



INSTITUTO DE ENSEÑANZA SECUNDARIA CASTELAR

FAMILIA PROFESIONAL DE INFORMÁTICA Y COMUNICACIONES

**TÉCNICO SUPERIOR EN DESARROLLO DE
APLICACIONES MULTIPLATAFORMA**

MEMORIA DEL MÓDULO PROFESIONAL DE PROYECTO

TÍTULO: Tienda Online: **MIEL CUARTA GENERACIÓN**

ALUMNO: Andrea Alcázar Galán

Curso Académico: 2021/22

Índice de contenidos

Índice

1. Descripción:	2
2. Justificación y objetivos:	2
3. Coste de implementación:	2
4. Requisitos técnicos:	2
4.1.Lenguaje de programación:	2
4.2.Entornos de desarrollo:	2
4.3.Control de versiones:	3
4.4 Espacio Web:	3
4.5. Base de Datos:	3
4.6.Test de código:	3
5. Metodología:	3
5.1. Planificación:	3
5.3. Implementación del código	3
5.4.Test	3
5.5. Cronograma de desarrollo.	4
6. Estructura de datos	4
6.1. Creación Base de datos	5
7. Evaluación del proyecto:	6
7.1.Propuestas de mejora	6
8.Bibliografía	6
9. Manual del Programador- tienda:	7
9.1. Implementación:	7
9.2. Backend:	8
10. Manual de Usuario - Tienda:	19
11. Manual del Programador- Android:	23
12. Manual del Usuario - Android:	28

1. Descripción:

El objetivo de la aplicación será crear una tienda virtual de miel y artesanía, donde se podrá adquirir diferentes productos, la web contará con un lista de productos que podrás añadir a un carrito. Se implementará una pasarela de pago vía tarjeta de crédito o PayPal. Además se controlará activamente el precio de los productos, ya que mi cliente necesitará manipularlo frecuentemente y para ello, se creará un panel de administrador gráfico. Como objetivo secundario queremos realizar un diseño personalizado que ayude a satisfacer las necesidades de nuestro cliente, en este caso, adecuado para los usuarios de mediana edad que será nuestro cliente objetivo.

Por otro lado, se desarrollará una aplicación Android que busca la optimización respecto a la gestión de los pedidos y las relaciones con los clientes.

2. Justificación y objetivos:

El proyecto que se va a realizar se enmarca en el contexto del ciclo de Desarrollo de Aplicaciones Multiplataforma del IES Castellar (Badajoz). Como cierre del ciclo se realiza una aplicación completa, con su correspondiente metodología de desarrollo y sus herramientas complementarias. Este módulo, con una carga de 40 horas, permite contemplar el ciclo completo de creación de un sistema informática.

La elección de una aplicación web, utilizando las tecnologías Spring Boot, MYSQL, Android. Ya que estas tecnologías me aportan justo lo que necesito para llevar mi proyecto a cabo.

3. Coste de implementación:

El coste de implementación en esta primera fase de desarrollo del proyecto es cero porque se emplean herramientas de código abierto que están a nuestra disposición por ser alumno del centro educativo. Las licencias de ese software propietario se asignaron gracias a nuestras cuentas de correo electrónico al principio de curso.

4. Requisitos técnicos:

El proyecto necesita varias herramientas, lenguajes y tecnologías que se detallan a continuación:

4.1.Lenguaje de programación:

En el desarrollo se va a utilizar el lenguaje de programación Java.

Java es un lenguaje de código abierto, multiplataforma, orientado a objetos que se puede utilizar como una plataforma en sí mismo. Es un lenguaje de programación rápido, seguro y fiable para codificar todo, desde aplicaciones móviles y software empresarial hasta aplicaciones de macrodatos y tecnologías del lado del servidor.

4.2.Entornos de desarrollo:

Como entornos de trabajo para esta aplicación utilizaremos algunos IDE de JetBrains como IntelliJ y Android Studio.

4.3. Control de versiones:

La aplicación se desarrolla utilizando una plataforma que nos ayuda a controlar los cambios en el código y a trabajar en equipo o desde diferentes ordenadores. Esta plataforma también ayuda a regresar a versiones anteriores, crear diferentes ramas de desarrollo y en general a contar con unas copias de seguridad fiable.

Las opciones actuales más extendidas son GitLab y GitHub, ambos casos contamos con acceso gratuito con varias funciones, usando nuestras cuentas del Castelar.

4.4 Espacio Web:

El proyecto cuenta con una tienda online.

4.5. Base de Datos:

Se utiliza una base de datos MYSQL proporcionada por el centro educativo

4.6. Test de código:

Dada la sencillez de la aplicación no se ha empleado ninguna herramienta de diseño de test. Solo se han practicado test informales para asegurarnos la calidad del código.

5. Metodología:

Para el desarrollo de la aplicación hemos seguido una serie de pasos que describen a continuación:

5.1. Planificación:

Antes de comenzar el desarrollo hemos revisado una serie de páginas, cursos, videotutoriales acerca de Spring Boot.

-Hemos generado un proyecto en GitLab para alojar la herramienta y la hemos clonado en el equipo.

5.3. Implementación del código

Una vez creado el proyecto y las tablas necesarias, hemos ido desarrollando el proyecto paso a paso, primero la parte de backend relacionada con la parte del panel de administrador.

5.4. Test

Aunque han sido test informales, cada pocos días se probaba el proyecto y se realizaban los ajustes necesarios para resolver los problemas.

5.5. Cronograma de desarrollo.

Semana/Fase	Planificación	Documentación	BBDD	Código	test
3ºMarzo	X		X		
4ºmarzo	X		X	X	
1ºabril	X		X	X	
2ºabril	X		X	X	
3ºabril	X			X	
4ºabril	X	X		X	
1ºmayo	X			X	
2ºmayo	X	X	X	X	
3ºmayo	X			X	
4ºmayo	X	X		X	X
1ºjunio	X	X		X	X

6. Estructura de datos

La base de datos que se emplea en la aplicación consta de 7 tablas con un total de 36 campos y está creada siguiendo el esquema ANSI SQL.

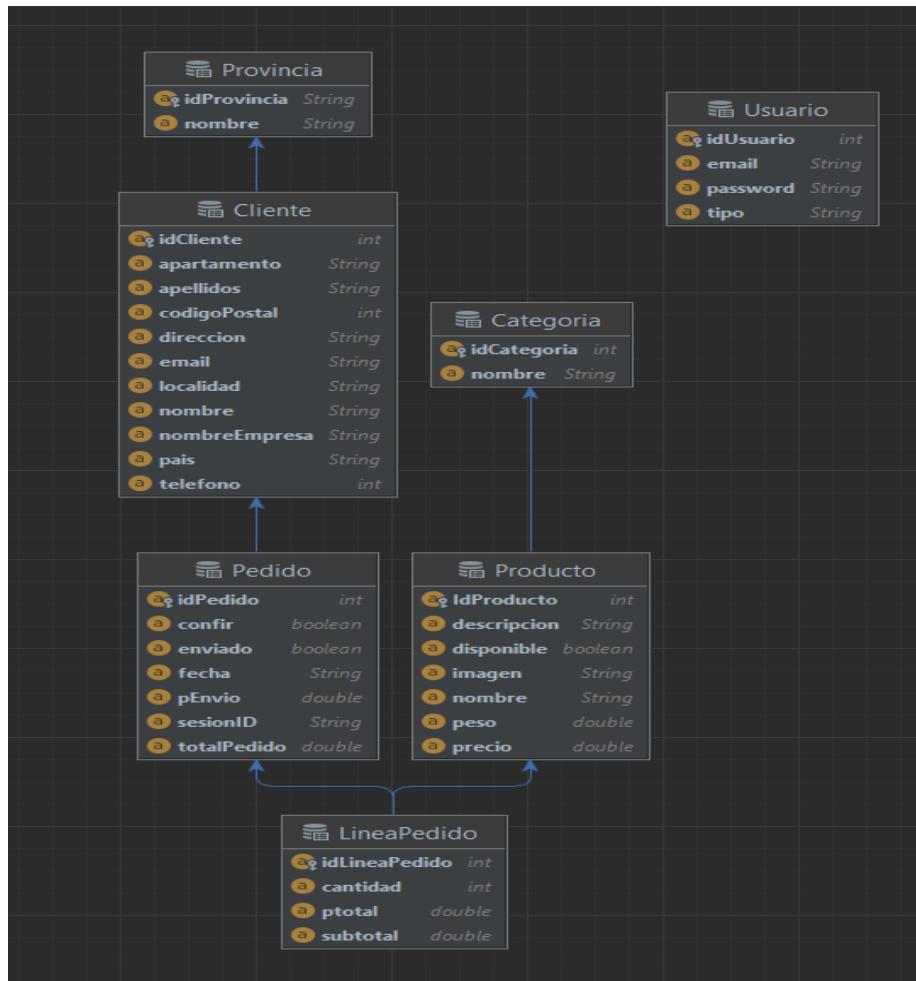
Para la aplicación se han diseñado un conjunto de entidades siguiendo el paradigma de la Programación Orientada a Objetos. Las clases son las siguientes:

Usuario, Provincia, Cliente, Pedido, LineaPedido, Producto y Categoría.

The screenshot displays a Java IDE interface with several open files: Controller.java, MainController.java, SPRING_SESSION, and application.properties. The application.properties file contains configuration for a MySQL database connection, including the URL, username (pmydm2201), password (dam2022), driver class (com.mysql.cj.jdbc.Driver), and other properties like server.port=9000 and spring.jpa.show-sql=true. The right side of the interface shows the database structure for the 'pmydm2201' database. It lists seven tables: categoria, cliente, linea_pedido, pedido, producto, provincia, and SPRING_SESSION. The SPRING_SESSION table is expanded to show its columns: PRIMARY_ID, SESSION_ID, CREATION_TIME, LAST_ACCESS_TIME, MAX_INACTIVE_INTERVAL, EXPIRY_TIME, and PRINCIPAL_NAME, along with its keys and indexes. A separate table, SPRING_SESSION_ATTRIBUTES, is also visible. The bottom of the interface shows a 'Server Objects' section.

