

DOCUMENTACIÓN BOARDALO TPVV

Grupo 15

Cristian Andrés Córdoba Silvestre
Daniel Ripoll Sánchez
Luis Alfonso Jiménez Rodríguez

Índice

| | |
|--|-----------|
| 1. Descripción general | 2 |
| 2. Descripción técnica del sistema..... | 2 |
| 2.1 Tecnologías | 2 |
| 2.2 Arquitectura | 3 |
| 3. INTEROPERACIÓN: API de Realizar Pago..... | 6 |
| 3.1 INTRODUCCIÓN | 6 |
| 3.2 PROYECTO BASE PARA LA INTERCONEXIÓN..... | 7 |
| 3.3 OBJETIVOS DE LA INTERCONEXIÓN | 10 |
| 3.4 DESCRIPCIÓN DE LA ARQUITECTURA EMPLEADA | 10 |
| 3.5 DETALLES DE LA COMUNICACIÓN SERVER TO SERVER Y EL USO DE PROXIES..... | 11 |
| 3.6 CLASES Y COMPONENTES RELEVANTES | 15 |
| 3.7 RESUMEN DE LOS BENEFICIOS Y MOTIVACIÓN DE LA COMUNICACIÓN SERVER TO SERVER Y DEL USO DE PROXIES | 18 |
| 3.8 CONSIDERACIONES ADICIONALES | 19 |
| 3.9. CONCLUSIONES | 23 |
| 4. Metodologías y planificación | 25 |
| 5. Diagrama EER | 28 |
| 6. Diseño de la aplicación | 28 |
| 6.1 Mockups | 28 |
| 6.2 Patrones de diseño:..... | 38 |
| 7. Problemas encontrados y su solución | 40 |
| 8. Aspectos extras de implementación | 42 |
| 8.1 Docker compose:..... | 42 |
| 8.2 Middlewares: | 43 |
| 8.3 Tests Unitarios: | 44 |
| 9. Mejoras y ampliaciones | 44 |

1. Descripción general

Link del repositorio de desarrollo del proyecto:

<https://github.com/cacs2-ua/PRIVADO-TPVV-IW-Grupal2.git>

La aplicación de **Boardalo** va a ser una plataforma de pago virtual, donde el cliente podrá crear una orden de pago que el usuario final realizará con una tarjeta. El sistema contará con una parte privada para el cliente, donde podrá ver los pagos realizados en su comercio, viendo la información de estos. También cuenta con un programa de incidencias, por si en algún momento surge algún problema con un pago o una duda con el aplicativo en sí. Los técnicos se encargarán de solucionar la duda y ayudar en lo posible al comercio.

2. Descripción técnica del sistema

2.1 Tecnologías

En el desarrollo de una Plataforma de Pago Virtual (TPVV), resulta crítico asegurar la solidez de la arquitectura, la seguridad en el intercambio de información financiera y la capacidad de procesar formularios de pago de forma robusta y escalable. En este contexto, Spring Boot y Thymeleaf surgen como herramientas idóneas, gracias a sus implementaciones nativas relacionadas como **Spring Security**, librería que ha resultado imprescindible para el desarrollo de nuestra aplicación para manejar el envío de la API Key de los comercios por medio de la cabecera **Authorization**.

2.1.2 PUNTOS CLAVE

2.1.2.1 Gestión de Formularios de Pago y Validaciones

Thymeleaf proporciona un motor de plantillas orientado a la generación dinámica de formularios y a la validación de los campos (p. ej., número de tarjeta, fecha de caducidad, CVC).

El flujo de envío de datos sensibles (tarjeta e importe) se integra de manera natural con los controladores de Spring Boot, facilitando la incorporación de restricciones y validaciones propias de un sistema de TPVV (formato de tarjeta, longitud, patrones de caducidad).

2.1.2.2 Seguridad e Integración con API Keys

Spring Boot simplifica la configuración de seguridad y la integración de filtros de autenticación mediante Spring Security, lo que permite validar la API Key en cada petición.

Un TPVV requiere un control exhaustivo de acceso (verificación de comercios, control de roles de usuario, protección de endpoints mediante **middlewares**¹, ...), y la combinación de Spring Boot con Thymeleaf hace viable incluir lógicas de seguridad que intercepten y gestionen los datos con las mejores prácticas (CSRF, validaciones, etc.).

2.1.2.3. Arquitectura Escalable y Legibilidad del Código

El arranque rápido que ofrece Spring Boot facilita la organización modular de componentes como servicios, repositorios y entidades, algo esencial cuando se gestionan varios comercios, sus pedidos y estados de pago.

Para un proyecto de TPVV, donde se maneja un alto volumen de transacciones y se requieren respuestas veloces, la estructura de Spring Boot permite escalar gradualmente y mejorar la mantenibilidad del código.

2.1.2.4. Soporte a Transacciones y Persistencia

Un TPVV debe asegurar la integridad de las operaciones financieras, por lo que se requiere un manejo transaccional robusto. Spring Boot proporciona soporte nativo para transacciones (a través de Spring Data y **JPA**²), facilitando la gestión de commits, rollbacks y la consistencia de la base de datos.

El uso de Thymeleaf en las vistas no interfiere con el motor transaccional de Spring, pues se limita a la capa de presentación, lo que garantiza una separación clara de responsabilidades.

2.2 Arquitectura

Springboot es un framework que permite desarrollar aplicaciones de carácter empresarial, lo cual implica que se trabaje siguiendo un desarrollo modular, mantenible y escalable. Para lograr este objetivo, SpringBoot utiliza un enfoque denominado **arquitectura multilayer**, el cual es común en la mayoría del resto de frameworks del **backend**: Laravel, .Net, Symphony, etc. En el caso de Spring Boot, este enfoque está compuesto de las siguientes capas:

¹ Un **middleware** es un software intermediario que se utiliza para interceptar solicitudes y respuestas, permitiendo funciones como autenticación, registro de logs o gestión de excepciones antes de que lleguen al controlador o al cliente.

² **JPA** (Java Persistence API) es una especificación de Java para mapear objetos a bases de datos relacionales (ORM). Permite gestionar datos mediante entidades Java, simplificando operaciones como consultas, inserciones, actualizaciones y eliminaciones.

2.2.1 Capa de Presentación (o capa Web)

- Incluye los Controladores (ubicados en el paquete `tpvv.controller`) que se encargan de recibir las peticiones HTTP y de preparar las respuestas adecuadas.
- Se apoya en plantillas **Thymeleaf** para renderizar las vistas y en recursos estáticos (CSS, JS) que se encuentran en la carpeta `static/` y en la subcarpeta `templates/`

2.2.2 Capa de Negocio (o capa de Servicios)

- Se encuentra en los `services` (`tpvv.service`). Aquí se definen las reglas de negocio y la lógica principal del sistema.
- Los servicios coordinan la comunicación entre la capa de presentación y la capa de persistencia, realizando validaciones o cálculos específicos cuando corresponde.

2.2.3 Capa de Persistencia

- Se apoya en los `repositories` (**`tpvv.repository`**), donde se manejan las operaciones CRUD frente a la base de datos (**PostgreSQL en el caso de nuestro proyecto**).
- Utiliza el modelo de datos definido en las entidades del paquete **`tpvv.model`** (por ejemplo, **Comercio**, **EstadoIncidencia**, etc.).

A parte de las capas típicas que se incluyen en la **arquitectura multilayer**, nosotros hemos implementado también una capa extra de **seguridad**. Como veremos más adelante, esta capa adicional resultará imprescindible para la interconexión, debido al manejo de la API Key por la cabecera **Authorization**.

2.2.4 Capa de Configuración y Seguridad

- Abarca los componentes encargados de configurar el proyecto: archivos de propiedades (application.properties), clases de configuración (tpvv.config.*) y la capa de seguridad (tpvv.security.*), donde se gestionan filtros y políticas de acceso (ej. ApiKeyAuthFilter, SecurityConfig). En nuestro caso, estos dos últimos archivos se encargan principalmente de manejar el filtro de la API Key.

2.2.5 Modelo Vista Controlador (MVC)

En Spring Boot, de la arquitectura multilayer también se deduce la implementación del patrón clásico **MVC (Modelo Vista Controlador)**. La implementación base de este patrón en Spring Boot sigue la siguiente estructura característica:

1. Modelo

- Las clases modelo se localizan en el paquete **model**. Estas clases representan la estructura de la información que se maneja en la aplicación (por ejemplo, **Comercio**, **EstadoIncidencia**, **Pago**, etc.).
- Adicionalmente, se definen **Data Transfer Objects (DTO)** en el paquete **dto**, el cual permite intercambiar datos de forma segura y clara entre capas, sin exponer la estructura interna de las entidades.

2. Vista

- Las vistas se gestionan principalmente mediante **Thymeleaf** y se ubican en **src/main/resources/templates/**.
- Esta capa se encarga de la representación visual. Cuando un controlador retorna una vista, Thymeleaf procesa la plantilla HTML, inyectando la información proveniente del modelo.

3. Controlador

- Los controladores residen en el paquete **controller**.
- Cada controlador atiende las peticiones (**GET, POST, etc.**) definidas mediante anotaciones como **@GetMapping** o **@PostMapping**.
- Se encarga de: recibir y validar parámetros, invocar métodos de la capa de servicios para la lógica de negocio y seleccionar la vista (plantilla Thymeleaf) o la respuesta REST adecuada, retornándosela al cliente.

Esta implementación MVC asegura que se cumpla el principio de **separación de responsabilidades**:

- El **Modelo** encapsula la lógica de datos.
- La **Vista** se centra en la presentación de la información.
- El **Controlador** intermedia y orquesta el flujo de datos entre ambos.

3. INTEROPERACIÓN: API de Realizar Pago

3.1 INTRODUCCIÓN

Nuestro proyecto expone la funcionalidad de permitir a proyectos externos el realizar pagos a través de nuestra pasarela de pago. Para ello, se exponen dos endpoints principales:

1. **“/pago/form”**: endpoint encargado de mostrar los detalles de la compra, así como mostrar el formulario para completar el pago.



The screenshot displays the 'TPVV BoarDalo' payment interface. It is divided into two main sections: 'Datos de la Compra' (Purchase Data) and 'Pago con tarjeta' (Card Payment).

