
50     public String getUsername() {
51         return username;
52     }
53
54     public void setUsername(String username) {
55         this.username = username;
56     }

```

```

58     public String getNombre() {
59         return nombre;
60     }
61
62     public void setNombre(String nombre) {
63         this.nombre = nombre;
64     }
65
66     public String getEmail() {
67         return email;
68     }
69
70     public void setEmail(String email) {
71         this.email = email;
72     }
73
74     public String getPassword() {
75         return password;
76     }
77
78     public void setPassword(String password) {
79         this.password = password;
80     }
81
82     public Integer getEstatus() {
83         return estatus;
84     }
85
86     public void setEstatus(Integer estatus) {
87         this.estatus = estatus;
88     }
89
90     public Date getFechaRegistro() {
91         return fechaRegistro;
92     }
93
94     public void setFechaRegistro(Date fechaRegistro) {
95         this.fechaRegistro = fechaRegistro;
96     }
97
98     public List<Perfil> getPerfiles() {
99         return perfiles;
100     }

```

```

1
2● public void setPerfiles(List<Perfil> perfiles) {
3    this.perfiles = perfiles;
4  }
5
6  // Metodo para agregar perfiles
7● public void agregar(Perfil tempPerfil) {
8    if (perfiles == null) {
9      perfiles = new LinkedList<>();
10   }
11   perfiles.add(tempPerfil);
12 }
13
14● @Override
15 public String toString() {
16   return "Usuario [id=" + id + ", username=" + username + ", nombre=" + nombre + ", email="
17     + ", password=" + password + ", estatus=" + estatus + ", fechaRegistro=" + fechaRe
18     + perfiles + "]\n";
19 }
20
21 }

```

UsuariosRepository.java:

1. Este es el repositorio UsuariosRepository, que extiende la interfaz JpaRepository proporcionada por Spring Data JPA para realizar operaciones CRUD (Create, Read, Update, Delete) en la entidad Usuario.
2. Métodos heredados:
  - FindByUsername(String username): Método heredado de JpaRepository que busca un usuario por su nombre de usuario.
  - FindByFechaRegistroNotNull(): Método que devuelve una lista de usuarios cuya fecha de registro no es nula.
3. Métodos personalizados:
  - Lock(int idUsuario): Método personalizado que utiliza la anotación @Query para actualizar el estado de un usuario que se ha bloqueado (estatus = 0) en la base de datos. Se utiliza la anotación @Modifying para indicar que esta consulta modificará el estado de la base de datos.
  - Unlock(int idUsuario): Método similar al anterior, pero que actualiza el estado de un usuario a activo (estatus = 1).

```

import java.util.List;

public interface UsuariosRepository extends JpaRepository<Usuario, Integer> {
  // Buscar usuario por username
  Usuario findByUsername(String username);
  List<Usuario> findByFechaRegistroNotNull();

  @Modifying
  @Query("UPDATE Usuario u SET u.estatus=0 WHERE u.id = :paramIdUsuario")
  int lock(@Param("paramIdUsuario") int idUsuario);

  @Modifying
  @Query("UPDATE Usuario u SET u.estatus=1 WHERE u.id = :paramIdUsuario")
  int unlock(@Param("paramIdUsuario") int idUsuario);
}

```



IUsuariosService.java:

El archivo IUsuariosService define la interfaz para el servicio relacionado con la gestión de usuarios en la aplicación.

1. Métodos definidos:

- Guardar(Usuario usuario): Método para guardar un nuevo usuario en la base de datos.
- Eliminar(Integer idUsuario): Método para eliminar un usuario existente de la base de datos, dado su ID.
- BuscarTodos(): Método para recuperar una lista de todos los usuarios registrados en la base de datos.
- BuscarRegistrados(): Método para recuperar una lista de usuarios que han sido registrados.
- BuscarPorId(Integer idUsuario): Método para buscar un usuario por su ID en la base de datos.
- BuscarPorUsername(String username): Método para buscar un usuario por su nombre de usuario en la base de datos.
- Bloquear(int idUsuario): Método para bloquear un usuario, cambiando su estado en la base de datos.
- Activar(int idUsuario): Método para activar un usuario, cambiando su estado en la base de datos.

```
package com.tfg.service;

import java.util.List;

public interface IUsuariosService {
    void guardar(Usuario usuario);
    void eliminar(Integer idUsuario);
    List<Usuario> buscarTodos();
    List<Usuario> buscarRegistrados();
    Usuario buscarPorId(Integer idUsuario);
    Usuario buscarPorUsername(String username);
    int bloquear(int idUsuario);
    int activar(int idUsuario);
}
```

UsuariosServiceJPA.java:

1. El archivo UsuariosServiceJpa implementa la interfaz IUsuariosService y proporciona la lógica de negocio para operaciones relacionadas con usuarios utilizando JPA (Java Persistence API).

## 2. Métodos implementados:

- Guardar(Usuario usuario): Guarda un nuevo usuario en la base de datos.
- Eliminar(Integer idUsuario): Elimina un usuario existente de la base de datos por su ID.
- BuscarTodos(): Recupera una lista de todos los usuarios registrados en la base de datos.
- BuscarPorId(Integer idUsuario): Busca un usuario por su ID en la base de datos.
- BuscarPorUsername(String username): Busca un usuario por su nombre de usuario en la base de datos.
- BuscarRegistrados(): Recupera una lista de usuarios que han sido registrados.
- Bloquear(int idUsuario): Bloquea un usuario cambiando su estado en la base de datos.
- Activar(int idUsuario): Activa un usuario cambiando su estado en la base de datos.

## 1. Uso de anotaciones:

- @Service: Indica que la clase es un componente de servicio en Spring.
- @Autowired: Realiza la inyección de dependencias del repositorio de usuarios (UsuariosRepository).
- @Transactional: Define que los métodos marcados con esta anotación deben ejecutarse dentro de una transacción de base de datos.

```
package com.tfg.service.db;

import java.util.List;

@Service
public class UsuariosServiceJpa implements IUsuariosService{

    @Autowired
    private UsuariosRepository usuariosRepo;

    @Override
    public void guardar(Usuario usuario) {
        usuariosRepo.save(usuario);
    }

    @Override
    public void eliminar(Integer idUsuario) {
        usuariosRepo.deleteById(idUsuario);
    }

    @Override
    public List<Usuario> buscarTodos() {
        return usuariosRepo.findAll();
    }

    @Override
    public Usuario buscarPorId(Integer idUsuario) {
        Optional<Usuario> optional = usuariosRepo.findById(idUsuario);
        if (optional.isPresent()) {
            return optional.get();
        }
        return null;
    }

    @Override
    public Usuario buscarPorUsername(String username) {
        return usuariosRepo.findByUsername(username);
    }

    @Override
    public List<Usuario> buscarRegistrados() {
        return usuariosRepo.findByFechaRegistroNotNull();
    }
}
```

```

    @Transactional
    @Override
    public int bloquear(int idUsuario) {
        int rows = usuariosRepo.lock(idUsuario);
        return rows;
    }

    @Transactional
    @Override
    public int activar(int idUsuario) {
        int rows = usuariosRepo.unlock(idUsuario);
        return rows;
    }
}













```

HTML:

1. Tabla de Usuarios: Se muestra una tabla con columnas para el nombre, el nombre de usuario, el correo electrónico y el estado del usuario.
2. Operaciones: Cada fila de la tabla incluye botones para eliminar un usuario y cambiar su estado (bloquear o desbloquear) según su estado actual.
3. Mensajes de Éxito: Se muestra un mensaje de éxito cuando se realiza una acción como eliminar o cambiar el estado de un usuario.

Pill-Express Usuarios Roles Ubicaciones Productos Almacenes Pedidos Detalles Repartos Bienvenido oscarisma123 [Salir](#)

### Listado de Usuarios

Nombre	Username	Email	Estatus	Operaciones
Ivan Tinajero	itinajero	ivanetinajero@gmail.com	Bloqueado	 
Luis Esparza Gomez	luis	luis@tinajero.net	Activo	 
Marisol Salinas Rodarte	marisol	marisol@tinajero.net	Activo	 
Daniel Lopez	diopez	daniel@gmail.com	Activo	 
Miguel Marquez Hernandez	miguel	miguel@tinajero.net	Activo	 
Oscar	oscarsma123	oscarsma123@hotmail.com	Activo	 

© 2024 PILL-EXPRESS, Inc. | WebApp Desarrollada con Spring Boot 2.1.4 | Autores: Oscar Arjona Eciija y Pablo Galvan Tejada | [Privacy](#) · [Terms](#)

## 7.6 Administrador-Roles.

UsuariosPerfilesController.java:

1. El controlador UsuariosPerfilesController gestiona las operaciones relacionadas con la entidad UsuarioPerfil.
2. Mapeo de URLs:
  - /usuarioPerfil/index: Muestra la lista de usuarios perfiles.
  - /usuarioPerfil/indexPaginate: Muestra la lista de usuarios perfiles de forma paginada.
  - /usuarioPerfil/create: Muestra el formulario para crear un nuevo usuario perfil.
  - /usuarioPerfil/save: Guarda un nuevo usuario perfil en la base de datos.
  - /usuarioPerfil/edit/{id}: Muestra el formulario para editar un usuario perfil existente.
  - /usuarioPerfil/delete/{id}: Elimina un usuario perfil existente de la base de datos.
3. Métodos:
  - MostrarIndex(Model model): Muestra la lista de usuarios perfiles.
  - MostrarIndexPaginado(Model model, Pageable page): Muestra la lista de usuarios perfiles de forma paginada.
  - Crear(UsuarioPerfil usuarioPerfil): Muestra el formulario para crear un nuevo usuario perfil.
  - Guardar(UsuarioPerfil usuarioPerfil, BindingResult result, RedirectAttributes attributes): Guarda un nuevo usuario perfil en la base de datos.
  - Editar(int idUsuarioPerfil, Model model): Muestra el formulario para editar un usuario perfil existente.
  - Eliminar(int idUsuarioPerfil, RedirectAttributes attributes): Elimina un usuario perfil existente de la base de datos.
4. Inyección de Dependencias:
  - @Autowired: Realiza la inyección de dependencias del servicio IUsuariosPerfilesService.

```

1 @Controller
2 @RequestMapping(value="/usuarioPerfil")
3 public class UsuariosPerfilesController {
4
5     • @Autowired
6       // @Qualifier("usuarioPerfilServiceJpa")
7       private IUsuariosPerfilesService serviceUsuarioPerfil;
8
9     • @RequestMapping(value="/index", method=RequestMethod.GET)
10      public String mostrarIndex(Model model) {
11          List<UsuarioPerfil> lista = serviceUsuarioPerfil.buscarTodas();
12          model.addAttribute("usuarioPerfil", lista);
13          return "usuarioPerfil/listUsuariosPerfiles";
14      }
15  }

```

```
@RequestMapping(value="/create", method=RequestMethod.GET)
public String crear(UsuarioPerfil usuarioPerfil) {
    return "usuarioPerfil/formUsuariosPerfiles";
}

@RequestMapping(value="/save", method=RequestMethod.POST)
public String guardar(UsuarioPerfil usuarioPerfil, BindingResult result, RedirectAttributes attributes) {
    if (result.hasErrors()) {
        System.out.println("Existieron errores");
        return "usuarioPerfil/formUsuariosPerfiles";
    }

    // Guardamos el objeto usuarioPerfil en la bd
    serviceUsuarioPerfil.guardar(usuarioPerfil);
    attributes.addFlashAttribute("msg", "Los datos de la categoría fueron guardados!");
    return "redirect:/usuarioPerfil/indexPaginate";
}

@GetMapping("/edit/{id}")
public String editar(@PathVariable("id") int idUsuarioPerfil, Model model) {
    UsuarioPerfil usuarioPerfil = serviceUsuarioPerfil.buscarPorId(idUsuarioPerfil);
    model.addAttribute("usuarioPerfil", usuarioPerfil);
    return "usuarioPerfil/formUsuariosPerfiles";
}

@GetMapping("/delete/{id}")
public String eliminar(@PathVariable("id") int idUsuarioPerfil, RedirectAttributes attributes) {
    // Eliminamos la usuarioPerfil.
    serviceUsuarioPerfil.eliminar(idUsuarioPerfil);
    attributes.addFlashAttribute("msg", "La categoría fue eliminada!");
    return "redirect:/usuarioPerfil/indexPaginate";
}
```

#### Modelo-UsuarioPerfil:

1. La clase UsuarioPerfil representa la relación entre un usuario y un perfil en la base de datos. Aquí tienes un resumen de la clase:
2. Anotaciones JPA:
  - @Entity: Indica que la clase es una entidad JPA.
  - @Table(name="UsuarioPerfil"): Especifica el nombre de la tabla en la base de datos.
3. Atributos:
  - Número: Representa el número de identificación del usuario perfil.
  - Usuario: Representa la relación con la entidad Usuario. Utiliza la anotación @OneToOne para indicar que un usuario perfil pertenece a un único usuario.
  - perfil: Representa la relación con la entidad Perfil. Utiliza la anotación @OneToOne para indicar que un usuario perfil tiene un único perfil.
4. Métodos:
  - Getters y setters para los atributos número, usuario y perfil.
  - ToString(): Genera una representación en cadena de texto del objeto UsuarioPerfil.