6.1. Creación Base de datos

Aquí se puede ver el diagrama final de la base de datos de la tienda online (Miel Cuarta Generación).



7. Evaluación del proyecto:

Del proyecto que hemos completando podemos sacar algunas conclusiones:

- A) Conclusión 1: mejorar mis conocimientos sobre Spring y fronted en HTML,CSS,Bootstrap

7.1. Propuestas de mejora

Qué podría mejorar si tuviese más tiempo:

- Spring Session o implementar Cookies.
- Spring Security para lograr una página web segura.
- Barra de Buscar (tienda) por Nombre del producto o categoría.

8. Bibliografía

Para este proyecto hemos planteado las siguientes páginas web:

- Enlace para material del proyecto en DRIVE:

<https://drive.google.com/drive/folders/1RPUgMoJfCaOouSanBucsv-8fw734qthS?usp=sharing>

- Spring Boot:

<https://sb-ui-kit-pro.startbootstrap.com/index.html>

<https://github.com/StartBootstrap/startbootstrap-shop-item>

<https://github.com/StartBootstrap/startbootstrap-shop-homepage>

https://www.javainuse.com/spring/springboot_session

<https://github.com/Java-Techie-jt/spring-boot-paypal-example>

https://www.youtube.com/watch?v=uglIUObNHZdo&ab_channel=CodeWithArjun

- Android:

<https://square.github.io/retrofit/>

<https://www.codegrepper.com/code-examples/whatever/retrofit+spring+boot>

9. Manual del Programador- tienda:

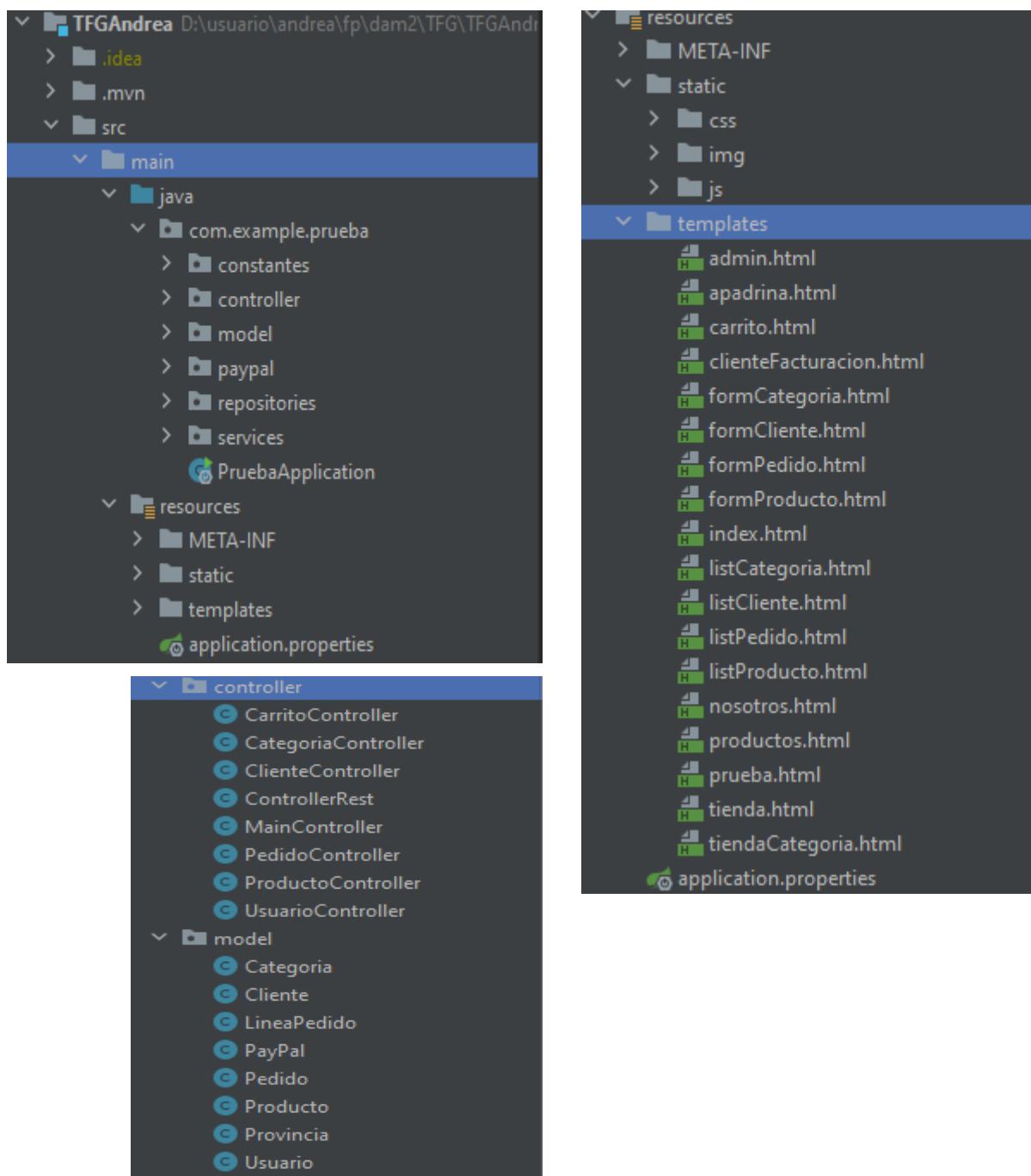
La aplicación está separada en dos partes principales que son el panel de administración y la tienda donde los clientes podrán adquirir sus productos.

La tienda destacará por su front y solo contará con backend en la parte de la elección de los productos y el pago de estos mediante PayPal. Además se mandará un correo electrónico de verificación al cliente. Para ello, se utilizará el pojo de LíneaPedido, un controlador Carrito, vista tienda, carrito y clienteFacturación.html

Por otro lado, el panel de administración contará con un login y accederemos a través de la url *localhost:9000/admin*. se utilizarán todos los Controller y vistas que puede ver en el punto 6.1.

9.1. Implementación:

Para el desarrollo de este proyecto se ha comenzado, primero creando un proyecto de Spring Boot con todas las dependencias necesarias. Se añadieron los POJOS, controllers y servicios correspondientes y se agregó la base de datos remota (castelarlamp...) al fichero properties. Como dependencias secundarias se añadieron PayPal, Bootstrap, Lombok, Tomcat, MailSender...



A continuación, se analizarán las partes generales de la aplicación mediante diferentes capturas de pantalla, centrándonos en el backend.

9.2. Backend:

Como se ha mencionado anteriormente el backend de este proyecto se puede diferenciar en dos partes:

-**En primer lugar**, la parte de administración, que solo utilizarán los usuarios administradores de la empresa, estos podrán agregar productos,categorías,clientes y revisar los pedidos realizados.

Para ello se debe diferenciar los POJOS Cliente y Usuario, ya que no son iguales y solo este último podrá acceder al login (bajo la url *localhost:9000/admin*) para modificar lo que se necesite.

Se utilizará por lo tanto, el controlador *UsuarioController* para implementar el login.

```
@GetMapping("/admin")
public String admin(Model model) {
    model.addAttribute(attributeName: "usuarioForm", new Usuario());
    return "admin";
}

@PostMapping("/admin/submit")
public String login(@ModelAttribute("loginForm") Usuario usuario) {
    Usuario usuarioTemp = servicio.loginByPass(usuario.getEmail(), usuario.getPassword());
    if (usuarioTemp != null) {
        Constante.login = usuarioTemp;
        return "redirect:/cliente/list"; //INICIA SESION
    } else {
        return "admin"; //DATOS INCORRECTOS
    }
}

@GetMapping("/admin/cerrar")
public String usuarioCerrarSesion() {
    Constante.login = null;
    return "redirect:/admin";
}
```

En la captura de la izquierda se puede ver un ejemplo básico de login donde se comprobará si el email y password introducido por el usuario es correcto en nuestra base de datos.

También se implementa un cerrar sesión.

```
@Query("SELECT u FROM Usuario u WHERE u.email = :email and u.password = :password")
Usuario loginByPass(@Param("email") String email, @Param("password") String password);
```

Una vez el usuario está logueado, lo primero que verá será la vista: */cliente/list* donde podrá encontrar una lista con todos los clientes que hayan realizado algún pedido. Como se ha mencionado anteriormente solo se podrán agregar Clientes,Categorías y Productos. El proceso que se ha seguido para desarrollar la inserción y listar es el siguiente:

Vistas: listCategoria y formCategoria. (Mencionar que para Productos y Clientes se ha seguido la misma dinámica).

```

<h1>Categorías de la Empresa</h1>
<table class="table">
  <thead>
    <tr>
      <th>ID</th>
      <th>Nombre</th>
    </thead>
    <tbody>
      <tr th:each="categoria : ${listaCategorias}">
        <td th:text="${categoria.idCategoria}"></td>
        <td th:text="${categoria.nombre}"></td>
        <td><a th:href="@{/categoria/edit/{id}(id=${categoria.idCategoria})}" class="btn btn-primaryList">Editar</a>
          <a th:href="@{/categoria/borrar/{id}(id=${categoria.idCategoria})}" class="btn btn-danger">Borrar</a>
        </td>
      </tr>
    </tbody>
  </table>
</div>

<div class="content-main">
  <form method="post" action="#" th:action="${categoriaForm.idCategoria != 0} ? @{/categoria/edit/submit} : @{/categoria/new/submit}" th:object="${categoriaForm}">
    <h1 th:text="${categoriaForm.idCategoria != 0} ? 'Editar categoria' : 'Nueva categoria'">Nueva categoria</h1>
    <div class="form-group">
      <label for="idCategoria"></label> <input type="hidden" class="form-control" id="idCategoria" th:field="*{idCategoria}" />
    </div>
    <div class="form-group">
      <label for="nombre">Nombre</label> <input type="text" class="form-control" id="nombre" placeholder="5928553w" th:field="*{nombre}" required/>
    </div>
    <button type="submit" class="btn btn btn-primary">Enviar</button>
  </form>
</div>

```

```

@GetMapping({@"/categoria/list"})
public String listar(Model model) {
  model.addAttribute( attributeName: "listaCategorias", service.findAll());
  return "listCategoria";
}

@GetMapping({@"/categoria/new"})
public String CategoriaNewForm(Model model) {
  model.addAttribute( attributeName: "categoriaForm", new Categoria());
  model.addAttribute( attributeName: "provincias", service.findAll());
  return "formCategoria";
}

@PostMapping({@"/categoria/new/submit"})
public String nuevoCategoriaSubmit(@ModelAttribute("categoriaForm") Categoria categoria) {
  service.add(categoria);
  return "redirect:/categoria/list";
}

```

Por otro lado, a la derecha se encuentran los métodos que se utilizan para editar y borrar categorías.