Datos de la Compra:

| | |
|---------------|------------------|
| Importe: | 888,89 € |
| Comercio: | Tienda Online v2 |
| Nº de pedido: | TICKET-888 |
| Fecha: | 09/09/2029 |
| Hora: | 13:45 |

Pago con tarjeta:

Nombre Completo:

Nº Tarjeta:

Caducidad:

Cód. Seguridad:

Acceptar

Figura 1: Datos de la pasarela de Pago

2. **“/pago/realizar”**: endpoint utilizado para registrar un pago en la base de datos del TPVV a partir de los datos introducidos. Este método se encarga de manejar la recepción de la API Key pasada por cabecera HTTP para obtener el comercio asociado a la hora de registrar el pago

3.2 PROYECTO BASE PARA LA INTERCONEXIÓN

Unas de nuestras preocupaciones iniciales era el cómo poder probar el correcto funcionamiento de la funcionalidad de “**realizar pago**” sin tener que esperar a que estuviese completada la implementación del resto de grupos. En base a esta motivación, tomamos la decisión de construir un proyecto base independiente del TPVV. Dicho proyecto simularía una **tienda online**, en la que se muestra la información de un pedido junto con un botón de “**Finalizar compra**”. Cuando dicho botón es pulsado, se llama al endpoint “**pago/form**” del TPVV para así mostrar los detalles de la compra y el formulario para completar el pago.



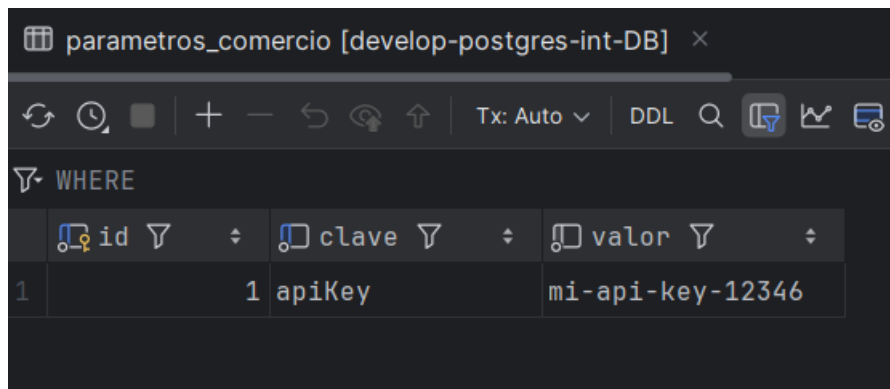
Figura 2: Información del pedido en el proyecto base que actúa como cliente

De esta manera, tenemos dos proyectos a considerar:

1. El proyecto **TPVV (plataforma de pago virtual)**, que opera como servidor y expone servicios de pasarela de pago.
2. El proyecto que simula una **Tienda Online**, que actúa como cliente y solicita los servicios de pago al servidor TPVV.

El proyecto del **TPVV** corre en el puerto **8123**, con una URL base de **http://localhost:8123/tpvv/boardalo**. En su base de datos se encuentran diversos comercios, cada uno con una **API Key** única.

El proyecto de la **Tienda Online** se ejecuta en el puerto **8246**, con una URL base de **http://localhost:8246**. Esta tienda está asociada a un comercio en concreto, por lo que dispone de la API Key que el TPVV requiere para validar la autenticación. Dicha Api Key se almacena en una tabla llamada “**parametros_comercio**”.



| | id | clave | valor |
|---|----|--------|------------------|
| 1 | 1 | apiKey | mi-api-key-12346 |

Figura 3: Tabla “parámetros_comercio”

La característica central de la integración radica en que la Tienda envía su **API Key** al TPVV mediante una comunicación **server to server**³. Si la **API Key** concuerda con uno de los comercios registrados en la base de datos del TPVV, el proceso de pago se habilita. De lo contrario, se responde con un error 404.

Con la comunicación **server to server** se logra que la API Key no se exponga **nunca** en el navegador. De esta manera, la API Key no puede obtenerse nunca mediante una inspección del DOM. Esto resulta de vital importancia para evitar ataques de interceptación de información como el **Man-in-the-middle**⁴.

³ La comunicación **server-to-server** es un intercambio directo de datos entre dos servidores sin intervención de un cliente.

⁴ El ataque **man-in-the-middle** es cuando un atacante intercepta y manipula la comunicación entre dos partes sin que ellas lo noten.

Mantener el encapsulamiento completo de la Api Key es un aspecto fundamental a la hora de mantener la confidencialidad y privacidad del comercio. La api key actúa como un identificador único del comercio y sirve para validar y autenticar su identidad.

El comercio podría estar utilizando la misma api key para realizar otros tipos de trámites diferentes al de realizar un pago. De esta forma, si se interceptase por un atacante la api key durante el proceso de un pago, la integridad del comercio se vería gravemente afectada. Esto es así puesto que esto no afectaría únicamente a la propia funcionalidad del pago, sino también al resto de funcionalidades que hiciesen uso de la api key, quedando el comercio en un estado de vulnerabilidad absoluta.

Para la implementación de la comunicación **server to server**, se ha hecho uso de **RestTemplate**, una clase de Spring Boot que facilita la comunicación HTTP entre servidores. Se ha configurado un bean⁵ de **RestTemplate** en la aplicación base de la tienda:

```
package interconnection.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class AppConfig {

    /**
     * Bean de RestTemplate para realizar solicitudes HTTP.
     *
     * @return Una instancia de RestTemplate.
     */
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

Figura 4: Bean de RestTemplate configurado en la aplicación base de la tienda

⁵ Un **Bean** en Spring Boot es un objeto gestionado por el **contenedor de Spring**. Representa un componente de la aplicación que Spring crea, configura y administra automáticamente, facilitando la inyección de dependencias y la modularidad del código.

3.3 OBJETIVOS DE LA INTERCONEXIÓN

1. **Brindar un formulario de pago** al usuario que compra en la Tienda, sirviéndose para ello del TPVV.
2. **Proteger el acceso** a los endpoints del TPVV mediante un mecanismo de autenticación basado en **API Keys**.
3. **Registrar y persistir** la información del pago, incluyendo tarjeta, importe y estado final (aceptado, rechazado o pendiente).
4. **Notificar a la Tienda** el resultado de cada transacción para que esta guarde en su propia base de datos los pedidos completados o pendientes.

3.4 DESCRIPCIÓN DE LA ARQUITECTURA EMPLEADA

La interconexión base que hemos implementado está compuesta por dos aplicaciones **Spring Boot** separadas, las cuales se comunican de forma **bidireccional** mediante una comunicación **server to server**:

1. **TPVV (Servidor)**
 - **URL Base:** `http://localhost:8123/tpvv/boardalo`
 - Mantiene una tabla de **Comercios** con sus respectivas **API Keys**.
 - Expone un **endpoint** para obtener el formulario de pago (`/pago/form`).
 - Expone un **endpoint** para procesar el pago (`/pago/realizar`).
 - Tras procesar un pago, **invoca** el endpoint de la Tienda para notificar la información del pedido completado.
2. **Tienda (Cliente)**
 - **URL Base:** `http://localhost:8246/`
 - Obtiene y muestra el formulario de pago que brinda el TPVV.
 - Envía las solicitudes de pago a **“/pago/realizar”** del TPVV.
 - Recibe la notificación final de pago (endpoint **/receivePedido**).
 - Almacena en su base de datos local el resultado del pedido.

3.5 DETALLES DE LA COMUNICACIÓN SERVER TO SERVER Y EL USO DE PROXIES

5.1 Visión Global del Flujo de Comunicación

1. **El usuario navega en la Tienda** y se muestra la información del pedido (mockeado)
2. **La Tienda** necesita mostrar un formulario de pago **“hosteado”** en el TPVV. Sin embargo, en lugar de redirigir directamente al TPVV, la Tienda actúa como un **proxy**⁶:
 - Llama al TPVV solicitando el HTML del formulario de pago a través del método **“pagoFormProxy”**.
 - Recibe ese HTML y lo **adapta** (inyecta campos ocultos, actualiza el <form>).
 - Entrega ese HTML final al navegador del usuario.

```
Optional<String> apiKeyOpt = parametroComercioService.getValorParametro( clave: "apiKey");
if (apiKeyOpt.isEmpty()) {
    model.addAttribute( attributeName: "error", attributeValue: "Error: API Key no encontrada.");
    return "error/404";
}
String apiKey = apiKeyOpt.get();

// Si hay errores en la query param, los mostramos
if (errors != null && !errors.isBlank()) {
    model.addAttribute( attributeName: "errorMessages", errors);
}

HttpHeaders headers = new HttpHeaders();
headers.set("Authorization", apiKey);

String url = "http://localhost:8123/tpvv/boardalo/pago/form?importe=" + precio
    + "&idTicket=" + ticket;

HttpEntity<String> entity = new HttpEntity<>(headers);

try {
    ResponseEntity<String> response =
        restTemplate.exchange(url, HttpMethod.GET, entity, String.class);
}
```

Figura 5: uso de **RestTemplate** para inyectar el HTML desde el servidor al cliente

3. **El usuario rellena los datos de la tarjeta** en la página servida por la Tienda y realiza el envío del formulario.

⁶ Un **proxy** es un intermediario que actúa entre un cliente y un servidor, gestionando las solicitudes y respuestas.

4. **De nuevo**, la petición de datos de la tarjeta (POST) se recibe primero en la Tienda, que vuelve a hacer una **llamada server to server** al TPVV para procesar el pago. Esto se realiza dentro del método “**realizarPagoProxy**”.

```
HttpEntity<PagoCompletoRequest> requestEntity = new HttpEntity<>(requestBody, headers);

String urlTPVV = "http://localhost:8123/tpvv/boardalo/pago/realizar";

try {

    ResponseEntity<String> response = restTemplate.postForEntity(urlTPVV, requestEntity, String.class);

    String body = response.getBody() != null ? response.getBody() : "";
```

Figura 6: la tienda usa **RestTemplate** para enviar al TPVV los datos del pago

5. **El TPVV** valida la **API Key**, registra el pago en su BD y, además, **notifica** a la Tienda que un pedido se ha procesado, con su estado final (rechazado, aceptado, pendiente). Todo esto se realiza dentro del método de **realizarPago**, dentro del fichero **PagoRestController.java** del proyecto del **TPVV**.

- Si el mensaje enviado desde el **TPVV** hasta la tienda empieza por “**OK**”, entonces significa que el pago se ha registrado como un **pago aceptado**.
- Si el mensaje empieza por “**PEND**”, entonces significa que el pago se ha registrado como un **pago pendiente**.
- Si el mensaje empieza por “**RECH**”, entonces significa que el pago se ha registrado como un **pago pendiente**.
- Si el mensaje empieza por “**RECH**”, entonces significa que el pago se ha registrado como un **pago rechazado**.
- Si el mensaje empieza por “**ERROR**”, entonces significa que ha sucedido algún error a la hora de realizar el pago.