```

15 @Entity
16 @Table(name="UsuarioPerfil")
17 public class UsuarioPerfil {
18     @Id
19     @GeneratedValue(strategy=GenerationType.IDENTITY)
20     private Integer numero;
21
22     @OneToOne
23     @JoinColumn(name = "idUsuario")
24     private Usuario usuario;
25
26     @OneToOne
27     @JoinColumn(name = "idPerfil")
28     private Perfil perfil;
29
30     public Integer getNumero() {
31         return numero;
32     }
33     public void setNumero(Integer numero) {
34         this.numero = numero;
35     }
36
37     public Usuario getUsuario() {
38         return usuario;
39     }
40
41     public void setUsuario(Usuario usuario) {
42         this.usuario = usuario;
43     }
44
45     public Perfil getPerfil() {
46         return perfil;
47     }
48
49     public void setPerfil(Perfil perfil) {
50         this.perfil = perfil;
51     }
52
53     @Override
54     public String toString() {
55         return "UsuarioPerfil [numero=" + numero + ", usuario=" + usuario + ", perfil=" + perfil + "
56     }

```

UsuariosPerfilesRepository.java:

1. El repositorio UsuariosPerfilesRepository proporciona métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la entidad UsuarioPerfil. Aquí tienes un resumen de la interfaz:
2. Extensión de JpaRepository: La interfaz UsuariosPerfilesRepository extiende JpaRepository<UsuarioPerfil, Integer>, lo que significa que hereda métodos para realizar operaciones CRUD en la entidad UsuarioPerfil. El tipo de entidades UsuarioPerfil, y el tipo de su identificador es Integer.

```

1 package com.tfg.repository;
2
3 import java.util.Optional;
4
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 import com.tfg.model.UsuarioPerfil;
8
9 public interface UsuariosPerfilesRepository extends JpaRepository<UsuarioPerfil, Integer> {}
10
11 }

```

IUsuariosPerfilesService.java:

1. La interfaz IUsuariosPerfilesService define los métodos que pueden ser implementados por diferentes clases de servicio para realizar operaciones relacionadas con la entidad UsuarioPerfil. Aquí tienes un resumen de la interfaz:
  - Guardar(UsuarioPerfil usuarioPerfil): Este método se utiliza para guardar un objeto UsuarioPerfil en la base de datos.
  - BuscarTodas(): Este método devuelve una lista de todos los objetos UsuarioPerfil almacenados en la base de datos.
  - BuscarPorId(Integer idUsuarioPerfil): Este método busca un objeto UsuarioPerfil por su ID y lo devuelve.
  - Eliminar(Integer idUsuarioPerfil): Este método elimina un objeto UsuarioPerfil de la base de datos según su ID.
  - BuscarTodas(Pageable page): Este método devuelve una página de objetos UsuarioPerfil paginados, lo que permite la paginación de resultados.

```
1 package com.tfg.service;
2
3 import java.util.List;
4
5 public interface IUsuariosPerfilesService {
6     void guardar(UsuarioPerfil usuarioPerfil);
7     List<UsuarioPerfil> buscarTodas();
8     UsuarioPerfil buscarPorId(Integer idUsuarioPerfil);
9     void eliminar(Integer idUsuarioPerfil);
10    Page<UsuarioPerfil> buscarTodas(Pageable page);
11 }
12
13
14
15
16
17
18
```

UsuariosPerfilesServiceJpa.java:

1. La clase UsuariosPerfilesServiceJpa implementa la interfaz IUsuariosPerfilesService y proporciona la lógica para realizar operaciones relacionadas con la entidad UsuarioPerfil. Aquí tienes un resumen de la implementación:
  - Guardar(UsuarioPerfil usuarioPerfil): Este método guarda un objeto UsuarioPerfil en la base de datos utilizando el repositorio UsuariosPerfilesRepository.
  - BuscarTodas(): Este método devuelve una lista de todos los objetos UsuarioPerfil almacenados en la base de datos utilizando el método findAll() del repositorio.
  - BuscarPorId(Integer idUsuarioPerfil): Este método busca un objeto UsuarioPerfil por su ID utilizando el método findById() del repositorio.

- **Eliminar(Integer idUsuarioPerfil):** Este método elimina un objeto **UsuarioPerfil** de la base de datos según su ID utilizando el método **deleteById()** del repositorio.
- **BuscarTodas(Pageable page):** Este método devuelve una página de objetos **UsuarioPerfil** paginados utilizando el método **findAll()** del repositorio con un parámetro **Pageable**. Esto permite la paginación de resultados.

```
@Service
@Primary
public class UsuariosPerfilesServiceJpa implements IUsuariosPerfilesService {

    @Autowired
    private UsuariosPerfilesRepository usuarioPerfilRepo;

    public void guardar(UsuarioPerfil usuarioPerfil) {
        usuarioPerfilRepo.save(usuarioPerfil);
    }

    public List<UsuarioPerfil> buscarTodas() {
        return usuarioPerfilRepo.findAll();
    }

    public UsuarioPerfil buscarPorId(Integer idUsuarioPerfil) {
        Optional<UsuarioPerfil> optional = usuarioPerfilRepo.findById(idUsuarioPerfil);
        if (optional.isPresent()) {
            return optional.get();
        }
        return null;
    }

    public void eliminar(Integer idUsuarioPerfil) {
        usuarioPerfilRepo.deleteById(idUsuarioPerfil);
    }

    @Override
    public Page<UsuarioPerfil> buscarTodas(Pageable page) {
        return usuarioPerfilRepo.findAll(page);
    }
}
```

HTML de la lista, en él se encuentra el select de todos los Roles y las operaciones de modificar:

Pill-Express

Usuarios

Roles

Ubicaciones

Productos

Almacenes

Pedidos

Detalles

Reportos

Bienvenido oscarma123

Salir

Listado de Roles

Rol 1 = Repartidor

Rol 2 = Administrador

Rol 3 = Cliente

Número	Id Usuario	Rol	Operaciones
1	1	1	
2	1	3	
3	2	1	
4	3	2	
5	5	3	

Anterior

Siguiente

© 2024 PILL-EXPRESS, Inc. | WebApp Desarrollada con Spring Boot 2.1.4 | Autores: Oscar Arjona Eciija y Pablo Galvan Tejada | - Privacy - Terms



HTML del formulario, en él se encuentra el formulario para modificar los roles:

The screenshot shows a web application interface for 'Pill-Express'. The top navigation bar includes links for 'Usuarios', 'Roles', 'Ubicaciones', 'Productos', 'Almacenes', 'Pedidos', 'Detalles', and 'Repartos'. The user is logged in as 'oscarsma123' and can click 'Salir'. The main content area features a form titled 'Cambiar Roles de Usuario'. Inside the form, there is a legend: 'Rol 1 = Repartidor | Rol 2 = Administrador | Rol 3 = Cliente'. Below this, a text input field contains the value '1'. At the bottom of the form is a 'Guardar' button. The footer contains copyright information for 2024, mentions the use of Spring Boot 2.1.4, lists the authors, and provides links for 'Privacy' and 'Terms'.

Pill-Express Usuarios Roles Ubicaciones Productos Almacenes Pedidos Detalles Repartos Bienvenido oscarsma123 Salir

**Cambiar Roles de Usuario**

Rol 1 = Repartidor | Rol 2 = Administrador | Rol 3 = Cliente

1

Guardar

© 2024 PILL-EXPRESS, Inc. | WebApp Desarrollada con Spring Boot 2.1.4 | Autores: Oscar Arjona Eclja y Pablo Galvan Tejada | · [Privacy](#) · [Terms](#)

## 7.7 Administrador-Ubicaciones.

UbicacionesController.java:

1. El controlador `UbicacionesController` gestiona las solicitudes relacionadas con las ubicaciones. Aquí está un resumen de sus métodos:
  - `MostrarIndex(Model model)`: Este método maneja la solicitud GET para mostrar todas las ubicaciones. Obtiene una lista de todas las ubicaciones utilizando el servicio `IUbicacionesService` y las agrega al modelo con el nombre "ubicaciones". Luego devuelve la vista "ubicaciones/listUbicaciones".
  - `MostrarIndexPaginado(Model model, Pageable page)`: Este método maneja la solicitud GET para mostrar las ubicaciones paginadas. Obtiene una página de ubicaciones utilizando el servicio `IUbicacionesService` y las agrega al modelo con el nombre "ubicaciones". Luego devuelve la vista "ubicaciones/listUbicaciones".
  - `Crear(Ubicación ubicación)`: Este método maneja la solicitud GET para mostrar el formulario de creación de una nueva ubicación. Devuelve la vista "ubicaciones/formUbicaciones".
  - `Guardar(Ubicación ubicación, BindingResult result, RedirectAttributes attributes)`: Este método maneja la solicitud POST para guardar una nueva ubicación. Primero valida si hay errores de validación en el objeto `Ubicacion`. Si no hay errores, guarda la ubicación utilizando el servicio `IUbicacionesService` y redirige a la página de ubicaciones paginadas. Si hay errores, vuelve a mostrar el formulario de creación con los mensajes de error.
  - `Editar(int idUbicación, Model model)`: Este método maneja la solicitud GET para mostrar el formulario de edición de una ubicación existente. Obtiene la ubicación por su ID utilizando el servicio `IUbicacionesService`, la agrega al modelo y devuelve la vista "ubicaciones/formUbicaciones".
  - `Eliminar(int idUbicación, RedirectAttributes attributes)`: Este método maneja la solicitud GET para eliminar una ubicación existente por su ID. Utiliza el servicio `IUbicacionesService` para eliminar la ubicación y luego redirige a la página de ubicaciones paginadas con un mensaje flash.

```
@Controller
@RequestMapping(value="/ubicaciones")
public class UbicacionesController {

    @Autowired
    // @Qualifier("ubicacionesServiceJpa")
    private IUbicacionesService serviceUbicacion;

    @RequestMapping(value="/index", method=RequestMethod.GET)
    public String mostrarIndex(Model model) {
        List<Ubicacion> lista = serviceUbicacion.buscarTodas();
        model.addAttribute("ubicaciones", lista);
        return "ubicaciones/listUbicaciones";
    }

    @GetMapping("/indexPaginate")
    public String mostrarIndexPaginado(Model model, Pageable page) {
        Page<Ubicacion> lista = serviceUbicacion.buscarTodas(page);
        model.addAttribute("ubicaciones", lista);
        return "ubicaciones/listUbicaciones";
    }
}
```

#### Modelo Ubicación:

1. La clase Ubicación representa una entidad en la base de datos que almacena información sobre ubicaciones. Aquí tienes un resumen de sus atributos y métodos:
2. Atributos:
  - Id: Identificador único de la ubicación.
  - Lat: Latitud de la ubicación.
  - Lng: Longitud de la ubicación.
  - Nombre: Nombre descriptivo de la ubicación.
3. Métodos:
  - GetId(): Retorna el ID de la ubicación.
  - SetId(Integer id): Establece el ID de la ubicación.
  - GetLat(): Retorna la latitud de la ubicación.
  - SetLat(Double lat): Establece la latitud de la ubicación.
  - GetLng(): Retorna la longitud de la ubicación.
  - SetLng(Double lng): Establece la longitud de la ubicación.
  - GetNombre(): Retorna el nombre de la ubicación.
  - SetNombre(String nombre): Establece el nombre de la ubicación.
  - ToString(): Retorna una representación en cadena de la ubicación, mostrando su ID, latitud, longitud y nombre.

```
@Entity
@Table(name="Ubicaciones")
public class Ubicacion {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private Double lat;
    private Double lng;
    private String nombre;
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }

    public Double getLat() {
        return lat;
    }
    public void setLat(Double lat) {
        this.lat = lat;
    }
    public Double getLng() {
        return lng;
    }
}
```

```
    public void setLng(Double lng) {
        this.lng = lng;
    }

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

    @Override
    public String toString() {
        return "Almacen [id=" + id + ", lat=" + lat + ", lng=" + lng + ", nombre=" + nombre + "]\n";
    }
}
```

UbicacionesRepository.java:

1. El repositorio UbicacionesRepository proporciona métodos para interactuar con la entidad Ubicación en la base de datos. Aquí tienes un resumen de su funcionalidad:
  - Métodos heredados de JpaRepository:
  - Save(T entity): Guarda una ubicación en la base de datos.
  - findById(ID id): Busca una ubicación por su ID.
  - findAll(): Retorna todas las ubicaciones en la base de datos.
  - deleteById(ID id): Elimina una ubicación por su ID.