A la izquierda se pueden ver los métodos utilizados para listar e insertar las categorías.

```

@GetMapping({@"/categoria/edit/{id}"})
public String editarCategoriaForm(@PathVariable Integer id, Model model, Categoria categoria) {
  categoria = service.findById(id); // pasamos el id al servicio
  if (categoria != null) {
    model.addAttribute( attributeName: "categoriaForm", categoria);
    model.addAttribute( attributeName: "provincias", service.findAll());
    return "formCategoria";
  } else {
    return "redirect:/categoria/new";
  }
}

@PostMapping({@"/categoria/edit/submit"})
public String editarCategoriaSubmit(@ModelAttribute("categoriaForm") Categoria categoria) {
  service.edit(categoria);
  return "redirect:/categoria/list";
}

@GetMapping({@"/categoria/borrar/{id}"})
public String borrarCategoriaForm(@PathVariable Integer id) {
  Categoria categoria = service.findById(id);
  if (categoria != null) {
    service.delete(categoria);
    return "redirect:/categoria/list";
  } else {
    return "redirect:/categoria/new";
  }
}

```

-En segundo lugar: la parte de Tienda Online y a la que tendrá acceso cualquier persona de internet. En esta segunda parte se utilizará el POJO Cliente ya que este es el realizará el pedido.



```
@GetMapping("/{id}")
public String welcome(Model model, HttpSession session) {
    /unchecked/
    List<String> messages = (List<String>) session.getAttribute("MY_SESSION_MESSAGES");
    if (messages == null) {
        messages = new ArrayList<>();
    }
    model.addAttribute("attributeName", "sessionMessages", messages);
    return "index";
}
```

Para el Control de Sesiones se ha utilizado Spring Session (Management). Cuando un usuario entra a la página web (`localhost:9000/`), se le asignará automáticamente una SESSION_KEY única, que estará almacenada en la tabla SPRING SESSION creada automáticamente por este método.

El cliente accederá a la vista tienda (se puede ver en pág 21), donde seleccionará un producto a su elección.

Si se pulsa en el botón *Ver Descripción* se podrá ver tanto una descripción más detallada sobre el producto como añadirlo al carrito. Cada uno de los datos del producto son obtenidos de la base de datos (nombre, precio, categoría a la que pertenece utilizando una consulta).

```
public interface ProductoRepository extends JpaRepository<Producto, Integer> {

    @Query("SELECT u FROM Producto u where u.categoría.idCategoria = :idCategoria")
    List<Producto> selectProducto(@Param("idCategoria") Integer idCategoria);
}
```

Este botón (de la imagen inferior) llevará hacia el método (`/productos/1`) y cargará la vista productos, además se creará un línea de pedido en la que se insertará la cantidad del producto que el cliente seleccione.

```
@GetMapping("{@v}/productos/{idProducto}")
public String listarDescripción(Model model, @PathVariable int idProducto) {

    LineaPedido lineaPedido = new LineaPedido();

    Producto producto = servicio.findById(idProducto);
    model.addAttribute("lineaPedido", lineaPedido);
    model.addAttribute("listaProductosTienda", producto);
    return "productos";
}
```



Cuando se pulse el botón *Agregar Carrito*, estos datos se pasarán a través de un *method=POST* hacia el método `/carrito/1`, donde se hará toda la lógica (crear pedido, lineaPedido y el cálculo del precio del pedido).

```
<section class="pt-5">
    <form method="post" action="#" th:action="@{/carrito/{idProducto}(idProducto=${listaProductosTienda.idProducto})}"
          th:object="${listaProductosTienda}">
        <form th:object="${lineaPedido}"...>
    </form>
</section>
```

Para realizar toda la creación de los pedidos, lineaPedido,etc se han utilizado varias consultas que se detallan a continuación:

```
public interface PedidoRepository extends JpaRepository<Pedido, Integer> {

    @Query("SELECT u FROM Pedido u where u.sessionID = :sessionID and u.confir= false")
    Pedido selectPedido( @Param("sessionID") String sessionID);
    @Query(value = "SELECT SESSION_ID FROM SPRING_SESSION", nativeQuery = true)
    String obtenerID();
}
```

Al añadir el producto al carrito lo que en realidad ocurre en la base de datos es una inserción a la tabla LineaPedido. Lo primero que se hará será obtener la SESSION_KEY actual, se la asignará a el objeto *pedido* y se creará un objeto *pedido1* con esa sesión. Si no existe ningún pedido se crea y se editará cuando ya exista algún registro.

```
@PostMapping({@"/carrito/{idProducto}"})
public String carrito(@ModelAttribute("lineaPedido") LineaPedido lineaPedido, Pedido pedido, @PathVariable int idProducto) {
    pedido.setSessionID(servicePedido.obtenerID());

    Pedido pedido1 = servicePedido.selectPedido(pedido.getSessionID());

    if (pedido1 == null) {
        servicePedido.add(pedido);
        lineaPedido.setPedido(pedido);
    } else {
        servicePedido.edit(pedido1);
        lineaPedido.setPedido(pedido1);
    }
}
```

A continuación, se buscará el producto seleccionado anteriormente y se creará una LineaPedido con ese producto. Además, se comprobará que el pedido no esté confirmado.

```
Producto producto = servicio.findById(idProducto);
lineaPedido.setProducto(producto);

LineaPedido linea = serviceLineaPedido.loginByProducto(lineaPedido.getProducto().getIdProducto());
```

Se utiliza la consulta loginByProducto().

```
public interface LineaPedidoRepository extends JpaRepository<LineaPedido, Integer> {

    @Query("SELECT u FROM LineaPedido u where u.pedido.idPedido = :idPedido and u.pedido.confir= false")
    List<LineaPedido> selectLineas(@Param("idPedido") Integer idPedido);

    @Query("SELECT u FROM LineaPedido u WHERE u.producto.IdProducto = :idProducto and u.pedido.confir= false")
    LineaPedido loginByProducto(@Param("idProducto") Integer idProducto);

}
```

El siguiente paso será validar si la lineaPedido creada anteriormente existe o no. Si no existe, se crea, y en ambos casos se calculará el Subtotal de la línea (Precio X cantidad), el Peso Total (Peso X cantidad).

Si la línea ya existe se sumarán las cantidades anteriores a la actual.

```
if (linea == null) {  
  
    lineaPedido.setSubtotal(lineaPedido.getProducto().getPrecio() * lineaPedido.getCantidad());  
    lineaPedido.setPtotal(lineaPedido.getProducto().getPeso() * lineaPedido.getCantidad());  
    calcularEnvio(lineaPedido);  
  
    serviceLineaPedido.add(lineaPedido);  
    recorrerCarrito(lineaPedido);  
  
} else {  
    linea.setCantidad(linea.getCantidad() + lineaPedido.getCantidad());  
  
    linea.setSubtotal(linea.getSubtotal() + (lineaPedido.getProducto().getPrecio() * lineaPedido.getCantidad()));  
    linea.setPtotal(linea.getPtotal() + (lineaPedido.getProducto().getPeso() * lineaPedido.getCantidad()));  
    calcularEnvio(linea);  
  
    serviceLineaPedido.edit(linea);  
    recorrerCarrito(linea);  
}  
return "redirect:/tienda";  
}
```

Por otro lado, se utilizan 2 métodos especiales:

- **calcularEnvio()**: se utiliza para establecer el precio del Envío dependiendo del peso del pedido, ya que las empresas como Correos siguen unos estándares. En este caso, por pedidos con un peso inferior a 3€, se cobrará 8€ de envío. Entre 3kg y 5.99kg se te cobrará 10€,etc...

```
public void calcularEnvio(@ModelAttribute("lineaPedido") LineaPedido linea) {  
  
    if (linea.getPtotal() < 3) {  
        linea.getPedido().setpEnvio(8);  
    } else if (linea.getPtotal() >= 3 && linea.getPtotal() < 6) {  
        linea.getPedido().setpEnvio(10);  
    } else if (linea.getPtotal() >= 6 && linea.getPtotal() < 10) {  
        linea.getPedido().setpEnvio(12);  
    } else {  
        linea.getPedido().setpEnvio(18);  
    }  
}
```

- **recorrerCarrito()**: este método simplemente se utiliza para calcular el total del carrito. Se usará la consulta `selectLineas()` que te devuelve una lista con cada línea del pedido y mediante un bucle for se irá recorriendo cada línea para calcular el total del carrito. Por último, a este total se le sumará el precio de envío correspondiente al peso del pedido.

```
public void recorrerCarrito(LineaPedido linea) {  
    double totalCarrito = 0;  
    List<LineaPedido> lineaPedido1 = serviceLineaPedido.selectLineas(linea.getPedido().getIdPedido());  
  
    for (LineaPedido lineaPedido : lineaPedido1) {  
        totalCarrito = totalCarrito + lineaPedido.getSubtotal();  
    }  
    linea.getPedido().setTotalPedido(totalCarrito + linea.getPedido().getpEnvio());  
}
```

A continuación, el cliente se dirigirá hacia la url `/carrito`, y como se puede comprobar, se mostrarán todos los productos que se hayan añadido. El total del pedido será siempre el precio del producto + los gastos de envío que se calculan siguiendo unos estándares de peso ya mencionados anteriormente.