```
// Llamada POST al proyecto cliente, enviando pedidoCompletoRequest
Mono<String> response = webClient.post() RequestBodyUriSpec
    .uri(urlBack) RequestBodySpec
    .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
    .body(BodyInserters.fromValue(pedidoCompletoRequest)) RequestHeadersSpec<capture of ?>
    .retrieve() ResponseSpec
    .bodyToMono(String.class);

String storeResponse = response.block();
```

Figura 7: El TPVV le envía a la tienda la información del pedido completado

6. En el caso del **TPVV**, para la comunicación **server to server** se ha utilizado **WebClient**, el cual a términos prácticos es equivalente a **RestTemplate**. Investigando, nos dimos que las dos formas principales de implementar comunicaciones **server to server** era usando tanto **WebClient** como **RestTemplate**, así que decidimos usar las dos, una para el **TPVV** y otra para la **tienda base**. Así porbábamos ambas alternativas.

```
© WebClientConfig.java x
1 package tpvv.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.reactive.function.client.WebClient;
6
7 @Configuration  Cristian Andrés Córdoba Silvestre
8 public class WebClientConfig {
9
10     @Bean  Cristian Andrés Córdoba Silvestre
11     > public WebClient webClient() { return WebClient.builder().build(); }
14 }
15
```

Figura 8: Configuración del **Bean** de **WebClient** dentro del proyecto del **TPVV**

```
        if (estadoPago.startsWith("RECH")) {  
            return ResponseEntity.ok( body: "RECH|" + razonEstadoPago + ".");  
        }  
  
        else if (estadoPago.startsWith("PEND")) {  
            return ResponseEntity.ok( body: "PEND|" + razonEstadoPago + ".");  
        }  
  
        else {  
            return ResponseEntity.ok( body: "OK|" + razonEstadoPago + ".");  
        }  
    } catch (IllegalArgumentException ex) {  
  
        return ResponseEntity.ok( body: "ERROR|Error 404");  
    }  
}
```

Figura 9: Manejo del estado del pago en la respuesta desde el TPVV a la tienda

7. Finalmente, la tienda recibe el mensaje correspondiente del **TPVV** y registra en su BD la información del nuevo pago y del nuevo pedido completado.

```
@PostMapping("/receivePedido")  Cristian Andrés Córdoba Silvestre  
public ResponseEntity<String> receivePedido(@RequestBody PedidoCompletoRequest request) {  
    log.debug("Recibido en la tienda un PedidoCompletoRequest: {}", request);  
  
    try {  
        pagoService.procesarPedido(request);  
        return ResponseEntity.ok( body: "Pedido recibido y guardado con éxito.");  
    } catch (IllegalArgumentException ex) {  
        return ResponseEntity.badRequest().body("Error 404");  
    }  
}
```

Figura 10: La tienda guarda la información del nuevo pedido completado en la BD

3.6 CLASES Y COMPONENTES RELEVANTES

3.6.1. En el TPVV

1. ApiKeyAuthFilter y SecurityConfig

- Se encargan de extraer la cabecera Authorization y de validar la **API Key**. Si la API Key no coincide con la de ningún comercio registrado en la BD del TPVV, entonces se devuelve un error **404**.
- Configuran que solo rutas /pago/** requieran la autenticación por medio de la **API Key** en la cabecera.

```
// Validar API-Key solo en rutas protegidas (ej: /pago/**)
if (path.startsWith("/pago/")) {
    String apiKey = extractApiKey(request);

    if (StringUtils.hasText(apiKey)) {
        Optional<Comercio> comercioOpt = comercioRepository.findByApiKey(apiKey);
        if (comercioOpt.isPresent()) {
            Comercio comercio = comercioOpt.get();
            UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(
                comercio, credentials: null, Collections.emptyList());
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authentication);
        } else {
            // API-Key no válida
            response.sendError(HttpServletResponse.SC_NOT_FOUND);
            return;
        }
    } else {
        // Falta la API-Key en la cabecera Authorization
        response.sendError(HttpServletResponse.SC_NOT_FOUND);
        return;
    }
}

// Continuar con el filtro
filterChain.doFilter(request, response);
}
```

Figura 11: fichero ApiKeyAuthFilter.java


```
// Permitir iframes en la misma ruta (necesario para la consola H2)
.headers( HeadersConfigurer<HttpSecurity> headers -> headers.frameOptions( FrameOptionsConfig frame -> frame.sameOrigin()))

// Control de sesiones (STATELESS)
.sessionManagement( SessionManagementConfigurer<HttpSecurity> session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

// Reglas de autorización
.authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
    .requestMatchers( @"/h2-console/**").permitAll() // Permitir H2
    .requestMatchers( @"/pago/**").authenticated() // Exigir autenticación en /pago
    .anyRequest().permitAll() // Todo lo demás, libre
)

// Filtro personalizado antes de UsernamePasswordAuthenticationFilter
.addFilterBefore(apiKeyAuthFilter, UsernamePasswordAuthenticationFilter.class)

// Autenticación básica
.httpBasic(Customizer.withDefaults());

// Construye el SecurityFilterChain
return http.build();
```

Figura 12: Fichero SecurityConfig.java

2. **PagoController** (Controlador web con Thymeleaf)
 - Método mostrarFormularioPago(...): Muestra la plantilla paymentForm.html.
3. **PagoRestController** (Controlador REST)
 - Método realizarPago(...): Procesa el JSON del pago, valida, guarda y notifica a la Tienda.
4. **PagoService**
 - Lógica de negocio principal para persistir el pago, determinar el estado y crear el DTO PedidoCompletoRequest que enviará de vuelta a la Tienda.
5. **IncidenciaController**(Detalles de una incidencia)

- Para indicar si se puede enviar un mensaje en la incidencia o no (solo usuario involucrados y que no hayan sido los últimos en responder), eran necesarias unas cuantas comprobaciones, las cuales las metimos en incidenciaService

```
@Transactional(readOnly = true)
public boolean canAddMessage(IncidenciaData incidencia, UsuarioData usuario) {
    Date fecha = new Date();
    boolean canAdd = false;
    if(!incidencia.getEstado().getNombre().equals(ESTADO_ASIGNADA) ||
        (!(Objects.equals(incidencia.getUsuarioTecnico().getId(), usuario.getId()) &&
            !(Objects.equals(incidencia.getUsuarioComercio().getId(), usuario.getId()))))) {
        return canAdd;
    }
    if(incidencia.getMensajes().isEmpty()){
        if(usuario.getTipoId() == USUARIO_TECNICO &&
            incidencia.getUsuarioTecnico() != null &&
            Objects.equals(usuario.getTipoId(), incidencia.getUsuarioTecnico().getTipoId())) {
            canAdd = true;
        }
    }
    }else{
        MensajeData mensaje = new ArrayList<>(incidencia.getMensajes()).get(incidencia.getMensajes().size()-1);
        if(!Objects.equals(mensaje.getUsuario().getId(), usuario.getId())){
            canAdd = true;
        }
    }
    return canAdd;
}
```

- También, para comprobar si se puede enviar una valoración (solo el usuario comercio implicado y con una incidencia en estado resuelta que no tenga una valoracion) era necesario una comprobación adicional. Quedándose el controlador así:

```
@GetMapping({"/comercio/incidencia/{id}", "/tecnico-or-admin/incidencia/{id}"})
public String detallesIncidencia(@PathVariable(value = "id") Long idIncidencia,
    Model model) {
    IncidenciaData incidencia = incidenciaService.obtenerIncidencia(idIncidencia);
    UsuarioData usuario = usuarioService.findById(getUsuarioLogeadoId());
    boolean canAddMessage = incidenciaService.canAddMessage(incidencia, usuario);
    boolean canValorar = usuario.getTipoId() == USUARIO_COMERCIO &&
        incidencia.getRazon_valoracion() == null &&
        incidencia.getUsuarioComercio() != null &&
        Objects.equals(incidencia.getEstado().getNombre(), ESTADO_RESUELTA) &&
        Objects.equals(incidencia.getUsuarioComercio().getId(), usuario.getId());
    model.addAttribute("incidencia", incidencia);
    model.addAttribute("usuario", usuario);
    model.addAttribute("canAddMessage", canAddMessage);
    model.addAttribute("canValorar", canValorar);
    return "detallesIncidencia";
}
```

3.6.2. En la Tienda

1. StoreController

- pagoFormProxy(...): Hace GET al TPVV para obtener el formulario, actúa como proxy inyectando datos.

- realizarPagoProxy(...): Hace POST al TPVV para realizar el pago y luego interpreta la respuesta (OK|, RECH|, PEND|).
- 2. **PagoService (en la Tienda)**
 - Método procesarPedido(...): Recibe el PedidoCompletoRequest proveniente del TPVV y lo almacena en la tabla pedido_completados.
- 3. **Entidades:** PedidoCompleto, ParametroComercio, etc.
 - Permiten almacenar la configuración (por ejemplo, la apiKey en la tabla parametros_comercio) y los registros de pedidos finalizados.
- 4. **RestTemplate** y su configuración en **AppConfig.java**.
 - Para orquestar las llamadas HTTP desde la Tienda al TPVV.

3.7 RESUMEN DE LOS BENEFICIOS Y MOTIVACIÓN DE LA COMUNICACIÓN SERVER TO SERVER Y DEL USO DE PROXIES

- **Control Centralizado del Flujo:** La Tienda, al hacer de proxy, conserva el control de la experiencia de usuario, sin exponer directamente la URL del TPVV.
- **Seguridad:** Se evita que el usuario interactúe directamente con la plataforma de pago, reduciendo el riesgo de exponer en el navegador la API Key o datos de implementación interna.
- **Mantenimiento:** Cambios futuros en la plataforma de pago pueden ser transparentes para el usuario, siempre y cuando la Tienda se encargue de “inyectar” la estructura de HTML que corresponda.

3.8 CONSIDERACIONES ADICIONALES

Para el formulario dentro de la pasarela de pago, se han realizado las siguientes validaciones:

- Ninguno de los campos puede estar vacío.
- El número de tarjeta ha de contener **exactamente** 16 dígitos. Los dígitos pueden escribirse o bien todos juntos o bien separados por un único espacio por cada grupo de cuatro dígitos (ejemplos de números de tarjeta válidos serían: “1234123412341234” y también “1234 1234 1234 1234”).
- La fecha de caducidad debe de seguir **exactamente** el formato “mm/aa”.
- El código de seguridad ha de contener **exactamente** 3 dígitos.

Cuando falla cualquiera de las validaciones mencionadas, se muestra por pantalla los mensajes de error correspondientes.

Errores en los datos introducidos: El nombre del titular de la tarjeta no puede estar vacío. El número de tarjeta no puede estar vacío. La fecha de caducidad no puede estar vacía. El código de seguridad no puede estar vacío.

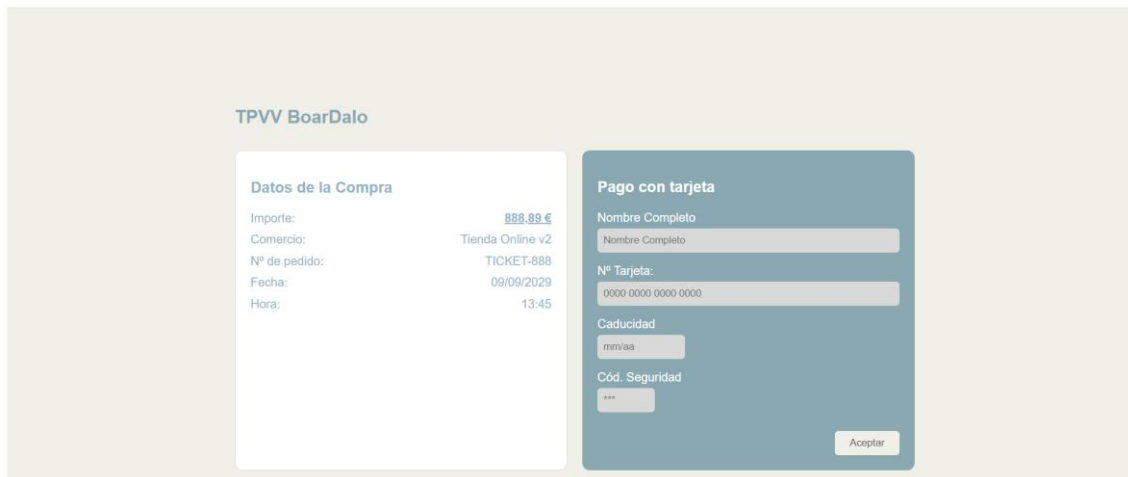


Figura 13: Error de validación. Todos los campos están vacíos.

Errores en los datos introducidos: El número de tarjeta debe tener exactamente 16 dígitos (los dígitos deben de introducirse todos seguidos o bien dejando un ÚNICO espacio en blanco entre cada grupo de cuatro dígitos). La fecha de caducidad debe tener el formato mm/yy. El CVC debe tener exactamente 3 dígitos.

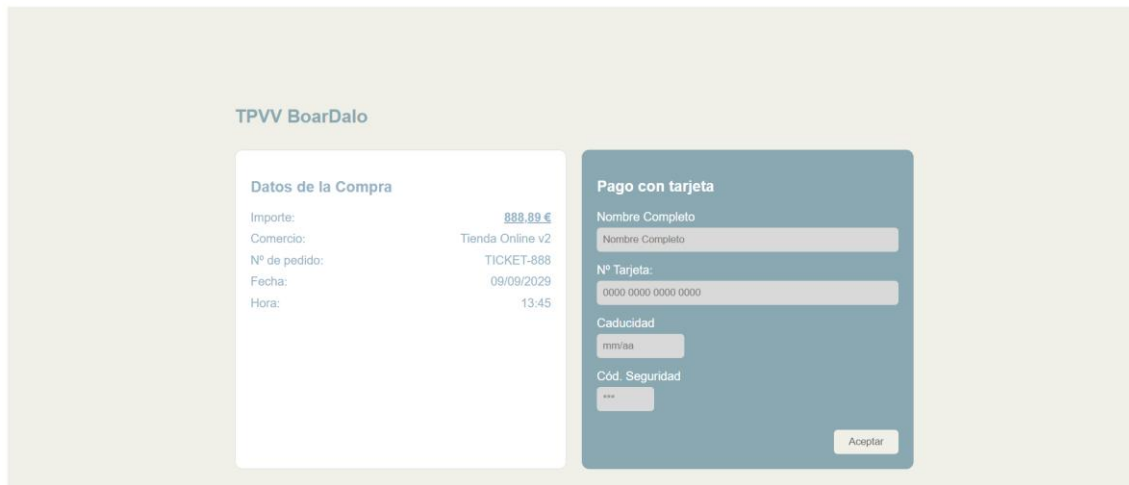


Figura 14: Error de validación. Se han introducido los datos con un formato incorrecto

Para realizar todas las validaciones se ha empleado regex⁷.

```
if (tarjetaData.getNumeroTarjeta() == null || tarjetaData.getNumeroTarjeta().isBlank()) {
    errorMsg.append("El número de tarjeta no puede estar vacío. ");
} else if (!tarjetaData.getNumeroTarjeta().matches("^\\d{16}|\\d{4} \\d{4} \\d{4} \\d{4}$")) {
    errorMsg.append("El número de tarjeta debe tener exactamente 16 dígitos (los dígitos deben de int
}

if (tarjetaData.getFechaCaducidad() == null || tarjetaData.getFechaCaducidad().isBlank()) {
    errorMsg.append("La fecha de caducidad no puede estar vacía. ");
} else if (!tarjetaData.getFechaCaducidad().matches("^([0-9]|1[0-2])\\d{2}$")) {
    errorMsg.append("La fecha de caducidad debe tener el formato mm/yy. ");
}