```
1 package com.tfg.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import com.tfg.model.Ubicacion;
6
7 public interface UbicacionesRepository extends JpaRepository<Ubicacion,Integer>{
8
9 }
10
```

IUbicacionesService.java:

1. La interfaz IUbicacionesService define los métodos que deben ser implementados por las clases de servicio que gestionan las ubicaciones en la aplicación. Aquí tienes un resumen de los métodos definidos en esta interfaz:
  - Guardar(Ubicación ubicación): Método para guardar una nueva ubicación en la base de datos.
  - BuscarTodas(): Método para buscar todas las ubicaciones en la base de datos y retornarlas como una lista.
  - BuscarPorId(Integer idUbicación): Método para buscar una ubicación por su ID.
  - Eliminar(Integer idUbicación): Método para eliminar una ubicación de la base de datos por su ID.
  - BuscarTodas(Pageable page): Método para buscar todas las ubicaciones en la base de datos y retornarlas paginadas utilizando un objeto Pageable.

```
1 package com.tfg.service;
2
3 import java.util.List;
4
5 public interface IUbicacionesService {
6     void guardar(Ubicacion ubicacion);
7     List<Ubicacion> buscarTodas();
8     Ubicacion buscarPorId(Integer idUbicacion);
9     void eliminar(Integer idUbicacion);
10    Page<Ubicacion> buscarTodas(Pageable page);
11 }
```

UbicacionesServiceJpa.java:

1. Esta clase UbicacionesServiceJpa implementa la interfaz IUbicacionesService para proporcionar la lógica de negocio relacionada con las ubicaciones en la aplicación. Aquí está un resumen de su funcionalidad:
  - Guardar(Ubicación ubicación): Este método guarda una nueva ubicación en la base de datos utilizando el repositorio UbicacionesRepository.
  - BuscarTodas(): Devuelve una lista de todas las ubicaciones en la base de datos utilizando el método findAll() del repositorio UbicacionesRepository.
  - BuscarPorId(Integer idUbicación): Busca una ubicación por su ID utilizando el método findById() del repositorio UbicacionesRepository.
  - Eliminar(Integer idUbicación): Elimina una ubicación de la base de datos por su ID utilizando el método deleteById() del repositorio UbicacionesRepository.

- BuscarTodas(Pageable page): Devuelve una página de ubicaciones paginadas utilizando el método findAll() del repositorio UbicacionesRepository con un parámetro Pageable.

```
@Service
@Primary
public class UbicacionesServiceJpa implements IUbicacionesService {

    @Autowired
    private UbicacionesRepository ubicacionRepo;

    public void guardar(Ubicacion ubicacion) {
        ubicacionRepo.save(ubicacion);
    }

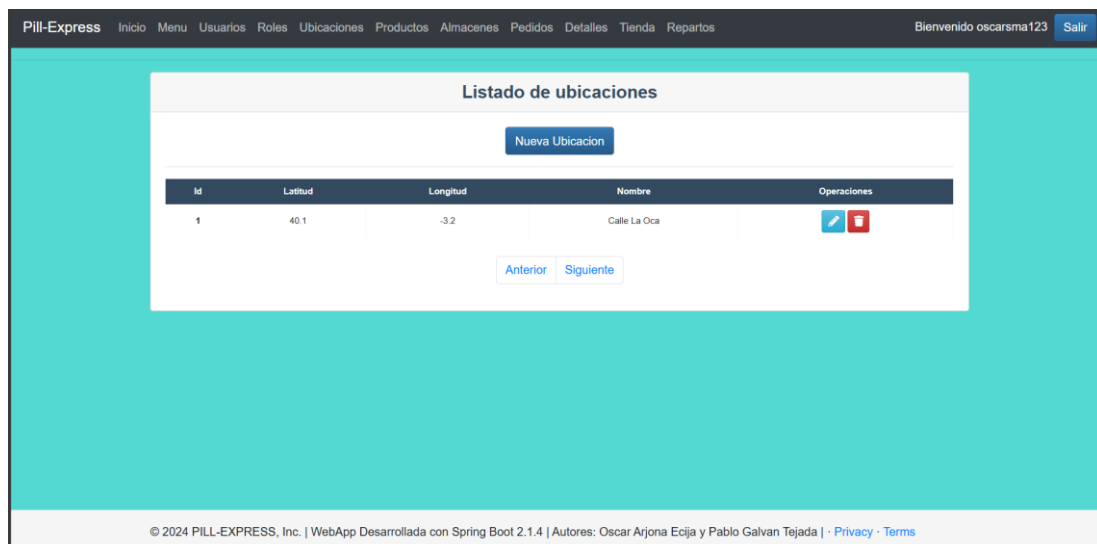
    public List<Ubicacion> buscarTodas() {
        return ubicacionRepo.findAll();
    }
}
```

```
public Ubicacion buscarPorId(Integer idUbicacion) {
    Optional<Ubicacion> optional = ubicacionRepo.findById(idUbicacion);
    if (optional.isPresent()) {
        return optional.get();
    }
    return null;
}

public void eliminar(Integer idUbicacion) {
    ubicacionRepo.deleteById(idUbicacion);
}