The screenshot shows the shopping cart page for 'Mel Cuarta Generación'. At the top, there's a navigation bar with links: BIENVENIDOS, NOSOTROS, APADRINA, TIENDA, CARRITO, and a shopping cart icon. Below the navigation is a section titled 'Productos de la Empresa' (Products of the Company) with a table:

Producto	Nombre	Precio	Cantidad	Subtotal	Total Peso	Categorías
	miel flores 1kg	6.99 €	1	6.99 €	1.0 kg	bote-miel regalos velas
	miel retama 1kg	8.99 €	1	8.99 €	1.0 kg	

Below this is a summary table for the 'Total Carrito' (Total Cart):

Envío	8.0 €
Total Pedido	23.98 €

At the bottom is a 'Pagar' (Pay) button.

Para eliminar un producto del carrito se utilizará el siguiente método. En primer lugar se procederá a eliminar la línea de Pedido y a continuación se llamará al método recorrerCarrito() que actualizará el precio total del pedido.

```
@GetMapping("/carrito/borrar/{id}")
public String borrarProductoForm(@PathVariable int id){
    LineaPedido lineaPedido=serviceLineaPedido.findById(id);
    if (lineaPedido!= null) {
        serviceLineaPedido.delete(lineaPedido);
        recorrerCarrito(lineaPedido);
        return "redirect:/carrito";
    }else{
        return "redirect:/index";
    }
}
```

The screenshot shows the shopping cart page after one item has been removed. The 'Productos de la Empresa' table now only contains one item:

Producto	Nombre	Precio	Cantidad	Subtotal	Total Peso
	miel flores 1kg	6.99 €	1	6.99 €	1.0 kg

The 'Total Carrito' summary table shows:

Envío	8.0 €
Total Pedido	14.99 €

At the bottom is a 'Pagar' (Pay) button.

Al eliminar todos los productos del carrito se mostrará así la vista:

The screenshot shows the shopping cart page after all items have been removed. The 'Productos de la Empresa' table is empty:

Producto	Nombre	Precio	Cantidad	Subtotal	Total Peso
----------	--------	--------	----------	----------	------------

The 'Total Carrito' summary table shows:

Envío	0.0 €
Total Pedido	0.0 €

Se comprueba que no exista ninguna lineaPedido, es decir, ningún producto en el carrito. En ese caso, tanto el envío como el total del pedido será 0.

```
@GetMapping({@PathVariable "carrito"})
public String verCarrito(Model model, LineaPedido lineaPedido) {
    Pedido pedido1 = servicePedido.selectPedido(servicePedido.obtenerID());
    lineaPedido.setPedido(pedido1);

    List<LineaPedido> lineaPedido1 = serviceLineaPedido.selectLineas(pedido1.getIdPedido());

    model.addAttribute( attributeName: "lineasCarrito", lineaPedido1);
    model.addAttribute( attributeName: "listCategorias", serviceCategoria.findAll());
    if (lineaPedido1.size() ==0){
        pedido1.setpEnvio(0);
        pedido1.setTotalPedido(0);
        model.addAttribute( attributeName: "totalCarrito", lineaPedido.getPedido().getTotalPedido());
        model.addAttribute( attributeName: "precioEnvio", pedido1.getpEnvio());
    }
    model.addAttribute( attributeName: "totalCarrito", lineaPedido.getPedido().getTotalPedido());
    model.addAttribute( attributeName: "precioEnvio", pedido1.getpEnvio());
    return "carrito";
}
```

Se implementará la siguiente condición en la vista /carrito para ocultar el botón pagar.

```
<div class="text-center" th:if="${lineasCarrito.size() !=0}">
    <a class="btn btn-outline-dark mt-auto" th:href="@{/cliente/facturacion}">Pagar</a>
</div>
</div>
```

Al pulsar en pagar se redirigirá hacia otra vista: *clienteFacturación*. Aquí el cliente rellenará sus datos, (como un registro) y se almacenarán estos datos en la base de datos (tabla Clientes).

Cuando se pulse el botón de pagar con tarjeta, se desplegará nuestra vista de PayPal para iniciar sesión con tu cuenta y realizar así la compra con éxito.

(En nuestro caso, utilizaremos un correo de prueba proporcionado por la api de PayPal para desarrolladores, que se mostrará más adelante,pág:17).

The screenshot shows a web page with a header 'Mel Cautia Generación' and navigation links 'BIENVENIDOS', 'NOSOTROS', 'APADRINA', 'TIENDA', 'CARRITO'. The main content has two sections:

- Nuevo cliente:** A form with fields for Nombre (5928553w), Apellidos (andrea), Nombre Empresa (c/los huertos), País (España), Dirección (usu), Apartamento (1234), Provincia (---), Localidad (6535353), Código postal (0), Teléfono (0), and Email (6535353).
- Total Carrito:** A table showing the cart summary:

Producto	miel flores 1kg
Subtotal	0.00
Envío	8.0 C
Total Pedido	14.99 C

A blue 'Pagar' button is located below the table.

```

@GetMapping({"/cliente/facturacion"})
public String clienteNewForm(Model model, LineaPedido lineaPedido, PayPal paypal) {
    Pedido pedido1 = servicePedido.selectPedido(servicePedido.obtenerID());
    lineaPedido.setPedido(pedido1);

    List<LineaPedido> lineaPedido1 = serviceLineaPedido.selectLineas(pedido1.getIdPedido());

    • paypal.setCurrency("EUR");
    • paypal.setMethod("paypal");
    • paypal.setIntent("sale");

    • model.addAttribute( attributeName: "price", pedido1.getTotalPedido());
    model.addAttribute( attributeName: "currency", paypal.getCurrency());
    model.addAttribute( attributeName: "method", paypal.getMethod());
    model.addAttribute( attributeName: "intent", paypal.getIntent());

    //CONFIRMAT
    model.addAttribute( attributeName: "clienteAceptar", new Cliente());
    model.addAttribute( attributeName: "provincias", provinciaService.findAll());
    model.addAttribute( attributeName: "lineasCarrito", lineaPedido1);
    model.addAttribute( attributeName: "totalCarrito", lineaPedido.getPedido().getTotalPedido());
    model.addAttribute( attributeName: "precioEnvio", pedido1.getpEnvio());

    return "clienteFacturacion";
}

```

A través de la vista obtenemos el valor total del Pedido.

```

<input type="text" id="price" name="price" th:value="${price}" hidden >
<input type="text" id="currency" name="currency" placeholder="Enter Currency" th:value="${currency}" hidden >
<input type="text" id="method" name="method" placeholder="Payment Method" th:value="${method}" hidden >
<input type="text" id="intent" name="intent" placeholder="intent" th:value="${intent}" hidden>

```

Algo a destacar es la utilización de una librería especial que mencionaremos a continuación.

Se utiliza PayPal, una clase POJO creada con *Lombok*. Una librería que te facilita el trabajo en la reducción del código, ya que implementando diferentes anotaciones, entre ellas se encuentra `@Data`, te permitirá utilizar métodos get y set, `toString` directamente.

```

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>

```

```

package com.example.prueba.model;

import lombok.Data;

@Data
public class PayPal {

    private String currency;
    private String method;
    private String intent;
    private String description;
}

```

Cuando el cliente pulse en el botón de pagar se comprobará si este cliente ya existe en nuestra base de datos y se ejecutará un método POST que explicaremos a continuación.

```

<form method="post" action="#">
    th:action="${clienteAceptar.idCliente != 0 and clienteAceptar.email !=null} ? @{/pay} : @{/pay}" th:object="${clienteAceptar}">
    <h1 th:text="${clienteAceptar.idCliente != 0} ? 'Pagar Tarjeta' : 'Nuevo cliente'">Direccion de facturación</h1>

```

En este método como se puede observar, se añadirá el cliente si no existe y otras serie de modificaciones en el pedido.

Si seguimos con la parte del pago con PayPal, previamente hemos tenido que crearnos una cuenta como desarrolladores y añadir la `client_key` y `secret_key` a las properties del proyecto.

PayPal también nos proporcionará una cuenta de cliente que se utilizará para realizar los testing.

<input type="checkbox"/> Account name	Type	Country	Date created	Manage accounts
<input type="checkbox"/> sb-texli16047056@business.example... DEFAULT	Business	ES	21 May 2022	...
<input type="checkbox"/> sb-61zqp16541917@personal.example... DEFAULT	Personal	ES	21 May 2022	...

A continuación crearemos un objeto payment que iremos rellenando con los datos obtenidos de la vista `clienteFacturacion`. Por otro lado, en nuestro servicio se ha realizado un método que establece toda la configuración de PayPal, como por ejemplo declarar las transacciones ,pagos, el Ammount (el precio),etc.

```
@PostMapping({@"/pay"})
public String nuevoClienteSubmit(@ModelAttribute("clienteAceptar") Cliente cliente, PayPal paypal) {
    Pedido pedido1 = servicePedido.selectPedido(servicePedido.obtenerID());
    Cliente cliente1 = clienteService.getCliente(cliente.getEmail());

    //creo el cliente si no existe
    if (cliente1 == null) {
        clienteService.add(cliente);
        pedido1.setCliente(cliente);
    } else {
        //añadir al pedido el cliente
        pedido1.setCliente(cliente1);
    }
    //agrego la fecha actual al pedido
    pedido1.setFecha(String.valueOf(LocalDate.now()));

    /**
     * La parte de Paypal
     */
    try {
        Payment payment = paypalService.createPayment(pedido1.getTotalPedido(), paypal.getCurrency(), paypal.getMethod(),
            paypal.getIntent(), cancelUrl: "http://localhost:9000/" + CANCEL_URL,
            successUrl: "http://localhost:9000/" + SUCCESS_URL);
        for (Links link : payment.getLinks()) {
            if (link.getRel().equals("approval_url")) {
                return "redirect:" + link.getHref();
            }
        }
    }

    } catch (PayPalRESTException e) {

        e.printStackTrace();
    }

    return "redirect:/";
}
```

```

@Service
public class PaypalService {

    @Autowired
    private APIContext apiContext;

    public Payment createPayment(Double total, String currency, String method, String intent, String cancelUrl, String successUrl) {
        Amount amount = new Amount();
        amount.setCurrency(currency);
        amount.setTotal(String.valueOf(total));

        Transaction transaction = new Transaction();
        transaction.setAmount(amount);

        List<Transaction> transactions = new ArrayList<>();
        transactions.add(transaction);

        Payer payer = new Payer();
        payer.setPaymentMethod(method.toString());

        Payment payment = new Payment();
        payment.setIntent(intent.toString());
        payment.setPayer(payer);
        payment.setTransactions(transactions);
        RedirectUrls redirectUrls = new RedirectUrls();
        redirectUrls.setCancelUrl(cancelUrl);
        redirectUrls.setReturnUrl(successUrl);
        payment.setRedirectUrls(redirectUrls);
        return payment.create(apiContext);
    }

    public Payment executePayment(String paymentId, String payerId) throws PayPalRESTException{
        Payment payment = new Payment();
        payment.setId(paymentId);
        PaymentExecution paymentExecute = new PaymentExecution();
        paymentExecute.setPayerId(payerId);
        return payment.execute(apiContext, paymentExecute);
    }
}

```

Por último, si obtenemos una url y/o conexión correcta se ejecutará otro método llamado *successPay()*, donde una vez el pago sea finalizado, se obtendrá una respuesta en JSON con los datos de la transacción.