if (tarjetaData.getCvc() == null || tarjetaData.getCvc().isBlank()) {
    errorMsg.append("El código de seguridad no puede estar vacío. ");
} else if (!tarjetaData.getCvc().matches("^\\d{3}$")) {
    errorMsg.append("El CVC debe tener exactamente 3 dígitos. ");
}
```

Figura 15: Uso de regex en el proyecto del TPVV para realizar las validaciones en los datos del pago

⁷ **regex** (expresión regular) es un patrón de texto utilizado para buscar, coincidir o validar cadenas. Es ampliamente utilizado en validaciones para asegurarse de que los datos cumplan ciertos formatos, como correos electrónicos o contraseñas.

En base a los cuatro primeros dígitos del número de la tarjeta, se guarda un pago con uno u otro estado de pago de la siguiente manera:

0000 **** * → **RECH0001**

0001 **** * → **RECH0002**

0002 **** * → **RECH0003**

0003 **** * → **RECH0004**

1000 **** * → **PEND0001**

1001 **** * → **PEND0002**

1002 **** * → **PEND0003**

1003 **** * → **PEND0004**

2000 **** * → **ACEPT0001**

2001 **** * → **ACEPT0002**

2002 **** * → **ACEPT0003**

2003 **** * → **ACEPT0004**

Cualquier otro número de tarjeta → **ACEPT1000**

| WHERE | | | |
|-------|----|-----------|----------------------------------|
| | id | nombre | razon_estado |
| 1 | 1 | ACEPT1000 | PAGO ACEPTADO: PAGO PROCESADO... |
| 2 | 2 | ACEPT0001 | PAGO ACEPTADO: IDENTIDAD DEL ... |
| 3 | 3 | ACEPT0002 | PAGO ACEPTADO: REVISIÓN ANTIF... |
| 4 | 4 | ACEPT0003 | PAGO ACEPTADO: CONFIRMACIÓN I... |
| 5 | 5 | ACEPT0004 | PAGO ACEPTADO: MONEDA SOPORTA... |
| 6 | 6 | PEND0001 | PAGO PENDIENTE: VERIFICACIÓN ... |
| 7 | 7 | PEND0002 | PAGO PENDIENTE: TRANSFERENCIA... |
| 8 | 8 | PEND0003 | PAGO PENDIENTE: CONVERSIÓN DE... |
| 9 | 9 | PEND0004 | PAGO PENDIENTE: PROCESO DE CO... |
| 10 | 10 | RECH0001 | PAGO RECHAZADO: SALDO INSUFIC... |
| 11 | 11 | RECH0002 | PAGO RECHAZADO: TARJETA BLOQU... |
| 12 | 12 | RECH0003 | PAGO RECHAZADO: TARJETA VENCI... |
| 13 | 13 | RECH0004 | PAGO RECHAZADO: FALLO EN LA C... |

Figura 16: Tipos de Estado de Pago guardados en la BD

De esta forma , dependiendo del tipo de estado de pago, al recibir los datos del pedido completado, la tienda mostrará un tipo diferente de pantalla:

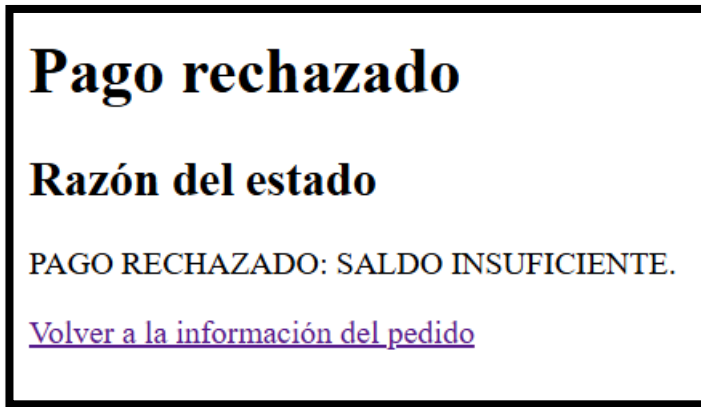


Figura 17: Pantalla mostrada en la tienda tras haberse efectuado un pago rechazado



Figura 18: Pantalla mostrada en la tienda tras haberse efectuado un pago pendiente

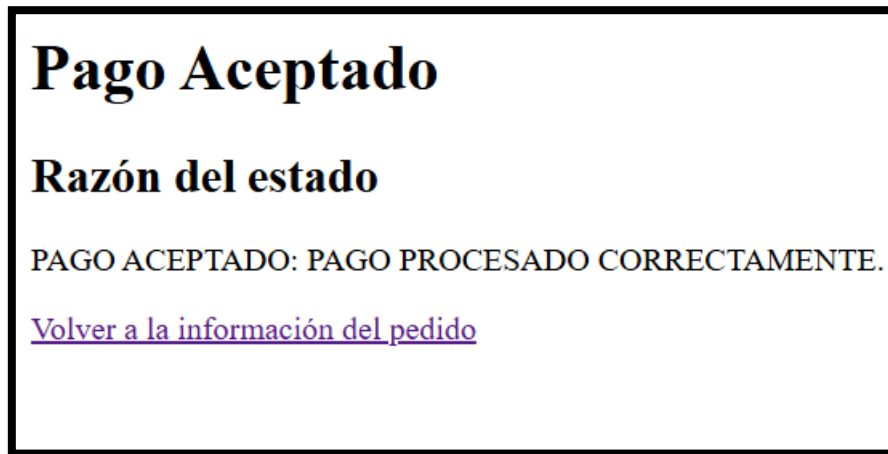


Figura 19: Pantalla mostrada en la tienda tras haberse efectuado un pago rechazado

3.9. CONCLUSIONES

La integración entre la **Tienda** y el **TPVV** demuestra una forma robusta de implementar un **flujo de pago**. La **API Key** provee un mecanismo simple de autenticación para cada comercio, y la **comunicación server to server** garantiza que los datos de pago se procesen y almacenen en la plataforma de forma segura, a la vez que la **Tienda** conserva la información final de los pedidos, proporcionando así una solución modular, escalable y fácil de mantener.

En este planteamiento, la Tienda hace un uso intensivo de **llamadas proxy**:

1. **Para el formulario** (GET al TPVV)
2. **Para el procesamiento** (POST al TPVV)

Por su parte, el TPVV complementa esta interacción **notificando** a la Tienda los resultados. Esta arquitectura ofrece varias ventajas en términos de **control**, **seguridad** y **mantenibilidad** de la solución. Aún así, existen diferentes aspectos que podrían mejorarse en implementaciones futuras.

Puntos de Mejora:

- **Almacenamiento seguro** de la **API Key** en la Tienda, evitando exponerla en texto plano.
- Reducir la manipulación de HTML por “reemplazos de cadena” e implementar soluciones más escalables (p. ej., exponer un JSON con la información del formulario y que la Tienda renderice su propio template).
- Incluir un sistema de **firmas HMAC** o **JWT** para robustecer la seguridad de la comunicación más allá de la simple verificación de API Key.

No obstante, el aspecto más importante a mejorar a tener en cuenta para implementaciones futuras sería el aseguramiento de que los datos del pago **no cambien** durante el propio proceso de realización del pago.

Básicamente con esto se pretende evitar que, por ejemplo, un atacante llegue a la pantalla de pasarela de pago, cambie el importe, (por ejemplo a través del DOM), y se termine efectuando un pago con un importe mucho menor al que tenía el pago real

Evidentemente, esto representa una acción realmente peligrosa, ya que cualquiera podría realizar cambios en los datos del pago desde la pantalla de la pasarela de pago (simplemente bastaría con acceder al **DOM** y cambiar desde allí cualquier elemento del HTML y después darle a realizar pago).

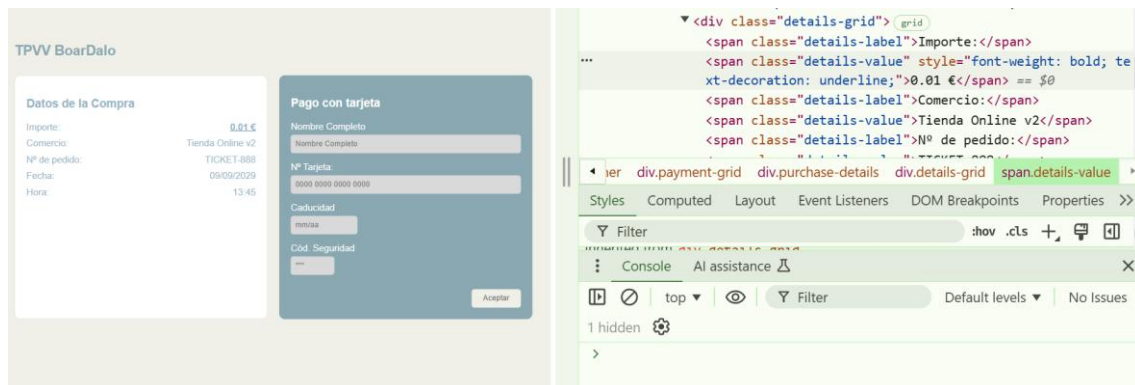


Figura 20: Cambiando los elementos del pago directamente desde el DOM

Para solucionar este problema, llegados a dos posibles opciones:

1. Implementar un sistema de **firmas**, de forma que cuando la tienda acceda a la pasarela de pago, el mismo **backend** de la tienda se encargue de cifrar mediante una clave pública los datos iniciales del pago (**importe, ticket**). Estos datos cifrados serían enviados al TPVV y almacenados en su BD. Cuando la tienda completase el pago, estos datos iniciales volverían a enviarse cifrados con la clave pública junto con el resto de información del pago. Cuando el **TPVV** reciba estos datos, si los datos cifrados iniciales no coinciden con los datos cifrados iniciales, entonces esto es un indicio de que ha sucedido alguna acción sospechosa, con lo que no se completaría el pago y se lanzaría un mensaje de error **404**.
2. Cuando la tienda acceda a la pasarela de pago, los datos iniciales del pago (**importe, ticket**) son enviados tal cual al **TPVV**. El **TPVV** recibe estos datos y los almacena en una tabla temporal. Una vez que la tienda haya completado el pago, todos los datos del pago son enviados al **TPVV**. Cuando el **TPVV** recibe estos datos, los compara con los datos iniciales almacenados en la tabla temporal. Si no coinciden, entonces el pago no se completa y se lanza un mensaje de error **404**.

Debido al alcance del proyecto y a las limitaciones temporales del mismo, al final no terminamos estas consideraciones. No obstante, es fundamental que este requisito fuese implementado en actualizaciones futuras de la aplicación.

4. Metodologías y planificación

Desarrollo Guiado por Pruebas (TDD):

Durante el desarrollo del proyecto, hemos adoptado el método de **Test Driven Development (TDD)** para implementar los sistemas de CRUD relacionados con el comercio y los usuarios. Este enfoque se basa en escribir primero las pruebas unitarias que definen el comportamiento esperado de las funcionalidades, para luego implementar el código necesario que permita superar dichas pruebas.

Al desarrollar de esta manera, hemos logrado varios beneficios importantes:

1. **Reducción de errores:** Al asegurarnos de que cada funcionalidad está respaldada por pruebas, disminuimos significativamente el riesgo de introducir errores o bugs en el código.
2. **Código más limpio y mantenible:** TDD fomenta la escritura de código más estructurado y modular, facilitando su comprensión y modificación futura.
3. **Desarrollo guiado por objetivos:** Este método nos obliga a enfocarnos en los requisitos y casos de uso específicos antes de escribir cualquier línea de código, lo que asegura que cada funcionalidad implementada cumple con las expectativas desde el principio.
4. **Detección temprana de problemas:** Las pruebas automatizadas permiten identificar problemas durante el desarrollo, en lugar de encontrarlos en etapas posteriores, donde pueden ser más costosos de corregir.

Gracias a la implementación de TDD, hemos construido un sistema robusto y confiable, lo que garantiza que las funcionalidades críticas, como la gestión de usuarios y transacciones comerciales, cumplen con los estándares de calidad desde el primer momento.

Programación en Pareja (Pair Programming)

Durante el desarrollo, hemos utilizado el método de **pair programming** para implementar la funcionalidad del borrado lógico de usuarios. Este enfoque consiste en trabajar por parejas, donde una persona actúa como la "mano ejecutora" (es decir, escribe el código) y la otra desempeña el papel de "supervisor" o "navegador", encargándose de revisar el trabajo, detectar posibles errores y proponer mejoras en tiempo real.

Este método ha demostrado ser eficaz en varias etapas del desarrollo:

1. **Implementación de nuevas funcionalidades:** La colaboración entre dos desarrolladores permite abordar problemas complejos desde diferentes perspectivas, logrando soluciones más creativas y efectivas.
2. **Detección y corrección de errores:** El rol del supervisor facilita la identificación de fallos en el código o inconsistencias lógicas de manera temprana, reduciendo el tiempo necesario para resolver errores posteriormente.
3. **Transferencia de conocimientos:** Pair programming fomenta el aprendizaje mutuo entre los integrantes del equipo, lo que mejora las habilidades individuales y fortalece el conocimiento colectivo del proyecto.
4. **Mejora de la calidad del código:** Este enfoque asegura que el código sea revisado continuamente, lo que contribuye a mantener altos estándares de calidad y buenas prácticas de programación.

Gracias a esta metodología, hemos optimizado tanto el proceso de desarrollo como la resolución de problemas, fortaleciendo la colaboración en el equipo y garantizando un código más robusto y confiable.

Kanban:

Hemos utilizado el método Kanban para la gestión y distribución de tareas y funcionalidades durante el desarrollo del proyecto. Cada historia de usuario, representada mediante tarjetas en un tablero de Trello, refleja una funcionalidad específica que debe ser desarrollada.