@Override
public Page<Ubicacion> buscarTodas(Pageable page) {
    return ubicacionRepo.findAll(page);
}
```

En el HTML listUbicaciones se encuentra la lista con el CRUD de ubicaciones:

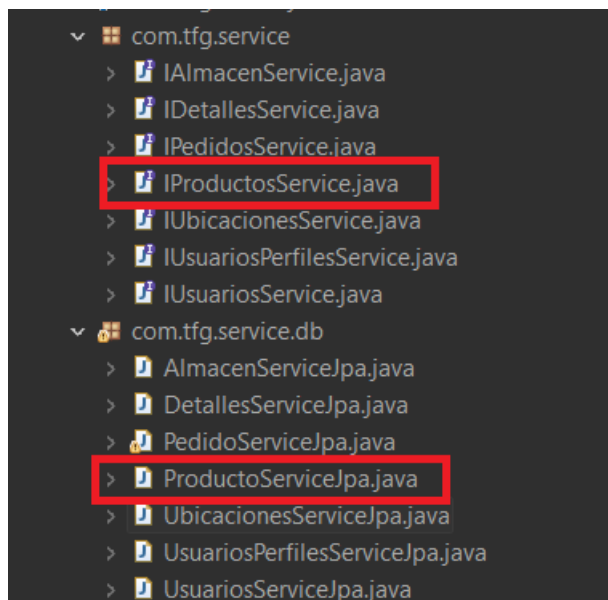
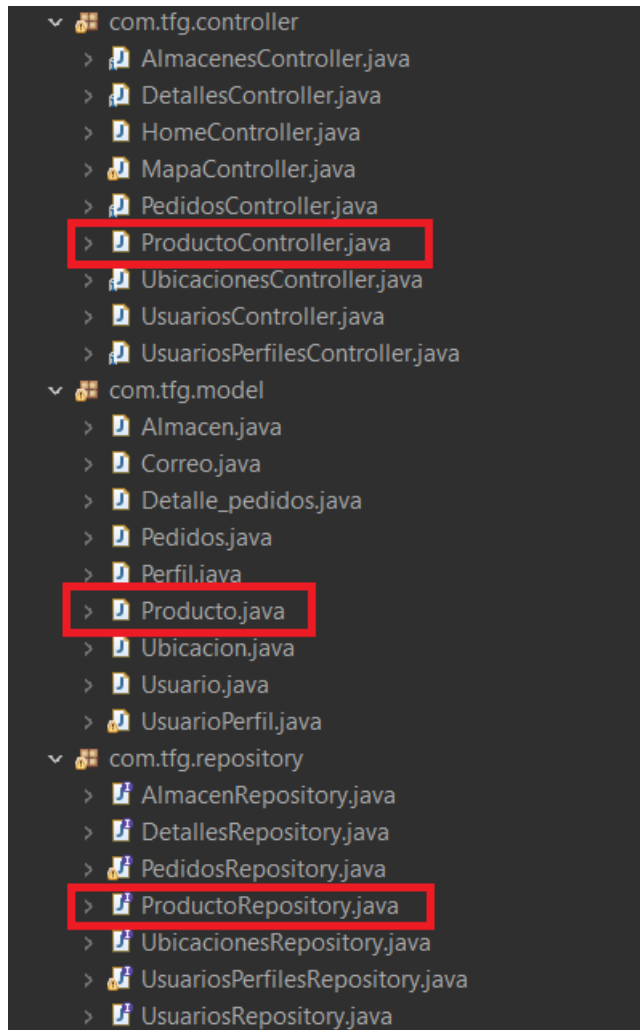


En el HTML formUbicaciones se encuentra el formulario para modificar las ubicaciones existentes:

The screenshot displays the Pill-Express web application interface. At the top, a dark navigation bar contains the application name 'Pill-Express' and a series of menu items: Inicio, Menu, Usuarios, Roles, Ubicaciones, Productos, Almacenes, Pedidos, Detalles, Tienda, and Repartos. On the right side of this bar, it says 'Bienvenido oscarama123' next to a blue 'Salir' button. The main content area has a light blue background. Centered in this area is a white form titled 'Datos de ubicacionistradores'. The form contains three input fields: 'Latitud' with the value '40.1', 'Longitud' with the value '-3.2', and 'Nombre' with the value 'Calle La Oca'. Below these fields is a blue 'Guardar' button.

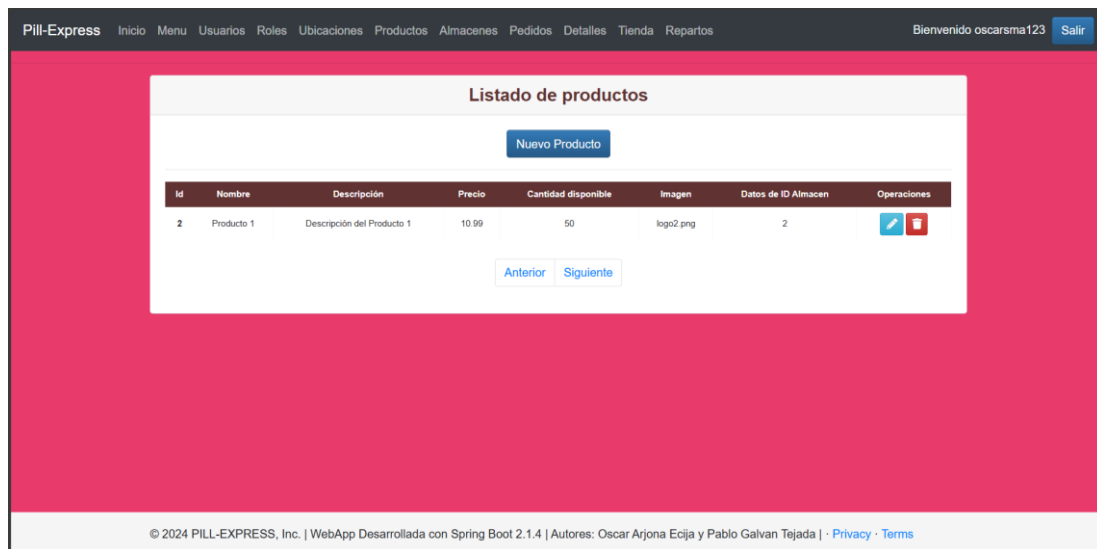
## 7.8 Administrador-Productos:

Sigue la misma lógica del Controller, el Repository, el Iservice y el ServiceJpa que, en el anterior modelo, salvo que tiene diferencias en los atributos del modelo.

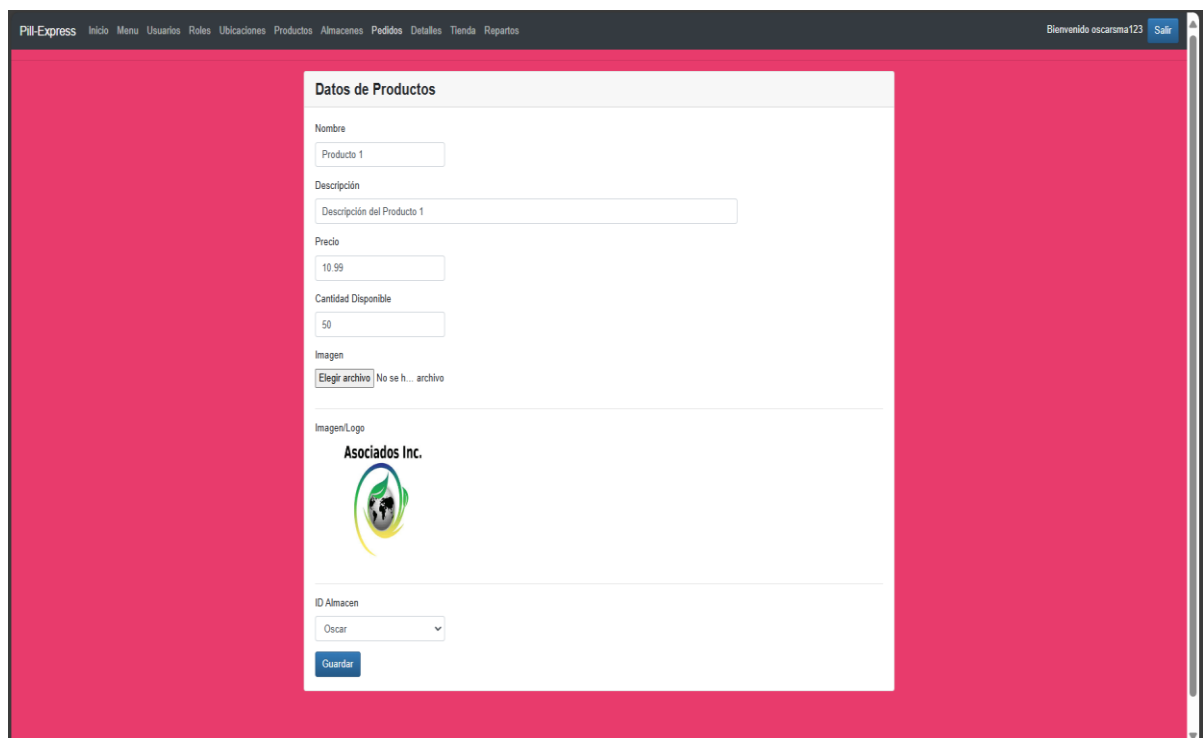


En el HTML listProductos se muestra los atributos de productos con su CRUD correspondiente:



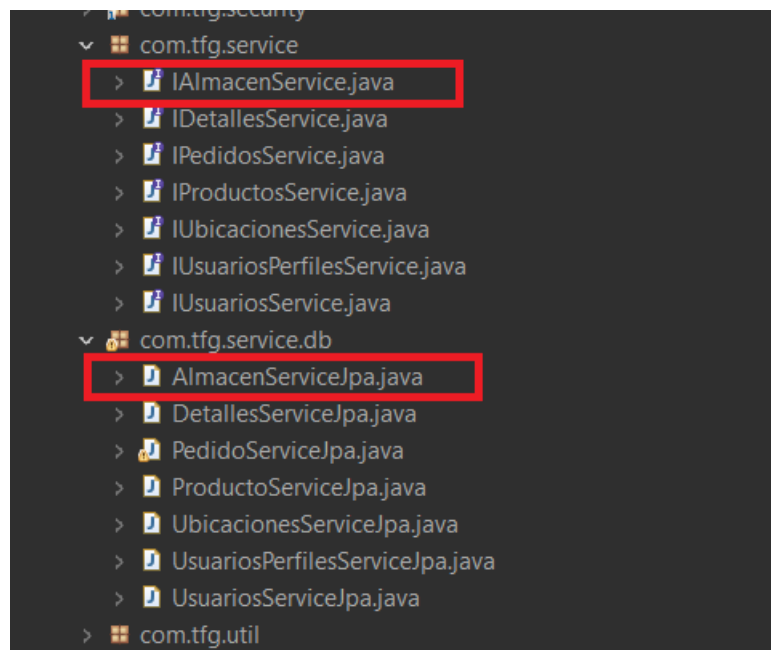
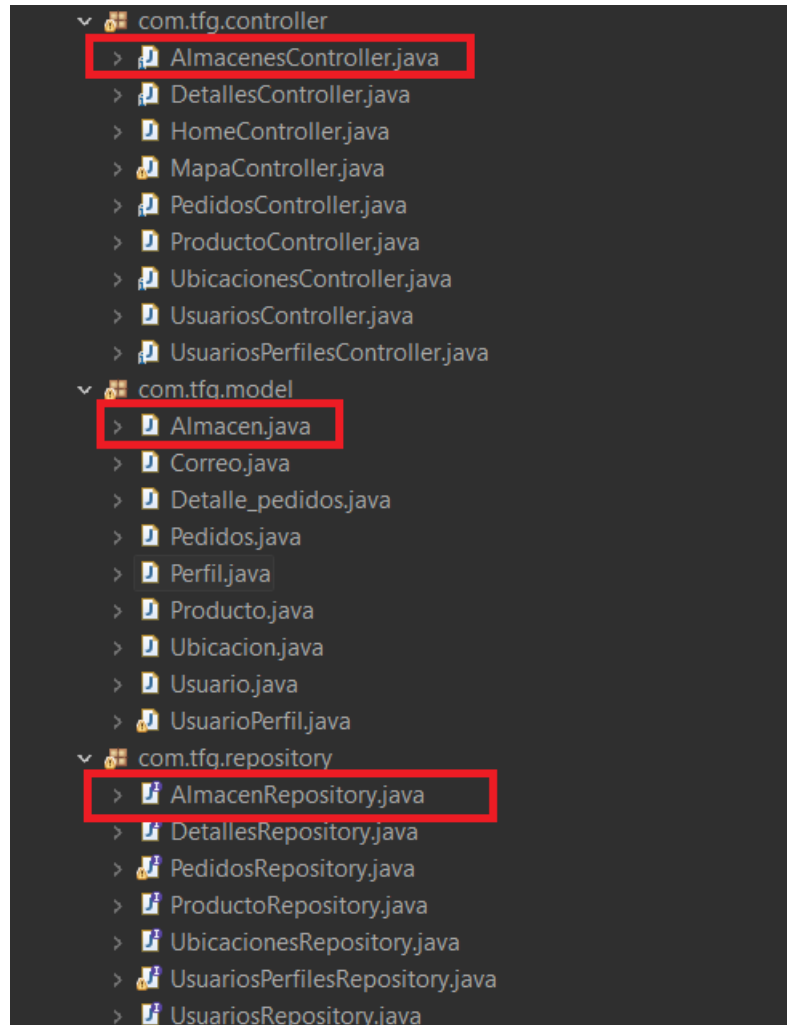


En el HTML formProductos se muestra el formulario para modificar o crear un producto y destaca porque hay un select para elegir el id de almacén y un buscador de archivos para seleccionar la imagen del producto correspondiente:

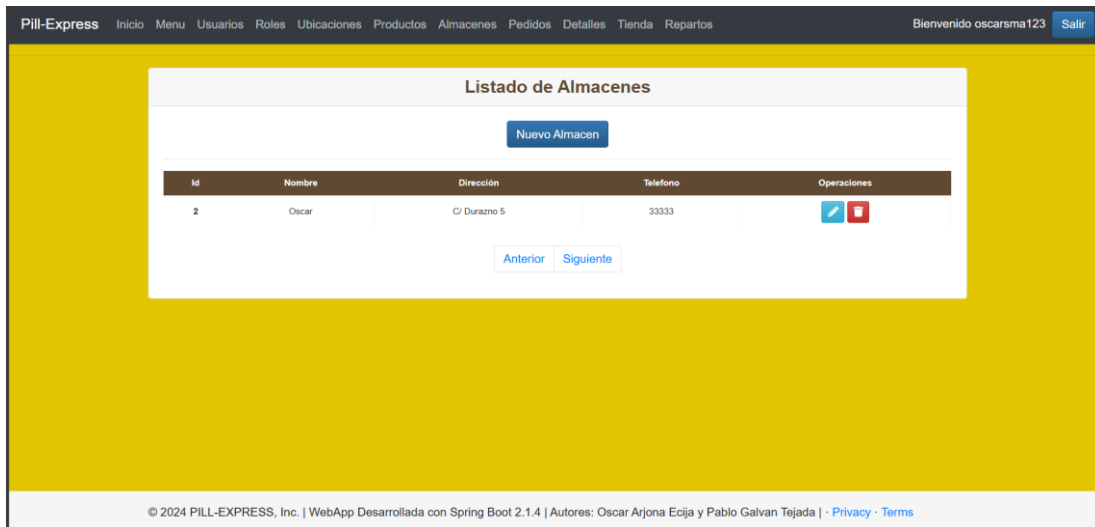


## 7.9 Administrador-Almacenes

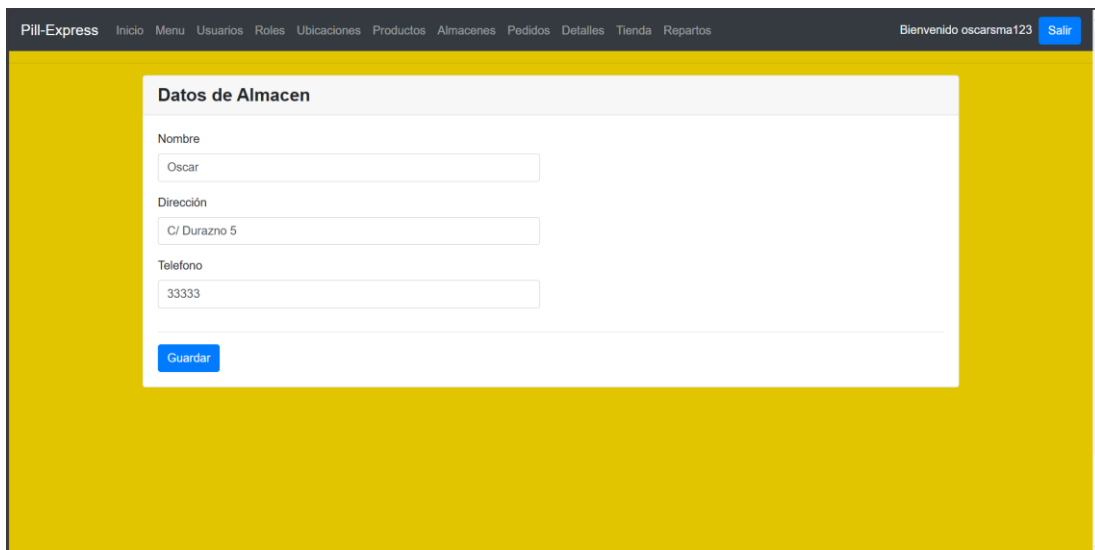
Se emplea la misma lógica del Controller, el Repository, el Iservice y el ServiceJpa que, en Ubicaciones o Productos, salvo que tiene diferencias en los atributos del modelo.



En el HTML listAlmacenes se muestra los atributos de productos con su CRUD correspondiente:

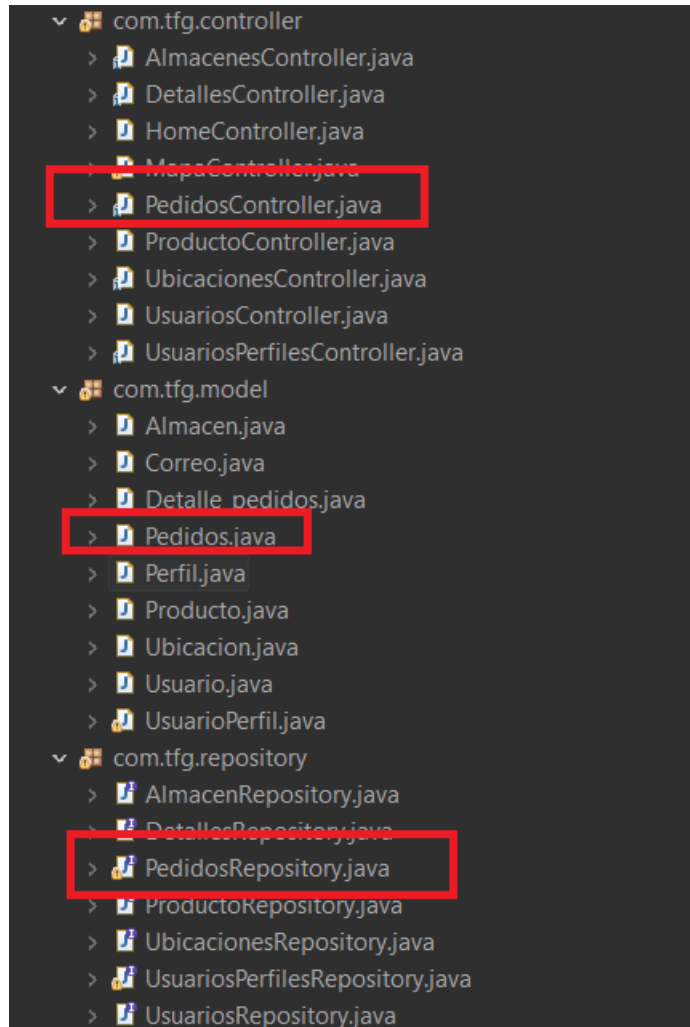


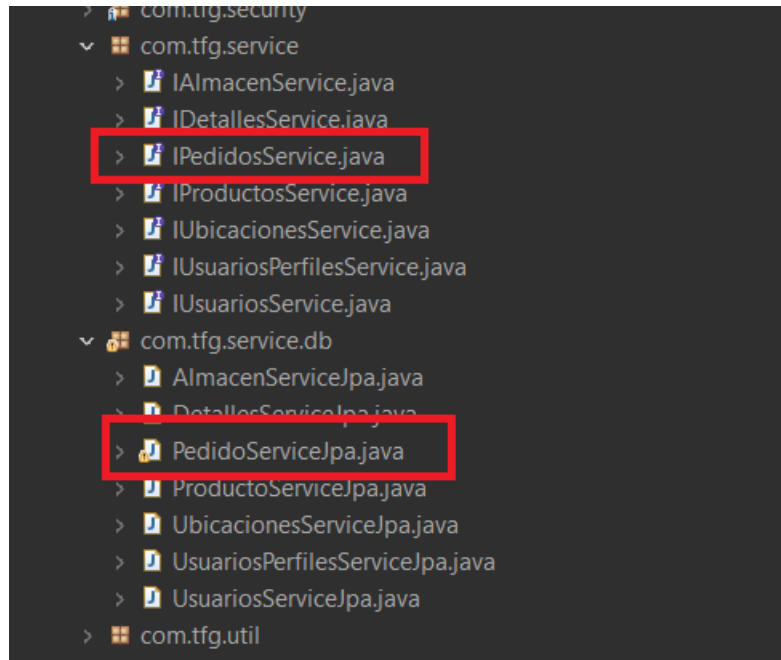
En el HTML formAlmacenes se muestra el formulario para crear o modificar los almacenes de la aplicación:



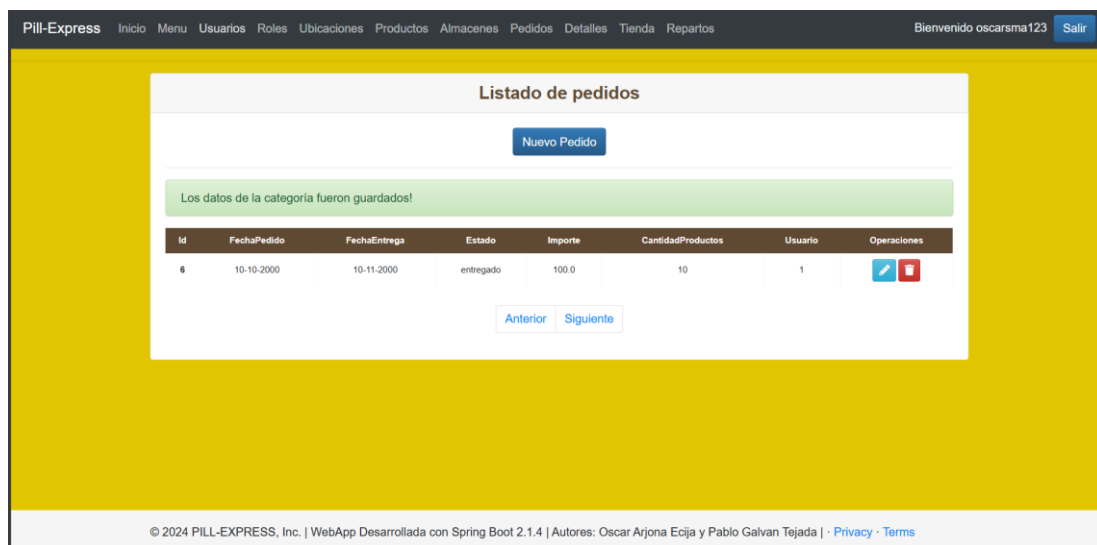
### 7.10 Administrador-Pedidos.

Se utiliza la misma lógica del Controller, el Repository, el Iservice y el ServiceIpa que, en Almacenes o Productos, salvo que tiene diferencias en los atributos del modelo.

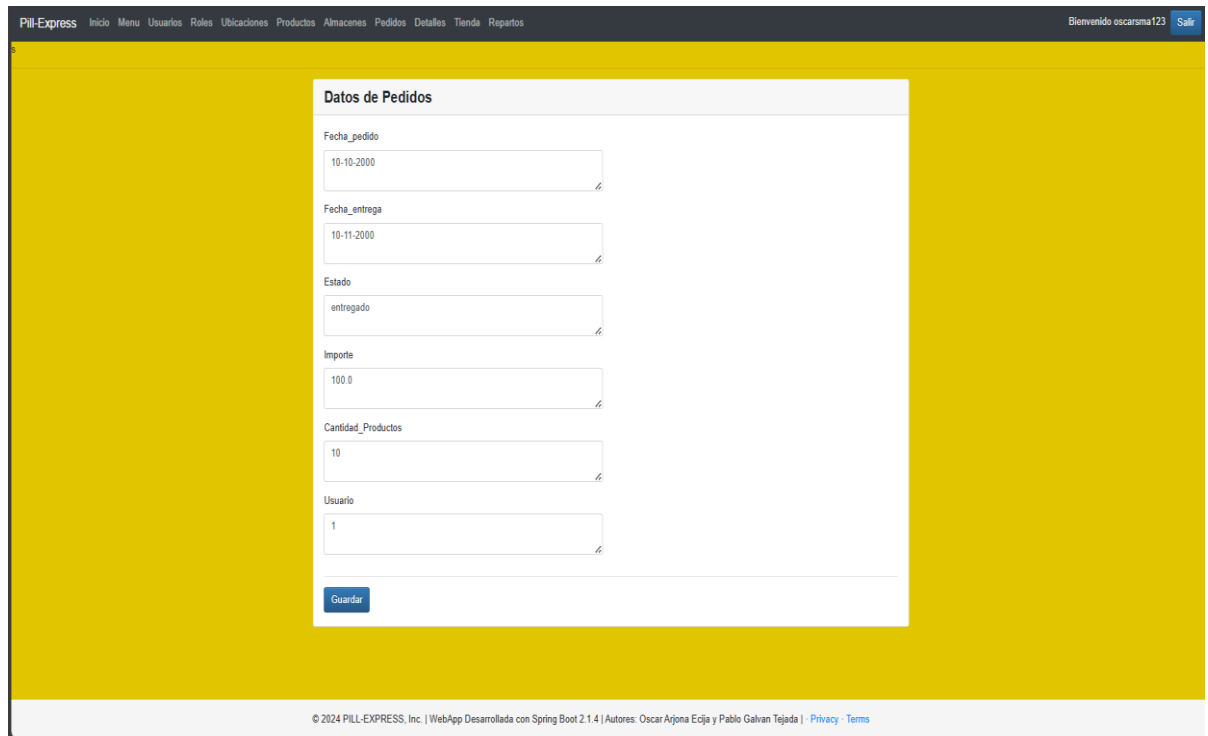




En el HTML listPedidos se muestra los atributos de productos con su CRUD correspondiente:



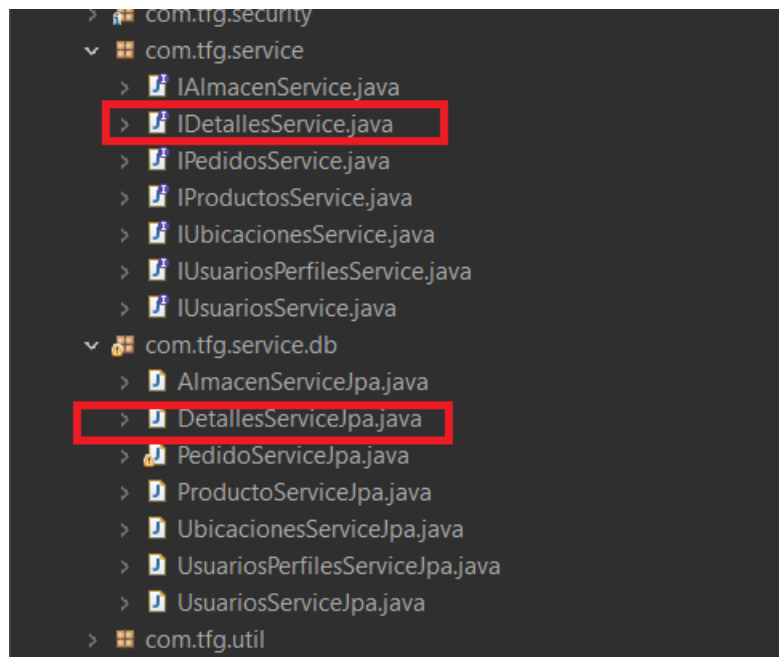
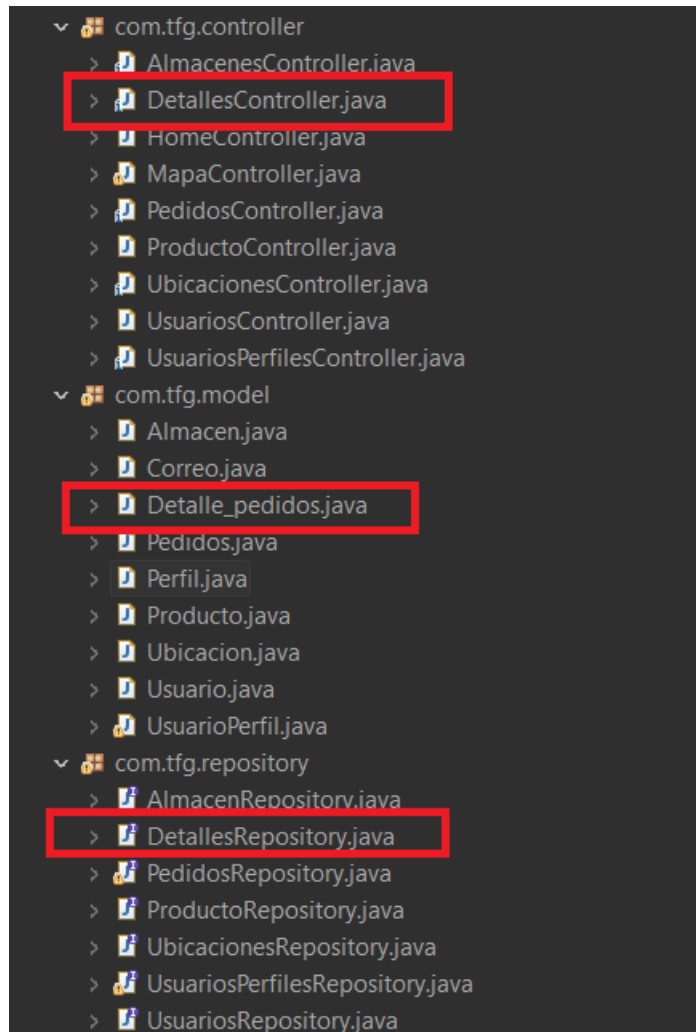
En el HTML formPedidos se muestra el formulario para crear o modificar los pedidos de la aplicación:



The screenshot displays the 'Pill-Express' web application interface. At the top, a navigation bar includes links for Inicio, Menu, Usuarios, Roles, Ubicaciones, Productos, Almacenes, Pedidos, Detalles, Tienda, and Reportes. The user is logged in as 'oscarma123'. The main content area features a yellow background with a white form titled 'Datos de Pedidos'. The form contains the following fields: 'Fecha\_pedido' (10-10-2000), 'Fecha\_entrega' (10-11-2000), 'Estado' (entregado), 'Importe' (100.0), 'Cantidad\_Productos' (10), and 'Usuario' (1). A 'Guardar' button is located at the bottom of the form. The footer contains copyright information for Pill-Express, Inc. and mentions the use of Spring Boot 2.1.4.

### 7.11 Administrador-Detalles Pedidos.

Se basa en la misma lógica del Controller, el Repository, el Iservice y el ServiceJpa que Pedidos, salvo que tiene diferencias en los atributos del modelo.



En el HTML listDetallesPedidos se muestra los atributos de productos con su CRUD correspondiente:

Pill-Express Inicio Menu Usuarios Roles Ubicaciones Productos Almacenes Pedidos Detalles Tienda Repartos Bienvenido oscarma123 [Salir](#)

### Listado de DetallePedidos

[Nuevo Detalle de pedidos](#)

Id	Producto	Cantidad	Precio	Pedidos	Operaciones
1	Producto 1	1	10.99	7	<a href="#">Editar</a> <a href="#">Eliminar</a>
2	Producto 1	1	10.99	8	<a href="#">Editar</a> <a href="#">Eliminar</a>
3	Producto 1	1	10.99	8	<a href="#">Editar</a> <a href="#">Eliminar</a>
4	Producto 1	1	10.99	8	<a href="#">Editar</a> <a href="#">Eliminar</a>
5	Producto 1	1	10.99	9	<a href="#">Editar</a> <a href="#">Eliminar</a>

[Anterior](#) [Siguiente](#)

© 2024 PILL-EXPRESS, Inc. | WebApp Desarrollada con Spring Boot 2.1.4 | Autores: Oscar Arjona Eoija y Pablo Galvan Tejada | [Privacy](#) [Terms](#)

En el HTML formDetallesPedidos se muestra el formulario para crear o modificar los almacenes de la aplicación:

Pill-Express Inicio Menu Usuarios Roles Ubicaciones Productos Almacenes Pedidos Detalles Tienda Repartos Bienvenido oscarma123 [Salir](#)

### Datos de DetallePedidos

Producto

Cantidad

Precio

Pedido

[Guardar](#)



## 7.12 Tienda.

En el HomeController.java encontramos el @GetMapping que permite mostrar los datos presentes en la tienda:

```
@GetMapping("/tienda")
public String mostrarTabla(Model model) {
    List<Producto> lista = serviceProductos.buscarTodas();
    model.addAttribute("productos", lista);
    return "tienda";
}
```

Aparte de eso, encontramos parte de su funcionamiento en el HomeController.java:

### 1. Anotación @GetMapping("/hacerCompra"):

Esta anotación indica que el método hacerCompra responderá a las solicitudes HTTP GET que lleguen a la ruta /hacerCompra.

### 2. Parámetros del Método:

- @RequestParam("productIds") List<Integer> productIds: Lista de IDs de los productos que se van a comprar.
- @RequestParam("importe") Double importe: El importe total de la compra.
- @RequestParam("totalProductos") Integer total: El número total de productos comprados.
- @RequestParam("nombreUsuario") String nombre: El nombre del usuario que está realizando la compra.

### 3. Model model: El modelo que se usará para pasar datos a la vista.

### 4. Bloque try:

- El bloque try se utiliza para manejar cualquier excepción que pueda ocurrir durante el proceso de compra.

### 5. Creación del Pedido:

- Pedidos pedidos = new Pedidos();: Crea una nueva instancia de Pedidos.
- Se establecen varios atributos del pedido, como la fecha del pedido, la fecha de entrega, el estado, el importe y la cantidad de productos.

### 6. Asignación del Usuario al Pedido:

- Usuario username = serviceUsuarios.buscarPorUsername(nombre);: Busca el usuario por su nombre.
- Pedidos.setUsuario(username);: Asigna el usuario al pedido.

### 7. Guardar el Pedido:

- `ServicePedidos.guardar(pedidos);` Guarda el pedido en la base de datos para obtener su ID.

#### 8. Procesamiento de los Productos:

- Un bucle `for` recorre cada `productoid` en `productoids`.
- `Producto producto = serviceProductos.buscarPorId(productoid);` Busca el producto por su ID.
- `Detalle_pedidos detalle = new Detalle_pedidos();` Crea una nueva instancia de `Detalle_pedidos`.
- Se establecen varios atributos del detalle del pedido, como el nombre del producto, la cantidad (asumida como 1), el precio y el pedido al que pertenece.
- `ServiceDetalles.guardar(detalle);` Guarda el detalle del pedido en la base de datos.
- `Producto.setCantidad_disponible(producto.getCantidad_disponible() - 1);` Actualiza la cantidad disponible del producto.
- `ServiceProductos.guardar(producto);` Guarda el producto actualizado.

#### 9. Mensaje de Éxito:

- `Model.addAttribute("mensaje", "Compra realizada con éxito");` Agrega un mensaje de éxito al modelo.
- `Return "compraConfirmada";` Retorna el nombre de la vista de confirmación de compra.

#### 10. Manejo de Excepciones:

- Si ocurre cualquier excepción durante el proceso, se captura en el bloque `catch`.
- `E.printStackTrace();` Imprime la traza del error para depuración.
- `Model.addAttribute("mensaje", "Hubo un error al procesar la compra. Por favor, inténtelo de nuevo.");` Agrega un mensaje de error al modelo.
- `Return "error";` Retorna el nombre de la vista de error.