También se confirmará el pedido y te llevará de forma exitosa a INDEX.

```

@GetMapping(value = @value CANCEL_URL)
public String cancelPay() {
    return "cancel";
}

@GetMapping(value = @value SUCCESS_URL)
public String successPay(@RequestParam("paymentId") String paymentId, @RequestParam("PayerID") String payerId) {
    Pedido pedido1 = servicePedido.selectPedido(servicePedido.obtenerID());
    try {
        Payment payment = paypalService.executePayment(paymentId, payerId);
        System.out.println(payment.toJSONString());
        if (payment.getState().equals("approved")) {
            /**
             * una vez el cliente ha pagado correctamente, el pedido pasará a estar confirmado.
             */
            pedido1.setConfirm(true);
            /**
             * Aquí enviamos un correo electrónico al cliente.
             */
            emailService.sendSimpleEmail(pedido1.getCliente().getEmail(), subject: "Tu pedido ha sido realizado",
                body: "Tu pedido se ha enviado correctamente a las " + pedido1.getFecha() + " con un importe de: "
                    + pedido1.getTotalPedido() + "€");
            return "/index";
        }
    } catch (PayPalRESTException e) {
        System.out.println(e.getMessage());
    }
    return "redirect:/";
}

```

```
{
    "id": "PAYID-MKFCFYI0AF36999KG856314S",
    "intent": "sale",
    "payer": {
        "payment_method": "paypal",
        "status": "VERIFIED",
        "payer_info": {
            "email": "sb-61zqp16541917@personal.example.com",
            "first_name": "John",
            "last_name": "Doe",
            "payer_id": "9E6PHSCC5KV36",
            "country_code": "ES",
            "shipping_address": {
                "recipient_name": "John Doe",
                "line1": "calle Vilamar 76993- 17469",
                "city": "Albacete",
                "country_code": "ES",
                "postal_code": "02001",
                "state": "Albacete"
            }
        }
    },
    "cart": "62X76229395445511",
    "transactions": [
        {
            "transactions": []
        }
    ]
}
```

The screenshot shows the homepage of the website. At the top, there's a navigation bar with links to 'BIENVENIDOS', 'NOSOTROS', 'APADRINA', 'TIENDA', 'CARRITO', and 'Otros marcadores'. On the left, there's a logo for 'Mel Cuarta Generación' and a sidebar with links to 'Quienes Somos', 'Tienda Online', and 'Apadrina Una Colmena'. The main content area features a large image of a honey dipper with honey dripping from it.

Por otro lado, una vez que se confirma el pedido también se enviará un correo electrónico al cliente sobre la información de su pedido.

```
@Service
public class Email {
    @Autowired
    private JavaMailSender mailSender;

    public void sendSimpleEmail(String toEmail,
                                String subject,
                                String body) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("mielcuartageneracion@gmail.com");
        message.setTo(toEmail);
        message.setText(body);
        message.setSubject(subject);
        mailSender.send(message);
    }
}
```

Para empezar se añaden las dependencias y la configuración necesaria al fichero *properties*.

Se utilizará una clase llamada Email a la cual le asignamos la *annotation@Service*, El envío de correos electrónicos se realiza mediante una librería MailSender que Spring te proporciona.

También se ha procedido a crear una cuenta de gmail de la empresa donde se enviarán los correos a todos los clientes.

The screenshot shows the Gmail inbox. There are three notifications at the top: 'Principal' (with 1 new), 'Social' (with 1 new), and 'Promociones' (with 50 new). Below, there's a notification for a new email from 'mielcuartageneracion@gmail.com' with the subject 'Tu pedido ha sido realizado'. The message content is: 'Tu pedido se ha enviado correctamente a las 2022-06-09 con un importe de: 15.99€'.

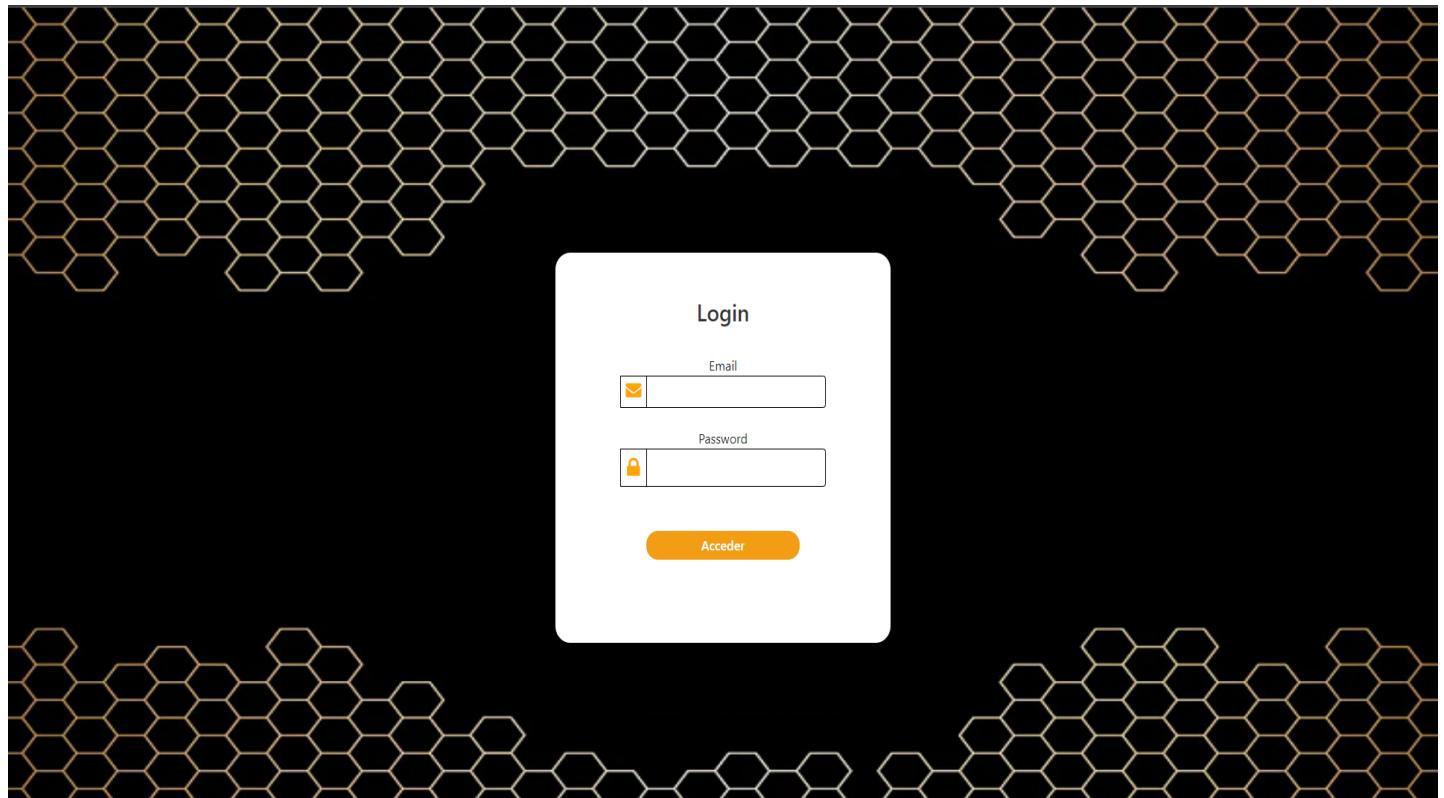
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=mielcuartageneracion@gmail.com
spring.mail.password=swycgmitiwupmmfup
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

Aquí se puede ver como las notificaciones de los correos electrónicos llegan correctamente.

10. Manual de Usuario - Tienda:

Parte del Panel de Administrador:

- Login para Usuarios Administradores:



- Panel de administración:

The screenshot shows the administrator dashboard for "Miel Cuarta Generación". On the left is a sidebar with navigation links: Cliente (Listar Cliente, Nuevo Cliente), Producto (Listar Producto, Nuevo Producto), Categoría (Listar Categoría), and Pedido. The main area has a header "Categorías de la Empresa" and a sub-header "Miel Cuarta Generación". It displays a table of categories:

ID	Nombre	Editar	Borrar
1	bote-miel	Editar	Borrar
2	regalos	Editar	Borrar
3	velas	Editar	Borrar

Parte de la Tienda Online:

- La vista de la Tienda

Miel Cuarta Generación

BIENVENIDOS NOSOTROS APADRINA TIENDA CARRITO

miel flores 1kg
6.99€
[Ver Descripción](#)

miel retama 1kg
8.99€
[Ver Descripción](#)

Bolsita dorada de miel 30g
1.65€
[Ver Descripción](#)

[Ver Descripción](#)

[Ver Descripción](#)

Miel Cuarta Generación

BIENVENIDOS NOSOTROS APADRINA TIENDA CARRITO

miel flores 1kg
6.99€
Categoría: bote-miel
Descripción: Miel Multiflora con Eucalipto. Muy energética, ayuda a combatir el cansancio.* Es rica en una gran variedad de vitaminas, minerales y
Peso: 1.0 Kg

Miel Cuarta Generación

BIENVENIDOS NOSOTROS APADRINA TIENDA CARRITO

Productos de la Empresa

Producto	Nombre	Precio	Cantidad	Subtotal	Total Peso
	miel flores 1kg	6.99 €	1	6.99 €	1.0 kg

Categorías

bote-miel
regalos
velas

Total Carrito

Envío	8.0 €
Total Pedido	14.99 €

[Pagar](#)

Nuevo cliente

Nombre
5928553w

Apellidos
andrea

Nombre Empresa
c/los huertos

País
España

Dirección
usu

Apartamento
1234

Provincia

Localidad
6535353

Código Postal
0

Teléfono
0

Email
6535353

Total Carrito

Producto	miel flores 1kg
Subtotal	6.99
Envío	8.0 C
Total Pedido	14.99 C

Pagar



Pagar con PayPal

Al disponer de una cuenta PayPal, las compras que cumplen los requisitos estarán cubiertas por nuestra política de Protección del comprador. También tendrá la posibilidad de activar nuestro programa Reembolso de gastos de devolución. [Consultar condiciones](#)

sb-61zqp16541917@personal.example.com

.....