Nuestro tablero está organizado en cuatro columnas principales:

1. **Backlog:** En esta columna se encuentran todas las tareas pendientes que aún no han sido priorizadas ni seleccionadas para su desarrollo.
2. **To Do:** Aquí se agrupan las tareas que han sido priorizadas y seleccionadas para ser desarrolladas próximamente. Estas tareas están listas para iniciar su ciclo de desarrollo.
3. **In Progress:** Esta columna representa las tareas que se encuentran en proceso de desarrollo. Indica que los equipos están trabajando activamente en ellas.
4. **Done:** En esta columna se colocan las tareas que ya han sido completadas y cumplen con los criterios de aceptación previamente definidos.

El enfoque Kanban nos ha permitido visualizar el flujo de trabajo de manera clara, identificar posibles cuellos de botella y garantizar una gestión eficiente del tiempo y los recursos. Además, gracias a la flexibilidad del tablero, hemos podido adaptar el proceso a las necesidades específicas del equipo y del proyecto.

<https://trello.com/invite/b/67588a8847aeacbda9d19cf3/ATTI1e94424a56e02cc6288328835de345f5D5FD854B/tpvv>

GitFlow:

Hemos utilizado GitFlow como modelo de ramificación para gestionar de manera eficiente el control de versiones durante el desarrollo del proyecto. Este flujo de trabajo nos ha permitido organizar las distintas etapas del desarrollo mediante un esquema claro de ramas, asegurando estabilidad en el código y facilitando la colaboración en equipo.

El modelo se estructura de la siguiente manera:

1. Rama principal (main): Es la rama principal que contiene el código en su estado más estable, preparado para ser desplegado en producción.
2. Rama de desarrollo (develop): Es la rama principal para el desarrollo continuo. Aquí se integran las nuevas funcionalidades antes de ser preparadas para el lanzamiento.
3. Ramas de funcionalidad: Cada nueva funcionalidad o mejora del proyecto se desarrolla en ramas temporales creadas a partir de develop. Una vez completadas y revisadas, se fusionan de nuevo en la rama develop.

El uso de GitFlow nos ha proporcionado las siguientes ventajas:

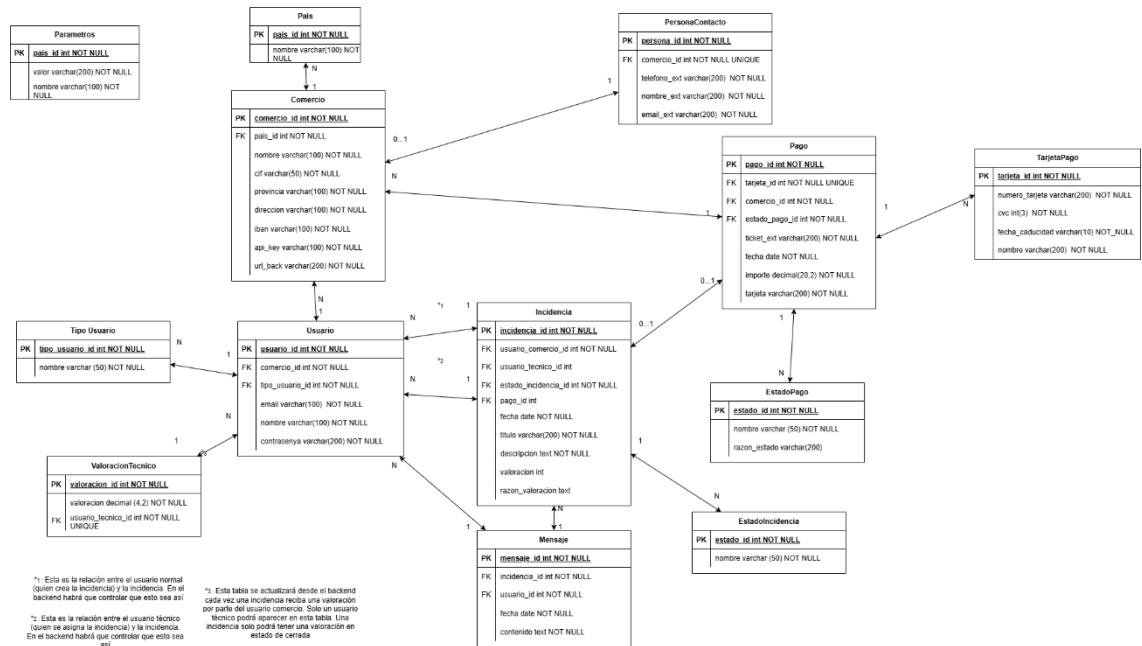
- Organización y claridad: Al separar las ramas según su propósito, hemos podido mantener un flujo de trabajo estructurado y comprensible para todos los miembros del equipo.
- Estabilidad: Gracias a la rama main como fuente única de código listo para producción, hemos reducido riesgos al realizar despliegues.
- Colaboración eficaz: Las ramas de funcionalidad han permitido que diferentes miembros trabajen en paralelo sin interferir en las tareas de los demás.

Este enfoque ha sido clave para gestionar un desarrollo ágil y controlado, asegurando tanto la calidad del producto final como la eficiencia del equipo.

Para el desarrollo

5. Diagrama EER

Este es nuestro diagrama de base de datos que seguimos para la realización del esquema



6. Diseño de la aplicación

6.1 Mockups

Estos son todos los mockups que realizamos antes de la fase de desarrollo. No los seguimos al 100% ya que hicimos algunas modificaciones, pero estos son:

Parte sin usuario registrado:

TPVV

Datos de la Compra

Importe: **218,42 €**

Comercio: Paquetería

Nº de pedido: 0000000001

Fecha: 09/12/2024

Hora: 10:46

Pago con tarjeta

Nº Tarjeta: 0000 0000 0000 0000

Caducidad: mm/aa

Cód. Seguridad: ***

Aceptar

Ilustración 1 Pasarela de pago

Iniciar sesión

Correo: info@tpvv.com

Contraseña: *****

Aceptar

Ilustración 2 Inicio de sesión

Parte de usuario comercio:

Incidencias

Pagos

Nombre

Mis incidencias

Id

Estado

dd/mm/AAAA - dd/mm/AAAA

| # | Fecha | Técnico | Título | Estados |
|----------------|---------------------|----------------|---------------------------|-----------|
| 00000 00001 | 10/07/2024 16:44 | - | No he podido completar... | En espera |
| 00000 00002 | 10/07/2024 16:44 | Paco Fernandez | No he podido completar... | Cerrada |
| 00000 00003 | 10/07/2024 16:44 | María Perez | No he podido completar... | Pendiente |
| 00000 00004 | 10/07/2024 16:44 | Paco Fernandez | No he podido completar... | En espera |
| 00000 00005 | 10/07/2024 16:44 | María Perez | No he podido completar... | Cerrada |
| 00000 00006 | 10/07/2024 16:44 | María Perez | No he podido completar... | Cerrada |

< 1 2 >

Ilustración 3 Incidencias del comercio

Incidenias

Pagos

Nombre

Pagos del comercio

Id

Id ticket

Estados

dd/mm/AAAA - dd/mm/AAAA

| # | Fecha | Id Ticket | Importe | Estado |
|------------|------------------|-----------------|-----------|-----------|
| 0000000002 | 10/07/2024 16:44 | 0000000002aj-23 | 314,99€ | Aceptado |
| 0000000003 | 10/07/2024 16:44 | 0000000002aj-23 | 34,99€ | Pendiente |
| 0000000004 | 10/07/2024 16:44 | 0000000002aj-23 | 12,99€ | Aceptado |
| 0000000005 | 10/07/2024 16:44 | 0000000002aj-23 | 1.123,99€ | Rechazado |
| 0000000006 | 10/07/2024 16:44 | 0000000002aj-23 | 314,99€ | Pendiente |
| 0000000007 | 10/07/2024 16:44 | 0000000002aj-23 | 9,99€ | Aceptado |

<

1

2

>

Ilustración 4 Pagos del comercio

Incidentes

Pagos

Nombre 

Detalle pago nº 0000000002

Detalles del pago

Fecha: 10/07/2024 16:44

Id Ticket Ext. : 000000002aj-23

Importe: 34,99€

Estado: Pendiente-PendBanc

Razón estado: El pago aún no ha llegado a la entidad bancaria

Importe: 34,99€

Pagador

Numero tarjeta: **** * 1111

Nombre tarjeta: Juan Jimenez Gonzalez

Incidencia


Este pago no tiene una incidencia asociada

Crear

Ilustración 5 Detalles del pago


Incidentes

Pagos

Nombre 

Nombre de usuario

Email: usuario@paqueteria.es

Comercio: Paqueteria 

Api Key comercio:
1234567890abcdef1234567890ab
cdef

Resetear api key

Ilustración 6 Perfil del usuario comercio



Incidencias Pagos Nombre ▾

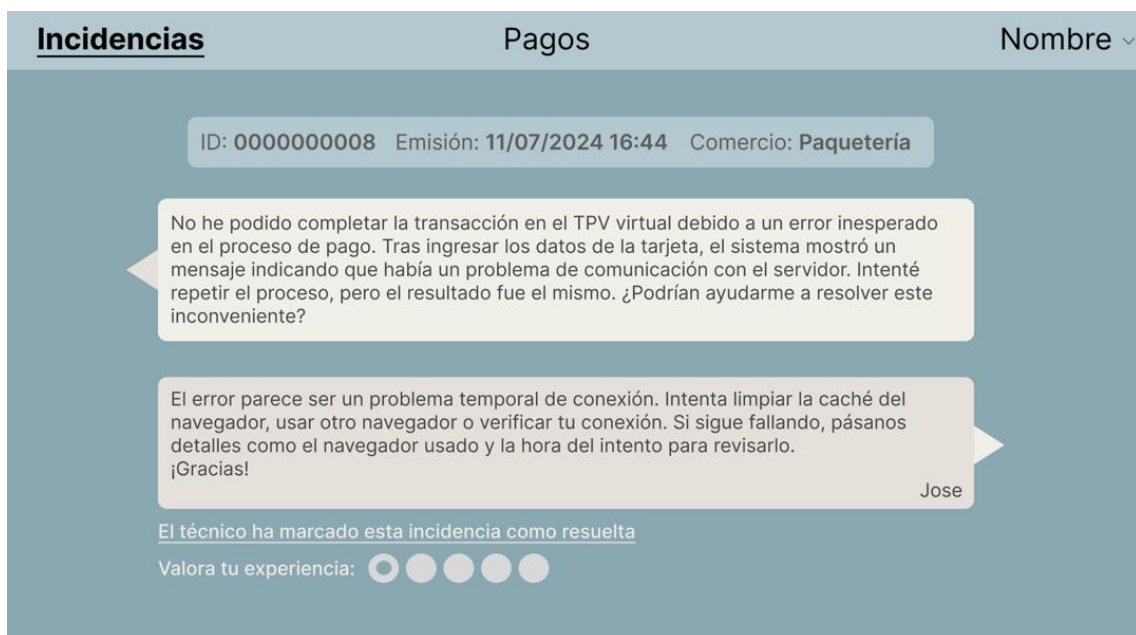
Título

ID pago: 0000000002

Descripción

Enviar

Ilustración 7 Crear incidencia



Incidencias Pagos Nombre ▾

ID: 0000000008 Emisión: 11/07/2024 16:44 Comercio: Paquetería

No he podido completar la transacción en el TPV virtual debido a un error inesperado en el proceso de pago. Tras ingresar los datos de la tarjeta, el sistema mostró un mensaje indicando que había un problema de comunicación con el servidor. Intenté repetir el proceso, pero el resultado fue el mismo. ¿Podrían ayudarme a resolver este inconveniente?

El error parece ser un problema temporal de conexión. Intenta limpiar la caché del navegador, usar otro navegador o verificar tu conexión. Si sigue fallando, pásanos detalles como el navegador usado y la hora del intento para revisarlo. ¡Gracias!

Jose

El técnico ha marcado esta incidencia como resuelta

Valora tu experiencia: ○ ○ ○ ○ ○

Ilustración 8 Detalles incidencia

Parte de usuario técnico:

Comercios

Inc

idencias

Nombre

Inc

idencias sin asignar

Id

Comercios

dd/mm/AAAA - dd/mm/AAAA

| # | Fecha | Comercio | Título | Asignarme |
|----------------|---------------------|----------------|---------------------------|-----------|
| 00000 00008 | 11/07/2024 16:44 | Paquetería | No he podido completar... | + |
| 00000 00009 | 11/07/2024 16:44 | Paquetería | No he podido completar... | + |
| 00000 00010 | 12/07/2024 16:44 | Gimnasio 1 | No he podido completar... | + |
| 00000 00011 | 13/07/2024 16:44 | Gimnasio 1 | No he podido completar... | + |
| 00000 00012 | 14/07/2024 16:44 | Hotel numero 2 | No he podido completar... | + |
| 00000 00013 | 17/07/2024 16:44 | Gimnasio 2 | No he podido completar... | + |

< 1 2 >

Mis inc

idencias

Id

Comercios

dd/mm/AAAA - dd/mm/AAAA

| # | Fecha | Comercio | Título | Estados |
|----------------|---------------------|----------------|---------------------------|-----------|
| 00000 00001 | 10/07/2024 16:44 | Hotel numero 2 | No he podido completar... | En espera |
| 00000 00002 | 10/07/2024 16:44 | Hotel numero 2 | No he podido completar... | Cerrada |
| 00000 00003 | 10/07/2024 16:44 | Hotel numero 2 | No he podido completar... | Pendiente |
| 00000 00004 | 10/07/2024 16:44 | Hotel numero 2 | No he podido completar... | En espera |
| 00000 00005 | 10/07/2024 16:44 | Hotel numero 2 | No he podido completar... | Cerrada |
| 00000 00006 | 10/07/2024 16:44 | Hotel numero 2 | No he podido completar... | Cerrada |

< 1 2 >

Ilustración 10 Incidencias técnico

Comercios

Inc

idencias

Nombre

Comercios

Id

Nombre

Cif

Pais

dd/mm/AAAA - dd/mm/AAAA

| # | Fecha Alta | Nombre | Cif | Pais | Provincia |
|----------------|---------------------|------------------|-----------|--------|-----------|
| 0000000 002 | 10/07/2024 16:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante |
| 0000000 002 | 10/07/2024 16:44 | Paquetería | 13243245R | España | Alicante |
| 0000000 002 | 10/07/2024 16:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante |
| 0000000 002 | 10/07/2024 16:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante |
| 0000000 002 | 10/07/2024 16:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante |
| 0000000 002 | 10/07/2024 16:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante |

< 1 2 >

Ilustración 9 Comercios técnico

Comercios

Incidencias

Nombre 

Paquetería

Fecha de alta: 10/07/2024 16:44

Pais: España

Provincia: Alicante

Dirección: Avenida pintor baeza 8

Cif: 43221234R

Iban: ES7921000813610123 456789

Url: https://www.paqueteria.es/endpoint

Persona de contacto

Nombre: 10/07/2024 16:44

Telefono: 000000002aj-23

Email: 34,99€

API KEY: 1234567890abcdef12345 67890abcdef

Resetear api key

Usuarios


usuario1@paqueteria.es 

usuario2@paqueteria.es 

Ilustración 12 Detalles de un comercio

Comercios

Incidencias

Nombre 

Nombre de usuario

Email: tecnico@tecnico.es


Media de valoraciones: 

Ilustración 11 Usuario técnico

Parte de usuario administrador:

Comercios

Usuarios

Incidencias

Pagos

Nombre

Incidencias

Id

Comercio