```
@GetMapping("/hacerCompra")
public String hacerCompra(@RequestParam("productoIds") List<Integer> productoIds,
    @RequestParam("importe") Double importe,
    @RequestParam("totalProductos") Integer total,
    @RequestParam("nombreUsuario") String nombre,
    Model model) {

    try {
        // Crear un nuevo pedido
        Pedidos pedidos = new Pedidos();
        pedidos.setFecha_pedido(new Date());
        pedidos.setFecha_entrega(new Date());
        pedidos.setEstado("pendiente");
        pedidos.setImporte(importe);
        pedidos.setCantidad_productos(total);

        Usuario username = serviceUsuarios.buscarPorUsername(nombre);
        pedidos.setUsuario(username);

        // Guardar el pedido para obtener su ID
        servicePedidos.guardar(pedidos);

        for (Integer productoId : productoIds) {
            Producto producto = serviceProductos.buscarPorId(productoId);
            Detalle_pedidos detalle = new Detalle_pedidos();
            detalle.setProducto(producto.getNombre());
            detalle.setCantidad(1);
            detalle.setPrecio(producto.getPrecio());
            detalle.setPedido(pedidos);
            serviceDetalles.guardar(detalle);

            producto.setCantidad_disponible(producto.getCantidad_disponible() - 1);
            serviceProductos.guardar(producto);
        }

        model.addAttribute("mensaje", "Compra realizada con éxito");
        return "compraConfirmada";
    } catch (Exception e) {
```

En la estructura de HTML tienda encontramos los siguientes elementos:

## 1. Encabezado y Barra de Navegación:

- Formulario de Búsqueda: Permite a los usuarios buscar productos.
- Barra de Navegación: Contiene el logo de la tienda y enlaces para abrir el carrito y salir de la aplicación.

```
<header>
  <nav class="navbar navbar-default">
    <form class="navbar-form navbar-left" id="searchForm" action="#" method="GET"
      onsubmit="event.preventDefault(); searchProducts()">
      <div class="search-container">
        <div class="form-group">
          <input type="text" class="form-control search-box" name="search" id="searchInput"
            placeholder="Buscar..."
            onkeydown="if (event.keyCode === 13) { searchProducts(); event.preventDefault(); }">
        </div>
        <button type="button" class="reset-button" onclick="resetProducts()">Reset</button>
      </div>
    </form>
    <div class="container-fluid" style="margin-left: 220px;">
      <div class="navbar-header">
        <a class="navbar-brand" href="#" style="color: #000; font-size: 24px; font-weight: bold; text-decoration: none;">
          
          Pill-Express
        </a>
      </div>
      <ul class="nav navbar-nav">
        <li><a href="javascript:void(0)" onclick="openCart()">Carrito de la Compra</a></li>
        <li><a href="http://localhost:8080/">Salir</a></li>
      </ul>
    </div>
  </nav>
</header>
```

## 2. Listado de Productos:

- Lista de Productos: Cada producto se muestra con su ID, nombre, imagen, precio, stock y un botón para añadir al carrito.

```

<!-- Carrito de la compra -->
<div id="cart" class="cart">
  <a href="javascript:void(0)" class="closebtn" onclick="closeCart()">&times;</a>
  <h2 style="color: black;">Carrito de la Compra</h2>
  <div id="total-items" style="text-align:center; margin-top:10px;">
    Productos en la cesta: <span id="total-count">0</span>
    <span style="display:none;" sec:authentication="name" id="nombreUsuario"></span>
  </div>
  <ul id="cart-items"></ul>
  <div id="total-price" style="text-align:center; margin-top:20px;">
    Total: <span id="total-precio">0.00</span>
  </div>
  <div class="checkout-container">
    <button onclick="checkout()" class="checkout-button">Hacer Compra</button>
  </div>
</div>

```

### 3. Modal de Descripción:

- Modal: Muestra la descripción de un producto cuando se hace clic en el botón de descripción.

```

<!-- Modal de descripcion-->
<div id="descriptionModal" class="modal">
  <div class="modal-content">
    <span class="close" onclick="closeModal()">&times;</span>
    <h2 style="color: black; font-size:20px;">Descripcion:</h2>
    <p id="modal-description" style="color: #007bb5; font-size:17px;"></p>
  </div>
</div>

```

Las funciones de JavaScript de la tienda en el HTML son las siguientes:

### 1. Buscar productos:

- Filtra los productos mostrados en función del término de búsqueda ingresado por el usuario.

```

// Funcion para realizar la búsqueda de productos
function searchProducts() {
  var searchTerm = document.getElementById("searchInput").value.trim().toLowerCase();
  var products = document.querySelectorAll(".product");
  products.forEach(function (product) {
    var productName = product.querySelector(".product-name").textContent.trim().toLowerCase();
    if (productName.includes(searchTerm)) {
      product.style.display = "block";
    } else {
      product.style.display = "none";
    }
  });
}

```

### 2. Reiniciar búsqueda:

- Muestra todos los productos y limpia el campo de búsqueda.

```

470
471 // Funcion para resetear la búsqueda y mostrar todos los productos
472 function resetProducts() {
473   var products = document.querySelectorAll(".product");
474   products.forEach(function (product) {
475     product.style.display = "block";
476   });
477   document.getElementById("searchInput").value = ""; // Limpiar el campo de búsqueda
478 }
479

```

### 3. Abrir y cerrar carrito:

- Muestra todos los productos y limpia el campo de búsqueda.

```

480 // Funcion para abrir el carrito
481 function openCart() {
482     document.getElementById("cart").style.width = "300px";
483 }
484
485 // Funcion para cerrar el carrito
486 function closeCart() {
487     document.getElementById("cart").style.width = "0";
488 }
489

```

#### 4. Añadir productos al carrito:

- Añade un producto al carrito de compras y actualiza el total de productos y el precio total.

```

function addToCart(button) {
    var productCard = document.querySelector("#product-" + button);
    var productName = productCard.querySelector(".product-name").textContent;
    var productImage = productCard.querySelector(".product-image").src;
    var productPrecio = productCard.querySelector(".product-precio").textContent;

    // Obtener el precio en formato numérico
    var productPriceNumber = parseFloat(productPrecio.replace('Precio: ', '').replace('â€', '').replace(',', ''));

    var cartItems = document.getElementById("cart-items");
    var li = document.createElement("li");
    li.setAttribute("data-id", button);
    li.setAttribute("data-price", productPriceNumber);

    var itemDetails = document.createElement("div");
    itemDetails.className = "cart-item-details";

    var nameSpan = document.createElement("span");
    nameSpan.className = "cart-item-name";
    nameSpan.textContent = productName;

    var priceSpan = document.createElement("span");
    priceSpan.className = "cart-item-price";
    priceSpan.textContent = productPrecio;

    itemDetails.appendChild(nameSpan);
    itemDetails.appendChild(priceSpan);

    var img = document.createElement("img");
    img.setAttribute("src", productImage);
    img.style.width = "50px";
    img.style.height = "50px";
    img.style.marginRight = "10px";

```

```

    var removeButton = document.createElement("button");
    removeButton.className = "cart-item-remove";
    removeButton.textContent = "Eliminar";
    removeButton.onclick = function () {
        li.remove();
        updateTotalPrice();
        updateTotalItems();
    };

    li.appendChild(img);
    li.appendChild(itemDetails);
    li.appendChild(removeButton);
    cartItems.appendChild(li);

    updateTotalPrice();
    updateTotalItems();
}

```

#### 5. Actualizar total del carrito:

- Calculan y actualizan el precio total y el total de productos en el carrito.

```

544 function updateTotalPrice() {
545     var cartItems = document.querySelectorAll("#cart-items li");
546     var totalPrice = 0;
547
548     cartItems.forEach(function (item) {
549         var itemPrice = parseFloat(item.getAttribute("data-price"));
550         totalPrice += itemPrice;
551     });
552
553     document.getElementById("total-precio").textContent = totalPrice.toFixed(2);
554 }
555
556 function updateTotalItems() {
557     var cartItems = document.querySelectorAll("#cart-items li");
558     var totalItems = cartItems.length;
559
560     document.getElementById("total-count").textContent = totalItems;
561 }

```

## 6. Realizar compra:

- Envía los datos del carrito al servidor para procesar la compra.

```

563 function checkout() {
564     var cartItems = document.querySelectorAll("#cart-items li");
565     var productIds = Array.from(cartItems).map(function (li) {
566         return li.getAttribute("data-id");
567     });
568
569     if (productIds.length === 0) {
570         alert("No hay productos en el carrito.");
571         return;
572     }
573
574     var totalPrecio = parseFloat(document.getElementById("total-precio").textContent);
575     var totalItems = parseInt(document.getElementById("total-count").textContent);
576     var nombreUsuario = document.getElementById("nombreUsuario").textContent;
577     // Crear datos de formulario codificados en URL
578     var formData = new URLSearchParams();
579     formData.append("productoIds", productIds.join(','));
580     formData.append("importe", totalPrecio.toFixed(2));
581     formData.append("totalProductos", totalItems);
582     formData.append("nombreUsuario", nombreUsuario);
583     // Crear la URL con los parámetros
584     var url = '/hacerCompra?' + formData.toString();
585
586     // Imprimir la URL que se va a enviar
587     console.log('URL enviada:', url);
588
589     // Enviar los datos al backend usando GET
590     fetch(url, {
591         method: 'GET',
592     })

```

```

593     .then(response => {
594         console.log('Respuesta del servidor:', response);
595         if (!response.ok) {
596             return response.text().then(text => { throw new Error(text) });
597         }
598         return response.text();
599     })
600     .then(data => {
601         alert("Compra realizada con Éxito");
602         window.location.href = "/compraConfirmada";
603     })
604     .catch(error => {
605         console.error('Error:', error);
606         alert("Hubo un error al procesar la compra. Por favor, inténtelo de nuevo.\n" + error.message);
607     });
608 }

```

## 7. Ver descripción del producto:

- Muestra y oculta el modal con la descripción del producto.