Iniciar sesión

[¿Tiene problemas para iniciar sesión?](#)

O

Pagar con tarjeta de débito o crédito

[Cancelar y volver a Test Store](#)

Español | English

Test Store



14,99 EUR

Hola, John.

Enviar a

John Doe
calle Vilamar 76993-17469, 02001 ALBACETE, ALBACETE

[Cambiar](#)

Pagar con

 Rabobank Nederland
Corriente ****0598

14,99 EUR

Forma de pago de garantía: VISA ****4242

 Establecer como mi forma de pago preferida

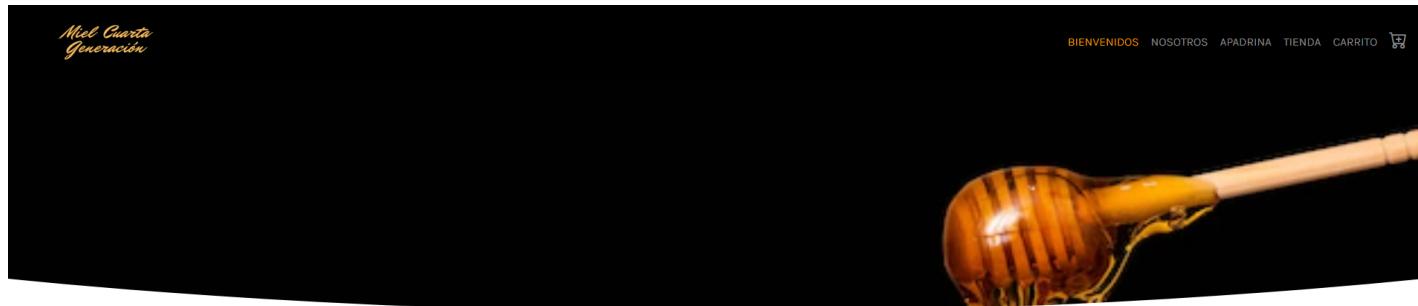
 The Bank Card Platinum Rewards
****4242

 Visa
Crédito ****1619

[+ Añadir tarjeta](#)**Continuar**

Podrá revisar el pedido antes de completar la compra.

Vistas Bienvenidos,Nosotros y Apadrina:



Quienes Somos

Somos una empresa familiar dedicada a la apicultura, cuidando nuestras propias colmenas hace más de 200 años. Nuestra explotación está ubicada en Castilblanco en un entorno Natural y privilegiado.

Conocenos



Tienda Online

Puede realizar su pedido en nuestra página web. También puede contactar con nosotros para solicitar cualquier consulta por cualquier red social o llamando a nuestros teléfonos de atención al cliente.

Ir a la tienda

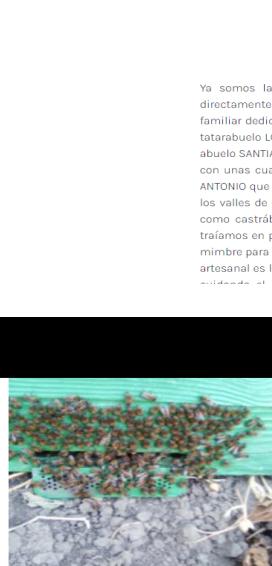


Apadrina Una Colmena

Desde tu casa y sin picaduras, podrás apadrinar tu propia colmena, además obtienes miel de dicha colmena. Si eres amante de la naturaleza yquieres participar es muy fácil, solo tienes que elegir una modalidad.

Apadrina

Bienvenidos:



Miel Cuarta Generación

BIENVENIDOS NOSOTROS APADRINA TIENDA CARRITO



Einstein dijo que si las abejas desaparecen, al hombre le quedarían 4 años en la Tierra. Justo hace unos años están desapareciendo una gran cantidad de abejas debido a diversos factores como la contaminación, pesticidas y una mayor incidencia de algunas enfermedades como la varroa, ayúdanos a mantener el medio ambiente y el equilibrio del ecosistema.

Desde tu casa y sin picaduras, podrás apadrinar tu propia colmena, además obtienes miel de dicha colmena apadrinada. Si eres amante de la naturaleza yquieres participar en este proyecto es muy fácil, solo tienes que elegir una modalidad, los apadrinamientos tienen duración de 1 año.

Modalidades

COLMENA INDIVIDUAL:

Puedes personalizar el nombre de tu colmena.

Recibirás 5 kilos de miel al año.

a vela de cera pura de abeja decorada.

Cuota de 60 euros al año.

Recibirás fotos y videos de tu colmena apadrinada.

Tlf de contacto : 606900157, Envío incluido

Modalidades

COLMENA FAMILIAR:

Pueden personalizar varios nombre en una colmena.

Recibirás 5 kilos de miel al año.

a vela de cera pura de abeja decorada.

Cuota de 60 euros al año.

Recibirás fotos y videos de tu colmena apadrinada.

Tlf de contacto : 606900157, Envío incluido

11. Manual del Programador- Android:

Para el desarrollo de la aplicación de Android hemos utilizado una librería llamada **Retrofit 2** y una API REST creada con spring boot.

Retrofit es un cliente HTTP que nos permite conectarnos a un servicio web REST y realizar peticiones (GET,POST,etc)

```
public static void conexion() {  
    Retrofit retrofit = new Retrofit.Builder()  
        .baseUrl("http://192.168.0.21:9000/")  
        .addConverterFactory(GsonConverterFactory.create())  
        .build();  
    crudInterface = retrofit.create(CRUDInterface.class);  
}
```

En primer lugar, para realizar la conexión correctamente a nuestra API REST agregamos en baseUrl la dirección IP en la que está alojada nuestra API, la cual podemos averiguar fácilmente realizando un ipconfig en el cmd.

A continuación, creamos un objeto de nuestra interfaz: *CRUDInterface* y le asignamos el método *retrofit.create()* que recibe como parámetro nuestra interfaz. que nos servirá para gestionar cada una de las peticiones a nuestra API.

```
public interface CRUDInterface {  
  
    @GET("/listCliente")  
    Call<List<Cliente>> findAllCliente();  
    @GET("/cliente/{id}")  
    Call<Cliente> getCliente(@Path("id") int id);  
    /**  
     * Fragment AllPedidos  
     * @return todos los pedidos where enviado = true  
     */  
    @GET("/listPedido")  
    Call<List<Pedido>> findAllPedido();  
    /**  
     * Fragment Pendientes  
     * @return todos los pedidos where enviado = false  
     */  
    @GET("/pedidoConfir")  
    Call<List<Pedido>> getConfir();  
    @GET("/infCliente/{id}")  
    Call<List<LineaPedido>> selectLineasPendiente(@Path("id") int id); //inf cliente  
    @PUT("/modiPedido/{id}")  
    Call<Pedido> updatePedido(@Body Pedido pedido);  
    @GET("/login/{email}/{password}")  
    Call<Usuario> loginByPass(@Path("email") String email,@Path("password") String pass);  
}
```

El primer fragmento que debemos explicar es LoginFragment, que consta de 5 métodos.

```
private SharedPreferences preferences;
private SharedPreferences.Editor editor;
private String llave = "sesion";
@Override
public void onStart() {
    super.onStart();
    inicializar();
    pulsarBoton();
}
```

En primer lugar comentar la implementación de SharedPreferences para el guardado de la sesión.

El método *pulsarBoton()*, se encarga de controlar si la sesión está guardada, en ese caso, se entrará directamente en la aplicación. En cambio, si la sesión no está guardada, el administrador tendrá que llenar los campos email y contraseña correctamente y se procederá a ejecutarse otro método que será el encargado de comprobar si ese usuario existe en la base de datos.

```
public void pulsarBoton(){
    if(revisarSesion()){
        getActivity().getSupportFragmentManager().beginTransaction().replace(R.id.container, new HomeFragment()).addToBackStack("").commit();
    }
    btLogin.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if(TextUtils.isEmpty(etEmail.getText().toString()) || TextUtils.isEmpty(etPassword.getText().toString())){
                Toast.makeText(getApplicationContext(), "Introduce la contraseña/password", Toast.LENGTH_SHORT).show();
            } else{
                login(etEmail.getText().toString(),etPassword.getText().toString());
            }
        }
    });
}
public void login(String email,String pass){...}
private boolean revisarSesion() { return this.preferences.getBoolean(llave, false); }
private void guardarSesion(boolean checked) {
    editor.putBoolean(llave, checked);
    editor.apply();
}
```

El siguiente paso es crear una llamada a la cuál le indicamos el tipo, que será `Usuario` y le asignamos el método `loginByPass()` de nuestra interfaz `crudInterface` lo cual devuelve un `Call`.