Estado

dd/mm/AAAA - dd/mm/AAAA

| # | Fecha | Comercio | Técnico | Título | Estados |
|----------------|---------------------|-------------------|-------------------|------------------------------|-----------|
| 00000 00001 | 10/07/2024 16:44 | Hotel numero 2 | - | No he podido completar... | En espera |
| 00000 00002 | 10/07/2024 16:44 | Hotel numero 2 | Paco Fernandez | No he podido completar... | Cerrada |
| 00000 00003 | 10/07/2024 16:44 | Hotel numero 2 | María Perez | No he podido completar... | Pendiente |
| 00000 00004 | 10/07/2024 16:44 | Hotel numero 2 | Paco Fernandez | No he podido completar... | En espera |
| 00000 00005 | 10/07/2024 16:44 | Hotel numero 2 | María Perez | No he podido completar... | Cerrada |
| 00000 00006 | 10/07/2024 16:44 | Hotel numero 2 | María Perez | No he podido completar... | Cerrada |

<

1

2

>

Ilustración 13 Incidencias administrador

Comercios

Usuarios

Incidencias

Pagos

Nombre ▾

Comercios

Id

Nombre

Cif

Pais

dd/mm/AAAA - dd/mm/AAAA

| # ▴▾ | Fecha Alta ▴▾ | Nombre | Cif | Pais | Provincia | Estado |
|------------|-----------------|------------------|-----------|--------|-----------|-----------|
| 0000000002 | 10/07/202416:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante | Activo |
| 0000000002 | 10/07/202416:44 | Paquetería | 13243245R | España | Alicante | Desactivo |
| 0000000002 | 10/07/202416:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante | Desactivo |
| 0000000002 | 10/07/202416:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante | Desactivo |
| 0000000002 | 10/07/202416:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante | Desactivo |
| 0000000002 | 10/07/202416:44 | Hotel 2 grupo 15 | 13243245R | España | Alicante | Desactivo |

<

1

2

>

Ilustración 14 Comercios en administrador

Comercios

Usuarios

Incidencias

Pagos

Nombre ▾

Usuarios

Id 🔍

Comercio ▾

Estado ▾

dd/mm/AAAA - dd/mm/AAAA 📅

| # ⬆️ | Fecha ⬆️ | Tipo | Comercio | Nombre | Email |
|----------------|---------------------|----------|----------------|----------------|---------------------|
| 00000 00001 | 10/07/2024 16:44 | Comercio | Hotel numero 2 | Paco Fernandez | comercio@hotel2.com |
| 00000 00001 | 10/07/2024 16:44 | Admin | - | Paco Fernandez | comercio@hotel2.com |
| 00000 00001 | 10/07/2024 16:44 | Técnico | - | Paco Fernandez | comercio@hotel2.com |
| 00000 00001 | 10/07/2024 16:44 | Comercio | Hotel numero 2 | Paco Fernandez | comercio@hotel2.com |
| 00000 00001 | 10/07/2024 16:44 | Comercio | Hotel numero 2 | Paco Fernandez | comercio@hotel2.com |
| 00000 00001 | 10/07/2024 16:44 | Comercio | Hotel numero 2 | Paco Fernandez | comercio@hotel2.com |

< 1 2 >

Ilustración 16 Usuario en administrador

Incidencias

Pagos

Nombre

Pagos del comercio

Id

Id ticket

Estados

dd/mm/AAAA - dd/mm/AAAA

| # | Fecha | Id Ticket | Importe | Estado | Comercio |
|------------|------------------|------------------|-----------|-----------|------------------|
| 0000000002 | 10/07/2024 16:44 | 00000000002aj-23 | 314,99€ | Aceptado | Hotel 2 grupo 15 |
| 0000000003 | 10/07/2024 16:44 | 00000000002aj-23 | 34,99€ | Pendiente | Hotel 2 grupo 11 |
| 0000000004 | 10/07/2024 16:44 | 00000000002aj-23 | 12,99€ | Aceptado | Hotel 2 grupo 15 |
| 0000000005 | 10/07/2024 16:44 | 00000000002aj-23 | 1.123,99€ | Rechazado | Hotel 2 grupo 15 |
| 0000000006 | 10/07/2024 16:44 | 00000000002aj-23 | 314,99€ | Pendiente | Hotel 2 grupo 15 |
| 0000000007 | 10/07/2024 16:44 | 00000000002aj-23 | 9,99€ | Aceptado | Hotel 2 grupo 15 |

<

1

2

>

Ilustración 15 Pagos usuario admin. (Era muy parecido al de comercio y copiamos)

Comercios **Usuarios** Incidencias Pagos Nombre ▾

Crear usuario

Datos generales

Tipo usuario*: Usuario comercio ▾

Comercio*: Paquetería - 111111111R ▾

Nombre*: España

Email*: user@gmail.com

Contraseña*: *

Repetir contraseña*: *

Crear

Ilustración 18 Crear usuario

Comercios Usuarios Incidencias Pagos Nombre ▾

Crear comercio

| <u>Datos generales</u> | <u>Persona de contacto</u> |
|---|---|
| Nombre*: Paquetería | Nombre: Paquetería |
| Cif*: 11111111R | Email: usuarioext@gmail.com |
| País*: España ▾ | Teléfono: +3466666666 |
| Provincia*: Alicante | <small>Una persona de contacto es opcional, pero si se pone 1 dato de la persona de contacto, será necesario completarlos todos</small> |
| Dirección*: Alicante | |
| Iban*: ES7921000813610123456789 | |
| Url*: https://www.paqueteria.es/endpoint | |

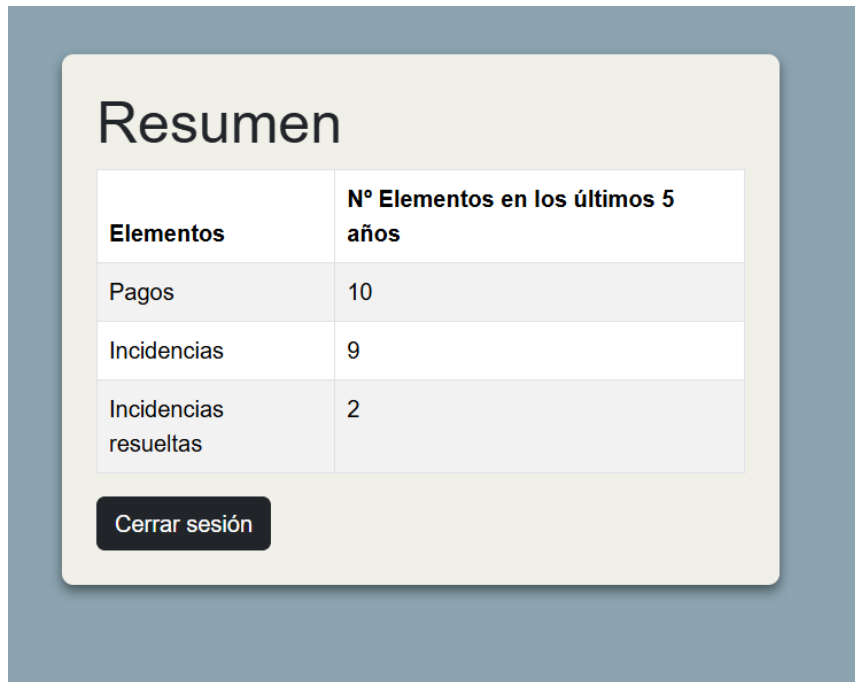
Crear

Ilustración 17 Crear comercio

6.2 Patrones de diseño:

Los patrones de diseño que hemos utilizado para los distintos apartados son:

- **Para la página principal:** Hemos creado una página principal donde se ve un sencillo dashboard que tiene un pequeño resumen con datos de los últimos elementos. El usuario también podrá cerrar la sesión desde aquí.



- **Para la navegación:** Tenemos una barra de navegación que indica las páginas de listados que tiene el usuario disponibles. Cada rol en nuestra aplicación tendrá un número de ítems distintos en la barra de navegación

TPVV-BoarDalo Comercios Usuarios Incidencias Pagos Nombre

- **Para los formularios:** Para los formularios tenemos una estructura simple de label-input donde el label tendrá un asterisco si el campo es obligatorio. El botón de submit del formulario tendrá un verbo en infinitivo que describirá la acción que hará ese botón



Crear Usuario

Datos generales

*Tipo usuario
Admin

*Comercio
Gimnasio ASN San Vicente

*Nombre
[Input]

*Email
[Input]

*Contraseña
[Input]

Registrar

- **Para la identificación del usuario:** En el inicio de sesión, tenemos un simple formulario donde se pide el correo y la contraseña del usuario. Dentro de los input, está de placeholder información que ayudará al usuario a rellenar este formulario



Iniciar sesión

Correo
info@tpvv.com

Contraseña

Aceptar

- **Para los listados:** Para todos los listados tenemos una columna donde mostramos el id del ítem y otra para la fecha, después un poco de información y por último una columna de acciones donde se pondrán distintos botones de acciones

| # | Fecha Alta | Nombre | Cif | País | Provincia | Estado | Acciones | |
|---|-----------------------|----------------------------------|---------------|---------|---------------|--------|--------------------------|----------------------------|
| 1 | 2022-10-27 00:00:00.0 | Gimnasio ASN San Vicente | CIF123456 | España | Alicante | Activo | Detalles | Desactivar |
| 2 | 2023-07-21 00:00:00.0 | Tienda de Juegos de Mesa Chessyx | CIF123457 | España | Madrid | Activo | Detalles | Desactivar |
| 4 | 2024-12-31 00:00:00.0 | Consultoría Global S.A. | B98765432 | Suiza | Ticino | Activo | Detalles | Desactivar |
| 5 | 2024-12-31 00:00:00.0 | Solutions Innovantes S.A.S. | FR12345678901 | Francia | Île-de-France | Activo | Detalles | Desactivar |

7. Problemas encontrados y su solución

Problema de paginación:

En las vistas del listado de comercios y usuarios, al cambiar de página, los filtros aplicados se perdían, lo que provocaba que los datos mostrados no respetaran las condiciones del filtrado establecido por el usuario.

Causa:

El problema ocurría porque los parámetros de los filtros no se pasaban de forma persistente en los enlaces de paginación. Como resultado, cada vez que se cambiaba de página, la solicitud ignoraba los filtros activos y devolvía resultados sin filtrar.

Solución:

Se actualizó la lógica de la vista para que los enlaces de paginación incluyeran los parámetros de filtrado (como `idFilter`, `nombreFilter`, `fechaDesdeStr`, etc.) junto con los parámetros de paginación (`page` y `totalPages`). Esto se implementó añadiendo los filtros como parte de la URL generada dinámicamente en los enlaces de las páginas anterior y siguiente.

Problema en detalles de comercio:

En la vista del listado de comercios, al intentar acceder a los detalles de un comercio que no tiene asignada una persona de contacto, se producía un error.

Causa:

El problema ocurría porque al intentar acceder a los atributos de un objeto nulo (objeto de la clase PersonaContactoData), se lanzaba una excepción.

Solución:

Antes de mostrar la información de la persona de contacto, se agregó una comprobación para verificar si el objeto existe, es decir, que no sea nulo.

Problema en detalle de incidencia:

En la vista de detalle de incidencia, las valoraciones dejaban entrar con un valor nulo en la razón de valoración, aun teniendo el required en el textarea de esta

Causa:

El problema era porque en el html, entre la etiqueta de abrir el textarea y la de cerrar, había un espacio en blanco, que contaba como un string con un carácter.

Solución:

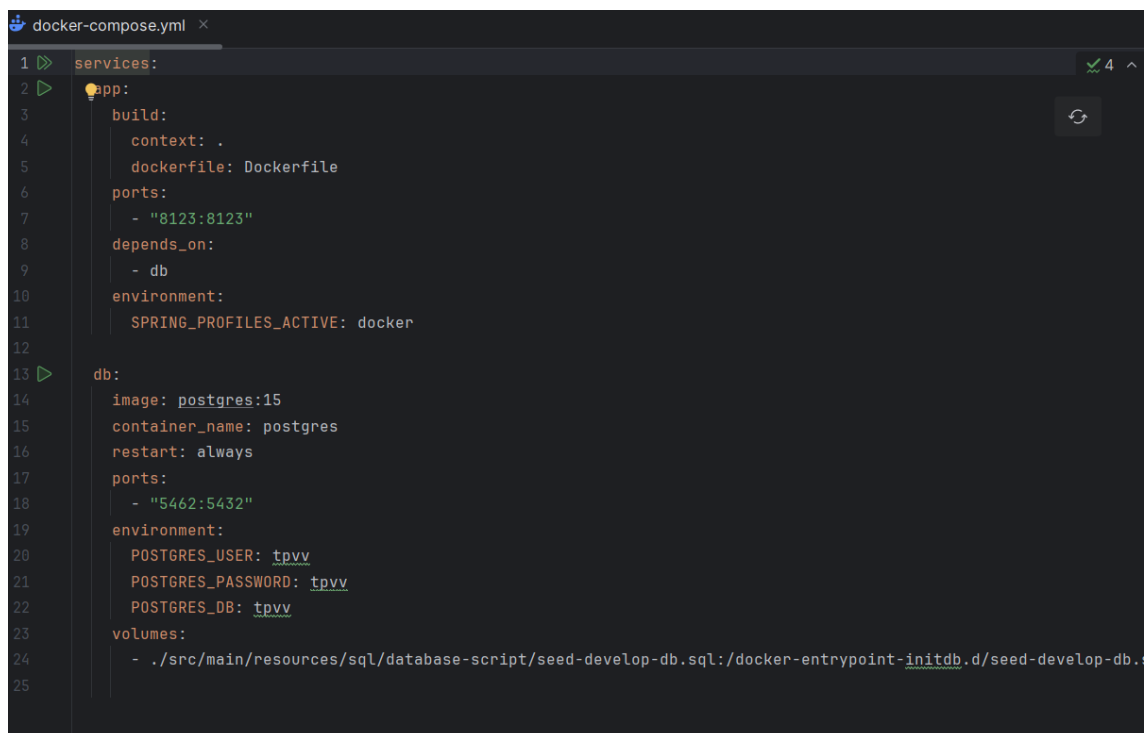
Primero, borrar el espacio en blanco con el textarea, y segundo, añadir un pequeño javascript que comprobase si el valor de ese input era una string vacía o con solo caracteres vacíos.

8. Aspectos extras de implementación

A continuación se mencionan aspectos que han servido como un **plus** de nuestro proyecto y que realzan la robustez y precisión de nuestro proyecto:

8.1 Docker compose:

Se ha implementado un sistema de **docker compose** mediante el cual tenemos nuestra aplicación dockerizada en un contenedor y la base de datos con una imagen de **PostgreSQL** en otro contenedor. Para ello, se ha creado un fichero **docker-compose.yml** mediante el cual, al ejecutarlo mediante el comando **docker compose up --build**, se generan las imágenes de nuestra aplicación y de nuestra base de datos, de forma que nuestra aplicación queda ejecutada correctamente y de forma automática. Gracias a este aspecto hemos dotado a nuestra aplicación de portabilidad, permitiendo poder ejecutar la aplicación en diferentes entornos sin importar los programas instalados en el mismo.