```

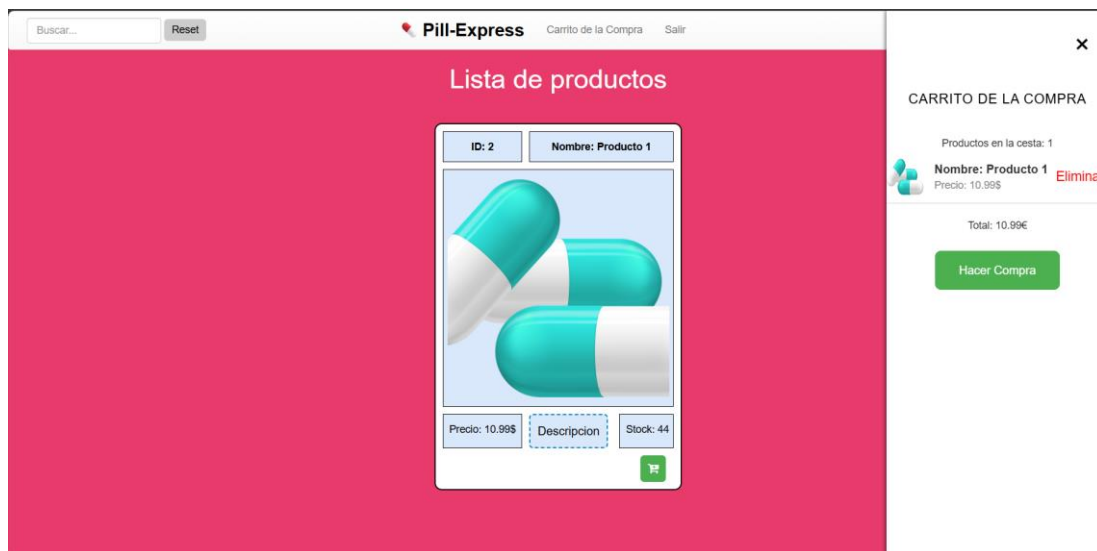
610 function verDescripcion(description) {
611     var productCard = document.querySelector("#product-" + description);
612     var productName = productCard.querySelector(".desc").textContent;
613
614     document.getElementById('modal-description').innerText = productName;
615     document.getElementById('descriptionModal').style.display = 'block';
616 }
617
618 function closeModal() {
619     document.getElementById('descriptionModal').style.display = 'none';
620 }

```

Aspecto de la Tienda en la aplicación:



Aspecto de la Tienda en la aplicación con la cesta desplegada:



### 7.13 Repartidores-Mapa.

La entrega de repartidores se ha realizado mediante un API, de Google Maps, sin embargo, ante la ausencia de una cuenta de pago, hay funciones que no podemos realizar, además de que las ubicaciones se tienen que añadir de manera manual y no automática:

1. Cuerpo (<body>): Contiene el contenido principal de la página.
  - Título (<h1>): Un encabezado centrado que indica "Mapa Repartidores".
  - Mapa de Google Maps (<div id="map">): Un contenedor donde se mostrará el mapa de Google Maps.
  - Actualización de Mapa: Campos de entrada para ingresar coordenadas de latitud y longitud, junto con un botón para actualizar el mapa con la ubicación especificada.
  - Selección de Origen y Destino: Dos selectores desplegables donde se puede seleccionar el origen y el destino para calcular la ruta. También hay un botón para calcular la ruta entre el origen y el destino seleccionados.

```

233<body>
234 <h1 style="text-align: center; color: white;">Mapa Repartidores</h1>
235 <div id="map"></div>
236 <br>
237<div>
238   <label for="lat">Latitud:</label>
239   <input type="text" id="lat" value="40">
240   <label for="lng">Longitud:</label>
241   <input type="text" id="lng" value="-3">
242   <button onclick="updateMap()">Actualizar Mapa</button>
243 </div>
244 <br>
245<div>
246   <label for="origen">Origen:</label>
247   <select id="origen">
248     <option value="">Seleccionar origen</option>
249     <option value="Almacén 1">Almacén 1</option>
250     <option value="Almacén 2">Almacén 2</option>
251     <option value="Almacén 3">Almacén 3</option>
252     <option value="Calle La Oca">Calle La Oca</option>
253     <option value="Calle Aguacate">Calle Aguacate</option>
254     <option value="Calle Baleares">Calle Baleares</option>
255     <option value="Calle Alicante">Calle Alicante</option>
256     <option value="Calle Valencia">Calle Valencia</option>
257     <option value="Calle Durazno">Calle Durazno</option>
258     <option value="Calle Numancia">Calle Numancia</option>
259   </select>
260   <label for="destino">Destino:</label>
261   <select id="destino">
262     <option value="">Seleccionar destino</option>
263     <option value="Almacén 1">Almacén 1</option>
264     <option value="Almacén 2">Almacén 2</option>
265     <option value="Almacén 3">Almacén 3</option>
266     <option value="Calle La Oca">Calle La Oca</option>
267     <option value="Calle Aguacate">Calle Aguacate</option>
268     <option value="Calle Baleares">Calle Baleares</option>
269     <option value="Calle Alicante">Calle Alicante</option>
270     <option value="Calle Valencia">Calle Valencia</option>
271     <option value="Calle Durazno">Calle Durazno</option>
272     <option value="Calle Numancia">Calle Numancia</option>
273   </select>
274   <button onclick="calcularRuta()">Realizar Ruta</button>
275 </div>

```

2. Script (<script>): Contiene el código JavaScript que inicializa el mapa de Google Maps, maneja la actualización del mapa, la selección del origen y el destino, y el



cálculo de la ruta entre ellos. También se agregan marcadores al mapa para cada ubicación especificada.

```

277<script>
278  let map;
279  let directionsService;
280  let directionsRenderer;
281
282  function initMap() {
283    const mapOptions = {
284      center: { lat: 40.41544173794185, lng: -3.685737346264716 },
285      zoom: 10
286    };
287
288    map = new google.maps.Map(document.getElementById("map"), mapOptions);
289    directionsService = new google.maps.DirectionsService();
290    directionsRenderer = new google.maps.DirectionsRenderer();
291    directionsRenderer.setMap(map);
292
293    const locations = [
294      { lat: 40.48144498520098, lng: -3.356719115749229, name: "Almacén 1" },
295      { lat: 40.29721820637667, lng: -3.796891656605112, name: "Almacén 2" },
296      { lat: 40.50239361277144, lng: -3.876896480692295, name: "Almacén 3" },
297      { lat: 40.3891475, lng: -3.7385116, name: "Calle La Oca" },
298      { lat: 40.3681122, lng: -3.7417559, name: "Calle Aguacate" },
299      { lat: 40.3935729, lng: -3.7131164, name: "Calle Baleares" },
300      { lat: 40.3942173, lng: -3.6940328, name: "Calle Alicante" },
301      { lat: 40.4068618, lng: -3.6999526, name: "Calle Valencia" },
302      { lat: 40.3778716, lng: -3.7551234, name: "Calle Durazno" },
303      { lat: 40.4563393, lng: -3.7100056, name: "Calle Numancia" }
304    ];
305
306    locations.forEach(function(location) {
307      let iconUrl;
308      if (location.name === "Almacén 1") {
309        iconUrl = 'http://maps.google.com/mapfiles/ms/icons/red-dot.png';
310      } else if (location.name === "Almacén 2") {
311        iconUrl = 'http://maps.google.com/mapfiles/ms/icons/red-dot.png';
312      } else if (location.name === "Almacén 3") {
313        iconUrl = 'http://maps.google.com/mapfiles/ms/icons/red-dot.png';
314      } else {
315        iconUrl = 'http://maps.google.com/mapfiles/ms/icons/blue-dot.png'; // Por defecto, otros almacenes
316      }
317
318      new google.maps.Marker({
319        position: { lat: location.lat, lng: location.lng },
320        map: map,
321        title: location.name,
322        icon: {
323          url: iconUrl
324        }
325      });
326    });
327  });
328
329  function updateMap() {
330    const lat = parseFloat(document.getElementById("lat").value);
331    const lng = parseFloat(document.getElementById("lng").value);
332
333    if (!isNaN(lat) && !isNaN(lng)) {
334      const newCenter = { lat: lat, lng: lng };
335      map.setCenter(newCenter);
336    } else {
337      alert("Por favor, ingresa valores válidos para latitud y longitud.");
338    }
339  }
340
341  function calcularRuta() {
342    const origen = document.getElementById("origen").value;
343    const destino = document.getElementById("destino").value;
344
345    if (origen === "" || destino === "") {
346      alert("Por favor, selecciona tanto el origen como el destino.");
347      return;
348    }
349
350    const request = {
351      origin: origen,
352      destination: destino,
353      travelMode: google.maps.TravelMode.DRIVING
354    };
355  }

```

```

320    new google.maps.Marker({
321      position: { lat: location.lat, lng: location.lng },
322      map: map,
323      title: location.name,
324      icon: {
325        url: iconUrl
326      }
327    });
328  });
329
330  function updateMap() {
331    const lat = parseFloat(document.getElementById("lat").value);
332    const lng = parseFloat(document.getElementById("lng").value);
333
334    if (!isNaN(lat) && !isNaN(lng)) {
335      const newCenter = { lat: lat, lng: lng };
336      map.setCenter(newCenter);
337    } else {
338      alert("Por favor, ingresa valores válidos para latitud y longitud.");
339    }
340  }
341
342  function calcularRuta() {
343    const origen = document.getElementById("origen").value;
344    const destino = document.getElementById("destino").value;
345
346    if (origen === "" || destino === "") {
347      alert("Por favor, selecciona tanto el origen como el destino.");
348      return;
349    }
350
351    const request = {
352      origin: origen,
353      destination: destino,
354      travelMode: google.maps.TravelMode.DRIVING
355    };
356  }

```

```

357         directionsService.route(request, function(result, status) {
358             if (status == google.maps.DirectionsStatus.OK) {
359                 directionsRenderer.setDirections(result);
360             } else {
361                 alert("Error al calcular la ruta: " + status);
362             }
363         });
364     }
365 }
366
367 document.addEventListener("DOMContentLoaded", function() {
368     initMap();
369 });
370 </script>
371 <!-- <script src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap" async defer></script>
372 </body>
373 </html>

```

## 7.14 Repartidores-Pedidos Pendientes.

Esta sección tiene parte de la lógica de Administrador-Usuarios, sin embargo, el Select muestra solo los pedidos pendientes y una vez entregados se da al botón de entregado y desaparece del ForEach.

Pill-Express Inicio Menu Usuarios Roles Ubicaciones Productos Almacenes Pedidos Detalles Tienda Repartos

Bienvenido oscarisma123 Salir

Listado de pedidos pendientes

Id	FechaPedido	FechaEntrega	Estado	Importe	CantidadProductos	Usuario	Operaciones
7	26-05-2024	26-05-2024	pendiente	10.99	1	7	
8	26-05-2024	26-05-2024	pendiente	32.97	3	7	
9	26-05-2024	26-05-2024	pendiente	10.99	1	7	

© 2024 PILL-EXPRESS, Inc. | WebApp Desarrollada con Spring Boot 2.1.4 | Autores: Oscar Arjona Eoija y Pablo Galvan Tejada | [Privacy](#) · [Terms](#)

## 7.15 Pom.xml y application.properties.

En el pom.xml encontramos las diferentes dependencias que posee el proyecto, como el Thymeleaf, Springboot, Security, etcétera.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4    <modelVersion>4.0.0</modelVersion>
5
6    <groupId>net.itinajero</groupId>
7    <artifactId>empleos</artifactId>
8    <version>0.0.1-SNAPSHOT</version>
9    <packaging>jar</packaging>
10
11    <name>empleos</name>
12    <description>Aplicacion para publicar ofertas de trabajo.</description>
13
14    <parent>
15      <groupId>org.springframework.boot</groupId>
16      <artifactId>spring-boot-starter-parent</artifactId>
17      <version>2.2.2.RELEASE</version>
18      <relativePath/> <!-- lookup parent from repository -->
19    </parent>
20
21    <properties>
22      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
24      <java.version>11</java.version>
25    </properties>
26
27    <dependencies>
28      <dependency>
29        <groupId>org.springframework.boot</groupId>
30        <artifactId>spring-boot-starter-thymeleaf</artifactId>
31      </dependency>
32
33      <dependency>
34        <groupId>org.springframework.boot</groupId>
35        <artifactId>spring-boot-starter-web</artifactId>
36      </dependency>

```

```

38      <dependency>
39        <groupId>org.springframework.boot</groupId>
40        <artifactId>spring-boot-starter-test</artifactId>
41        <scope>test</scope>
42      </dependency>
43
44      <!-- Requerido para trabajar con Spring Data JPA -->
45      <dependency>
46        <groupId>org.springframework.boot</groupId>
47        <artifactId>spring-boot-starter-data-jpa</artifactId>
48      </dependency>
49
50      <dependency>
51        <groupId>mysql</groupId>
52        <artifactId>mysql-connector-java</artifactId>
53        <scope>runtime</scope>
54      </dependency>
55
56      <dependency>
57        <groupId>org.projectlombok</groupId>
58        <artifactId>lombok</artifactId>
59        <optional>true</optional>
60      </dependency>
61
62      <!-- Add support form automatic reloading -->
63      <dependency>
64        <groupId>org.springframework.boot</groupId>
65        <artifactId>spring-boot-devtools</artifactId>
66      </dependency>
67
68      <!-- Requerido para trabajar Thymeleaf Spring Security Tags -->
69      <dependency>
70        <groupId>org.thymeleaf.extras</groupId>
71        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
72      </dependency>
73
74      <!-- Requerido para trabajar Spring Security -->
75      <dependency>
76        <groupId>org.springframework.boot</groupId>
77        <artifactId>spring-boot-starter-security</artifactId>
78      </dependency>

```

```

78     </dependencies>
79
80 </dependencies>
81
82 <build>
83     <plugins>
84         <plugin>
85             <groupId>org.springframework.boot</groupId>
86             <artifactId>spring-boot-maven-plugin</artifactId>
87             <configuration>
88                 <excludes>
89                     <exclude>
90                         <groupId>org.projectlombok</groupId>
91                         <artifactId>lombok</artifactId>
92                     </exclude>
93                 </excludes>
94             </configuration>
95         </plugin>
96     </plugins>
97 </build>
98
99 </project>

```

En `application.properties` encontramos elementos como la configuración de la conexión con la base de datos, la implementación de imágenes en la aplicación, etcétera.