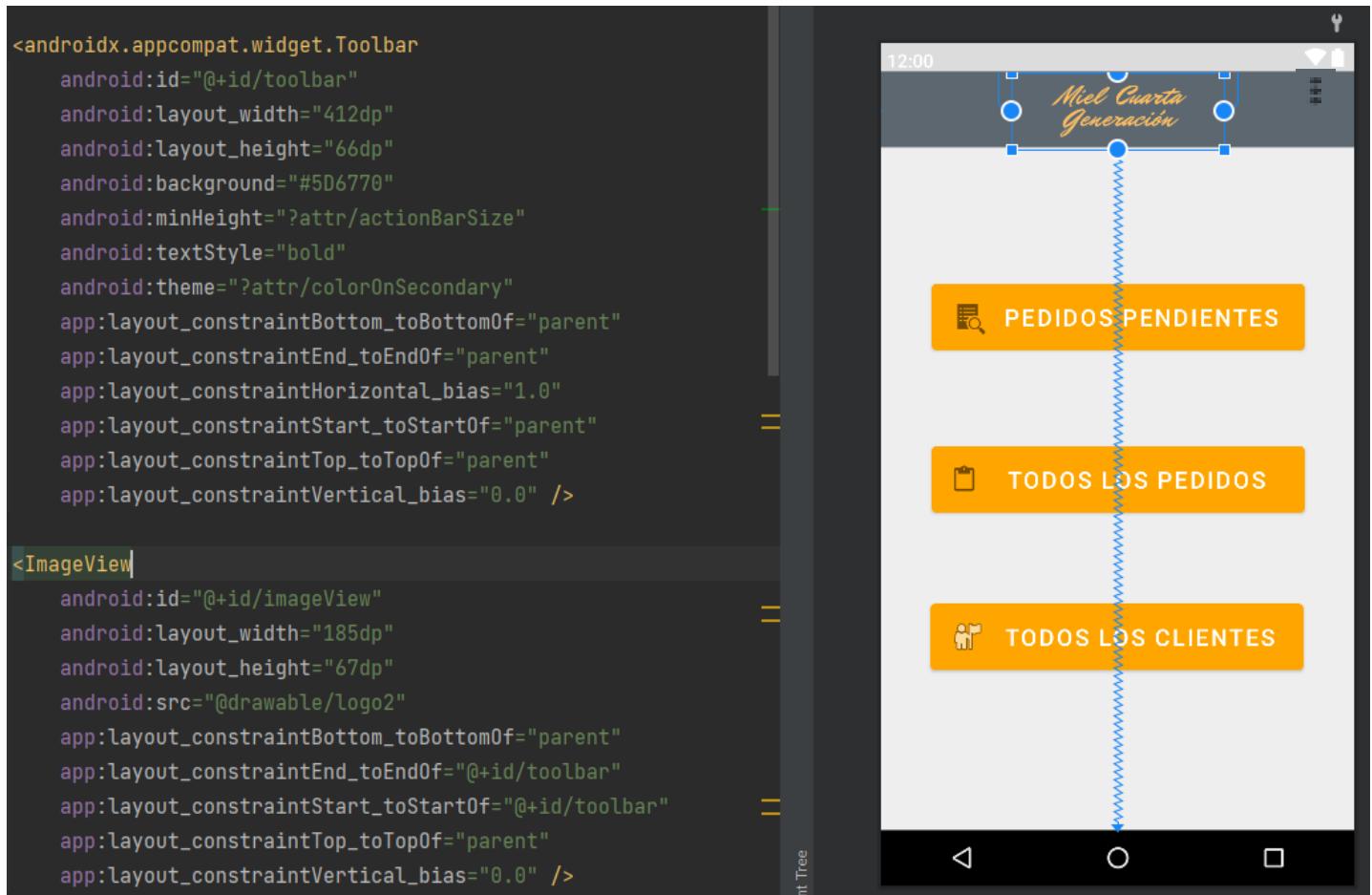
Para realizar la llamada ejecutamos el método `enqueue()` que recibe un objeto tipo `Callback` como parámetro, este ejecuta una clase anónima que tiene dos métodos:

El primero es el método `onResponse()`, que se ejecuta si la llamada a la API se realizó con éxito y en ese caso, obtendrás el usuario que se ha logueado y se iniciará sesión correctamente. Adicionalmente, se comprueba si el usuario administrador marcó la opción de guardar sesión o no (activando o no el check box).

Por otro lado, el método `onFailure()` que se ejecutará si algo falla en nuestra petición.

```
public void login(String email,String pass){
    conexion();
    Call<Usuario> call = crudInterface.loginByPass(email,pass);
    call.enqueue(new Callback<Usuario>() {
        @Override
        public void onResponse(Call<Usuario> call, Response<Usuario> response) {
            if(!response.isSuccessful()){
                Log.d("response err", response.message());
                return;
            }
            usuario= response.body();
            Log.d("usuario", usuario.toString());
            guardarSesion(checkBox.isChecked());
            getActivity().getSupportFragmentManager().beginTransaction().replace(R.id.container, new HomeFrgmen
        }
        @Override
        public void onFailure(Call<Usuario> call, Throwable t) {
            Log.d("throw err", t.getMessage());
        }
    });
}
```

El resto de fragmentos siguen la misma dinámica respecto a las llamadas a la API. Una vez pasamos al HomeFragment, algo a destacar es la creación del propio Toolbar que tiene toda la aplicación, con el logo de la empresa en el centro.



```
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
    View view = inflater.inflate(R.layout.fragment_home, container, false);  
    toolbar = view.findViewById(R.id.toolbar);  
    AppCompatActivity activity = (AppCompatActivity) getActivity();  
    activity.setSupportActionBar(toolbar);  
    activity.getSupportActionBar().setTitle("");  
    setHasOptionsMenu(true);  
    return view;  
}
```

Esto se ha conseguido mediante la implementación de las siguientes líneas de código en cada fragmento.

Una vez se seleccione una de las 3 opciones, la dinámica de cada uno de ellos será prácticamente idéntica, como ya se mencionó anteriormente. Solo se utilizarán 2 métodos para realizar toda la lógica de cada fragmento, uno de ellos será inicializar los TextView o Recycler que se van a necesitar, y por otro lado, el método encargado de realizar la petición correspondiente a la API.

Por otro lado, si se elige la opción de PedidosPendientes, se cargará el fragmento *PendienteFragment*, el cual realizará una petición a la API donde se obtendrá una lista con todos los pedidos que no estén confirmados (*where enviado = false*).

```
private void getPedidoConfig() {
    conexion();

    Call<List<Pedido>> call = crudInterface.getConfirm();
    call.enqueue(new Callback<List<Pedido>>() {
        @RequiresApi(api = Build.VERSION_CODES.N)
        @Override
        public void onResponse(Call<List<Pedido>> call, Response<List<Pedido>> response) {
            if (!response.isSuccessful()) {
                Log.d("tag: " + "response err", response.message());
                return;
            }
            listaPedidos = response.body();
            recycler.setAdapter(new RecyclerPedidoPending(listaPedidos, getContext()));
            recycler.setLayoutManager(new LinearLayoutManager(getContext()));
            for (Pedido p : listaPedidos) {
                Log.d("tag: " + "pedidos sin confirmar", p.toString());
            }
        }

        @Override
        public void onFailure(Call<List<Pedido>> call, Throwable t) {
            Log.d("tag: " + "throw err", t.getMessage());
        }
    });
}
```

La dinámica de estos fragmentos en todo momento será la de guardar en el *viewModel* el objeto Pedido ya que en el próximo fragmento *ConfirmarFragment*, se utilizará para obtener el IdPedido del pedido que se seleccione en el recycler, y así poder cargar la información correspondiente.

```
@Override
public void onStart() {
    super.onStart();
    instanciar();
    getLineas(viewModelCP.getPedido().getIdPedido());
    pulsarBoton();
}
```

```
@Override
public void onBindViewHolder(@NonNull OpcionesEncargosViewHolder holder, int position) {

    holder.txtValor.setText(opciones.get(position).getFecha() + " " + opciones.get(position).getCliente().getNombre());
    holder.cardViewEmpresa.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            viewModelCP.setPedido(opciones.get(position));
            ((FragmentActivity) context).getSupportFragmentManager().beginTransaction().replace(R.id.container, new ConfirmarFragment());
        }
    });
}
```

Algo a destacar respecto al resto de fragmentos es la opción de confirmar el pedido mediante un botón que se le ofrecerá al usuario. Se realizará mediante una petición PUT a la API. Cuando se pulse el botón se mandará una notificación que mostrará: *Pedido Confirmado*. Por otro lado, se ocultarán los botones de Confirmar y Ver Cliente y solo se podrá un botón que me llevará al fragmento HomeFragment.

Por último, tanto *AllPedidosFragment* como *AllClientesFragment*, siguen la misma dinámica por lo que la explicación se podrá encontrar en el propio proyecto.