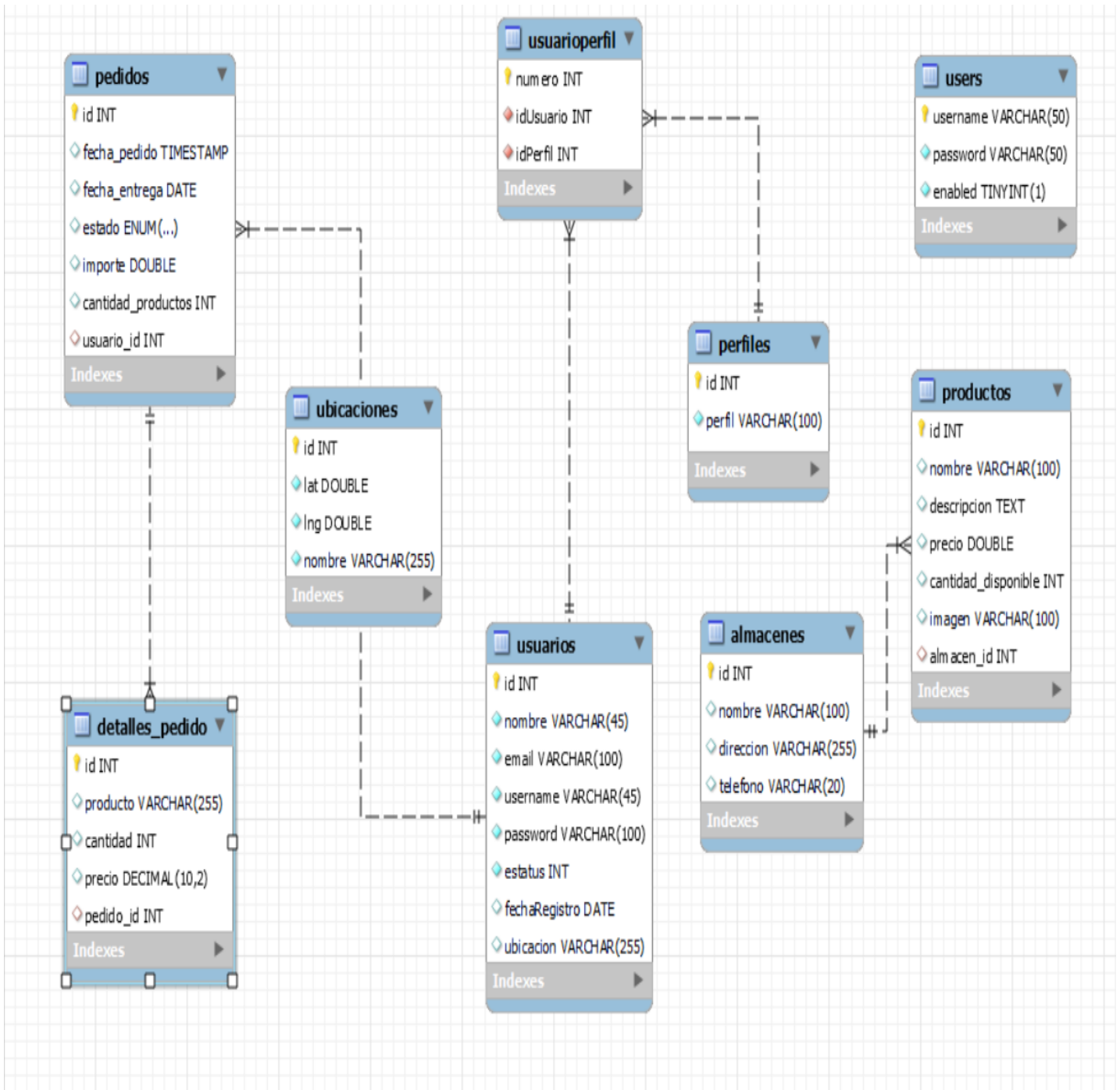
```

1 # CONFIGURACION DEL EMBEDDED SERVER
2 #server.servlet.session.tracking-modes=cookie
3 #server.port=8999
4 # Podemos personalizar el ContextPath de la aplicacion
5 #server.servlet.context-path=/myEmpleosApp
6
7 # CONFIGURACION MYSQL DATASOURCE (MySQL 8.0)
8 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
9 spring.datasource.url=jdbc:mysql://localhost:3306/tfg_bd?useSSL=false&serverTimezone=America/Mexico_City&allowPub
10 spring.datasource.username=root
11 spring.datasource.password=AlumnoIFP
12
13 # CONFIGURACION DE SPRING DATA JPA
14 spring.jpa.generate-ddl=false
15 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
16 spring.jpa.show-sql=true
17 # Spring Data JPA buscara los nombres de las tablas respetando Mayusculas/Minusculas
18 spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
19
20 # CONFIGURACION MULTIPART (SUBIDA DE ARCHIVOS)
21 # ¿Habilitamos subida de archivos?
22 spring.servlet.multipart.enabled=true
23 # Directorio temporal para subir archivos (Windows)
24 spring.servlet.multipart.location=c:/tmp
25 # Directorio intermedio para subir archivos (Linux/MAC)
26 #spring.servlet.multipart.location=/tmp
27 # Maximo tamaño de archivos que se pueden subir
28 spring.servlet.multipart.max-file-size=2MB
29
30 # CONFIGURACION DE LA PAGINACION
31 spring.data.web.pageable.default-page-size=5
32
33 # PROPIEDADES DE EMPLEOS APP
34 empleosapp.ruta.imagenes=c:/empleos/img-vacantes/
35 empleosapp.ruta.cv=c:/empleos/files-cv/
36
37 #spring.security.user.name=itinajero
38 #spring.security.user.password=masterkey
39
40 # CONFIGURACION DEL CHARSET PARA HTTP
41 spring.http.encoding.enabled=true
42 spring.http.encoding.charset=UTF-8

```

## 7.16 Base de Datos.

Modelo de entidad-relacional de la base de datos:



## 8. Conclusiones.

El proyecto Pill-Express ha permitido explorar y desarrollar una mejora para la venta y entrega a domicilio de productos farmacéuticos, integrando múltiples aspectos técnicos y funcionales. A continuación, se detallan las ideas principales a las que se ha llegado gracias a al desarrollo del proyecto:

**Integración Eficiente de Roles:** La estructura del sistema, que incluye apartados específicos para administrador, cliente y repartidores, demuestra la importancia de una clara separación de roles dentro de una aplicación de comercio electrónico. Esto permite una gestión más eficiente y segura de las operaciones, garantizando que cada usuario tenga acceso solo a la información y funcionalidades relevantes para su rol.

**Uso de Spring Boot para el Desarrollo Web:** La elección de Spring Boot como framework para el desarrollo del backend ha sido crucial para la eficiencia y escalabilidad del proyecto. Spring Boot ha proporcionado un entorno robusto y modular que facilita la creación de servicios web, lo cual es fundamental para la comunicación entre los diferentes componentes de la aplicación.

**Desarrollo en Java con Eclipse:** La implementación en Java utilizando Eclipse ha demostrado ser una combinación efectiva para el desarrollo de aplicaciones empresariales. Java, con su enfoque en la seguridad y portabilidad, junto con Eclipse, un IDE potente y extensible, ha permitido una gestión efectiva del proyecto.

**Gestión de Base de Datos con MySQL:** La integración de MySQL como sistema de gestión de base de datos ha asegurado la fiabilidad y consistencia de los datos almacenados. La estructura relacional de MySQL ha facilitado la organización y consulta de la información, soportando las necesidades operativas de la aplicación, tales como el seguimiento de pedidos, gestión de usuarios y control de inventario.

**Facilitación del Acceso a Productos Farmacéuticos:** Pill-Express ha demostrado ser una solución viable para mejorar el acceso a productos farmacéuticos, especialmente para personas con movilidad reducida o en áreas con poca accesibilidad a farmacias. La entrega a domicilio optimiza la comodidad del cliente y potencia los tratamientos médicos.

**Consideraciones de Seguridad y Privacidad:** Durante el desarrollo, se ha puesto un énfasis significativo en la seguridad y privacidad de los datos de los usuarios. El manejo adecuado de información sensible y la implementación de medidas de seguridad, como la autenticación y autorización, son fundamentales en el contexto de una aplicación que maneja datos médicos.

**Experiencia del Usuario:** La aplicación ha sido diseñada con un enfoque centrado en el usuario, ofreciendo una interfaz intuitiva y funcionalidades que facilitan la navegación y el uso del sistema. Esto incluye procesos simplificados para realizar pedidos y la creación de rutas de entregas.

**Impacto en el Mercado Farmacéutico:** Pill-Express tiene el potencial de transformar el mercado farmacéutico, introduciendo una nueva forma de distribución que puede complementar las ventas tradicionales en farmacias físicas. Esto no solo aumenta las opciones para los consumidores, sino que también abre nuevas oportunidades de negocio para las farmacias que adoptan el modelo de entrega a domicilio.

A lo largo del desarrollo de Pill-Express, se han identificado varias áreas en las que el proyecto puede mejorar y expandirse para ofrecer una mejor experiencia al usuario y una funcionalidad más robusta. Las principales futuras mejoras incluyen:

**Sistema de Puntuación y Opiniones de Usuarios:** Incorporar un sistema de puntuación y opiniones permitirá a los usuarios calificar los productos y dejar comentarios sobre su experiencia. Esto no solo ayudará a otros clientes a tomar decisiones, sino que también proporcionará valiosa visión para mejorar el servicio y la oferta de productos. La implementación de este sistema incluirá:

- Interfaz de usuario para agregar y visualizar calificaciones y comentarios.
- Funcionalidades de moderación para gestionar y filtrar opiniones inapropiadas.

**Mejoras en el Aspecto (Front End):** Una mejora significativa en la interfaz de usuario y la experiencia de usuario es fundamental para atraer y retener a los usuarios. Esto incluye:

- Rediseño de la interfaz para hacerla más intuitiva y visualmente atractiva.
- Optimización del diseño responsive para asegurar una experiencia coherente en diferentes dispositivos y tamaños de pantalla.

**Integración Avanzada de Google Maps:** Adquirir una cuenta de la API de Google Maps permitirá acceder a funcionalidades avanzadas que mejorarán significativamente la experiencia de entrega y localización.

**Desarrollo de una Aplicación Móvil para Android:** Extender la plataforma Pill-Express a dispositivos móviles mediante el desarrollo de una aplicación nativa para Android. Esto permitirá a los usuarios acceder al servicio de manera más conveniente desde sus smartphones, y las mejoras incluirán:

- Desarrollo de una aplicación nativa utilizando Android Studio y Kotlin.
- Integración completa con las funcionalidades del backend ya existentes.



- Notificaciones de pedidos y promociones.
- Interfaz optimizada para dispositivos móviles.

Estas mejoras no solo mejorarán la funcionalidad y la experiencia del usuario de Pill-Express. La implementación de estas mejoras requiere una planificación cuidadosa y una dedicación continua para lograr una aplicación de gran calidad.

En resumen, el proyecto Pill-Express ha permitido demostrar cómo una aplicación web puede mejorar significativamente el acceso a productos farmacéuticos y optimizar la gestión operativa. La integración de tecnologías como Spring Boot, Java, y MySQL ha sido clave para lograr una solución robusta y escalable. Este proyecto sienta las bases para futuras mejoras y lograr un desarrollo en el campo de la distribución farmacéutica digital.

## 9. Bibliografía.

- Cosmina, L. (2018). Spring Boot 2 Recipes: A Problem-Solution Approach. Apress.
- Domínguez, J. (2021). Spring Security in Action. Manning Publications.
- Tim, O., & Arnaud, H. (2021). Maven: The Definitive Guide. O'Reilly Media.
- Walls, C. (2019). Spring Boot in Action. Manning Publications.