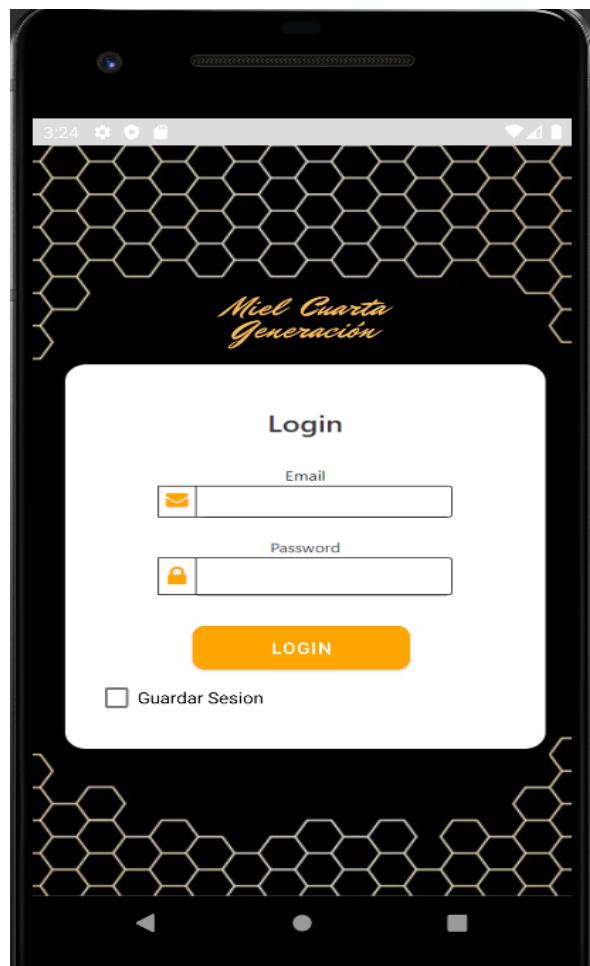
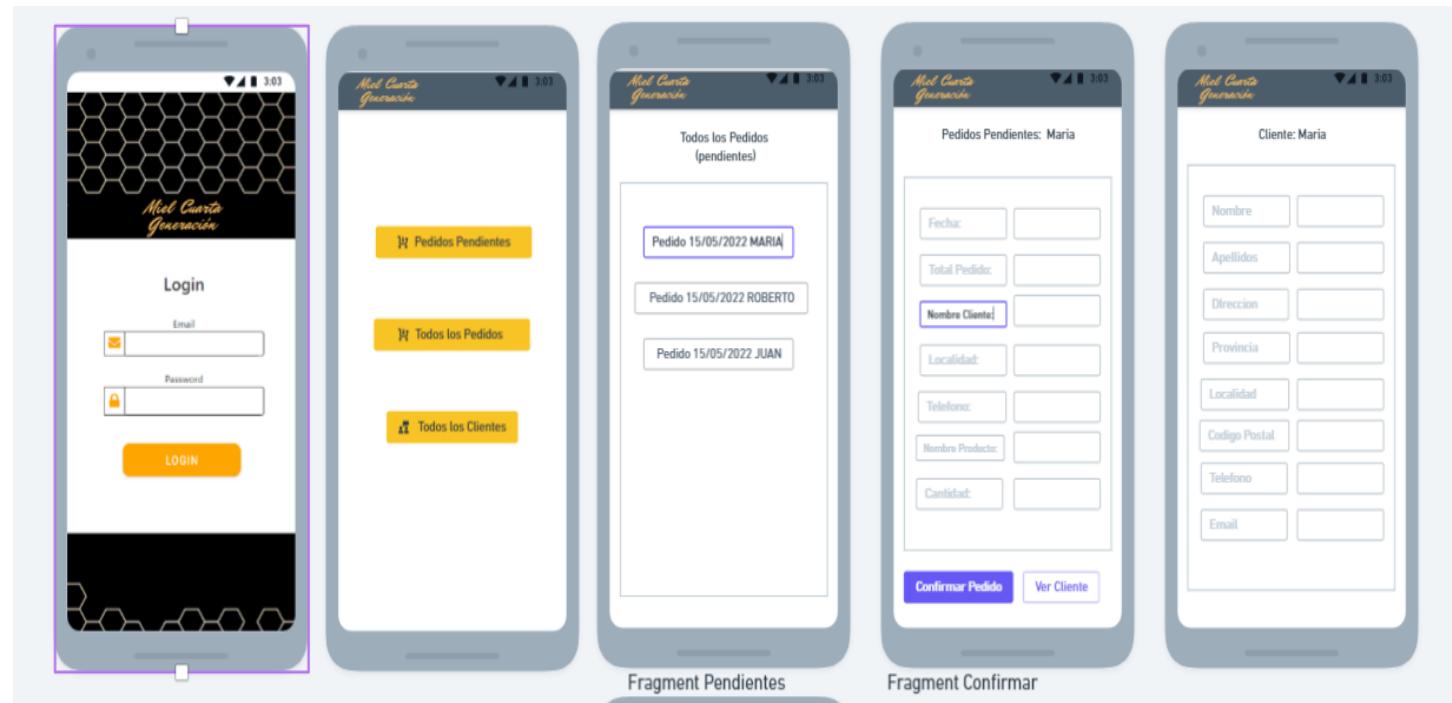
```
@PutMapping("/modiPedido/{id}")
public void updatePedido(@RequestBody Pedido pedido) {
    pedido.setEnviado(true);
    pedidoService.edit(pedido);
}
```

```
btHome.setVisibility(View.GONE);
btConfirmar.setOnClickListener(new View.OnClickListener() {
    @SuppressLint("ResourceAsColor")
    @Override
    public void onClick(View view) {
        update(viewModelCP.getPedido());
        btConfirmar.setVisibility(View.GONE);
        btVerCliente.setVisibility(View.GONE);
        Snackbar mySnackbar = Snackbar.make(view, "PEDIDO CONFIRMADO", Snackbar.LENGTH_LONG);
        mySnackbar.show();
        btHome.setVisibility(View.VISIBLE);
    }
});
```

12. Manual del Usuario - Android:

Esta app de Android se ha desarrollado para facilitar el trabajo, en este caso, de los usuarios administradores, los cuales necesitan la mayor parte del tiempo revisar información sobre los clientes o sobre los pedidos pendientes a realizar.

Este es un esquema realizado mediante una plataforma (*whimsical*) para hacernos una idea inicial de cómo quedará nuestra aplicación Android.



Al abrir la aplicación de Android se nos muestra un login con el que se tendrá que iniciar sesión si quieras seguir navegando por la app.

La aplicación hace uso de *SharedPreferences* (una clase proporcionada por Android) para guardar la sesión del usuario correctamente, y así facilitarle la accesibilidad.

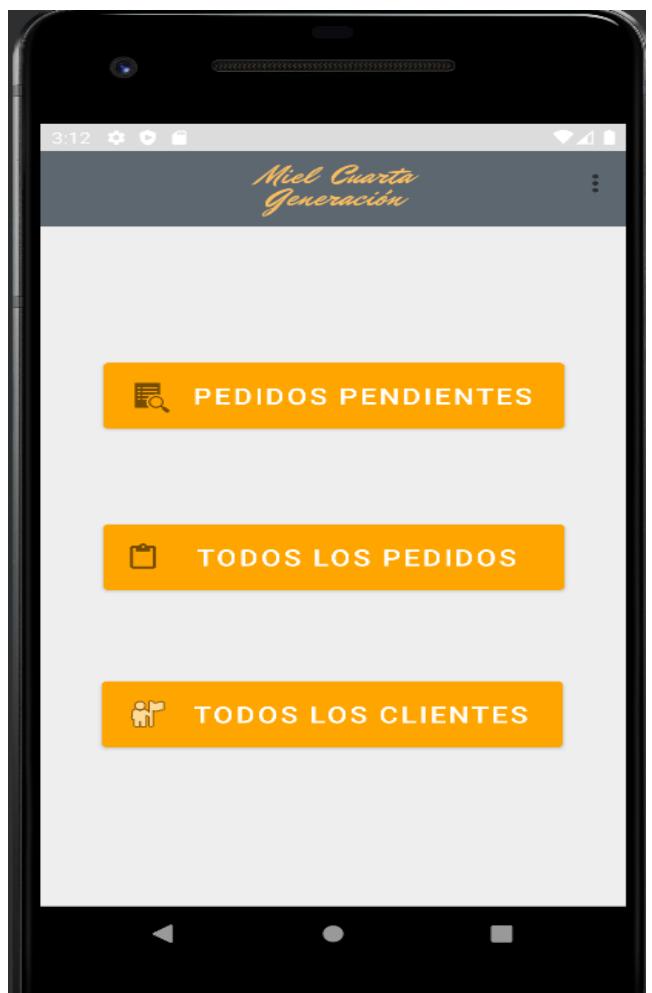
Este es el menú principal que se ha desarrollado para ser lo más intuitivo y práctico posible, y así poder reducir el tiempo de los administradores al buscar la información que necesiten en cada momento.

El funcionamiento de la aplicación es sencillo, pulsando en cualquiera de las opciones se accederá a un recycler que mostrará una lista de los pedidos o de los clientes correspondientes.

Una vez se pulse en cualquiera de las opciones del recycler se cargará un fragmento de información del pedido/cliente.

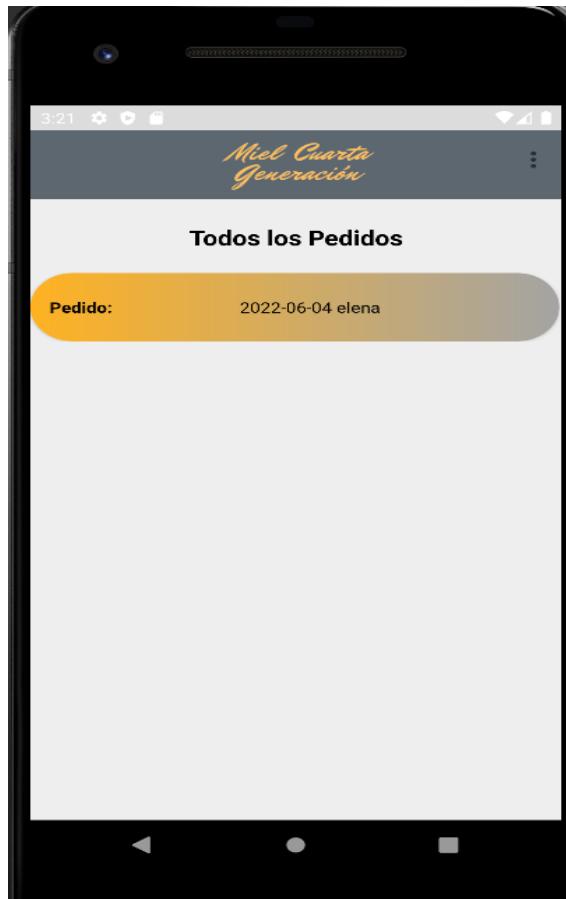
En el caso de los pedidos, se tendrá un botón para ver la información del cliente que realizó ese pedido.

Por último, si el usuario administrador elige la opción de pedidos pendientes, podrá ver los pedidos que tiene que realizar. Una vez el administrador haya terminado de preparar el pedido, tendrá la opción de confirmarlo mediante un botón, en ese momento desaparecerá de la lista de pedidos pendientes y pasará a la lista de todos los pedidos.



Si se pulsa en el botón *Confirmar* se mandará una notificación de: *Pedido Confirmado*, como se puede ver en la captura inferior.

Y por otro lado, no se permitirá acceder a las opciones anteriores y solo se podrá volver al fragmento de inicio y poder seguir consultando la información que nuevamente se necesite.



En el caso de los Todos los Pedidos será el mismo sistema, y tan solo se podrá ver la información del cliente que realizó el pedido.



Por último, en la parte de Ver todos los Clientes, como se puede observar, sigue la misma dinámica y muestra el recycler correspondiente y la información del cliente que se haya seleccionado.