```
1 services:
2   app:
3     build:
4       context: .
5       dockerfile: Dockerfile
6     ports:
7       - "8123:8123"
8     depends_on:
9       - db
10    environment:
11      SPRING_PROFILES_ACTIVE: docker
12
13  db:
14    image: postgres:15
15    container_name: postgres
16    restart: always
17    ports:
18      - "5462:5432"
19    environment:
20      POSTGRES_USER: tpvv
21      POSTGRES_PASSWORD: tpvv
22      POSTGRES_DB: tpvv
23    volumes:
24      - ./src/main/resources/sql/database-script/seed-develop-db.sql:/docker-entrypoint-initdb.d/seed-develop-db.sql
```

Figura 1: fichero Docker-compose.yml

8.2 Middlewares:

Se han implementado una serie de middlewares para proteger determinadas URLs de forma que únicamente se pueda acceder a dichas URLs si el usuario está autenticado o si el usuario tiene un rol específico (**comercio, técnico o administrador**). En el caso de que se trate acceder a una URL protegida por uno de estos middlewares sin tener los permisos necesarios, entonces se mostrará un error **HTTP 404 Not Found**.

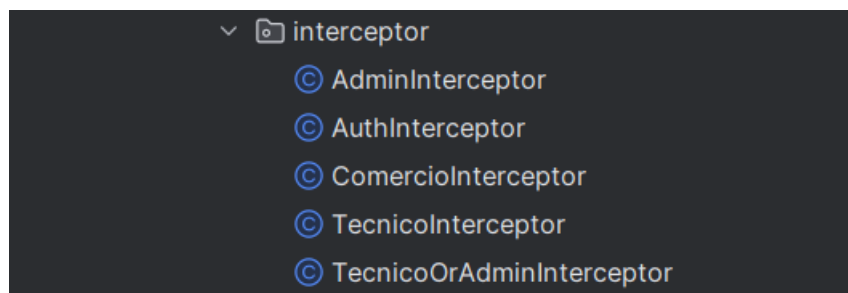
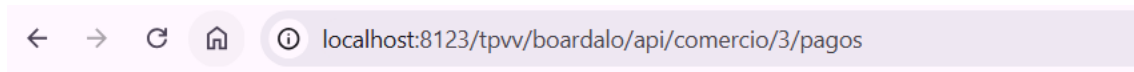


Figura 2: Ficheros de Middleware



404 - Página No Encontrada

[Volver al Inicio](#)

Figura 3: Error de acceso a una URL. No se tienen los permisos suficientes

8.3 Tests Unitarios:

Se ha implementado una suite de tests unitarios para comprobar toda la capa de repositorio y parte de funcionalidades de la capa deservicio. Con esto agregamos una capa de robustez a la aplicación y nos aseguramos de poder detectar bugs y regresiones a la hora de realizar actualizaciones futuras en la misma.

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 92, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 35.120 s
[INFO] Finished at: 2025-01-17T19:17:12+01:00
[INFO] -----
```

Figura 4: Resultado de ejecutar los tests de la aplicación.

9. Mejoras y ampliaciones

Aquí vamos a listar algunas mejoras que podría tener el sistema para estar más completo:

- **Estadísticas:** Usando la api de Google charts⁸, podríamos hacer un apartado para administrador con una pantalla más detalladas de estadísticas de cada comercio, técnico, pagos etc...
- **Seguridad en el formulario de pago:** Añadir una tabla extra para reforzar la seguridad a la hora de realizar un pago para un comercio
- **Mensajes de confirmación:** Un mensaje que indique si ha habido problemas en el servidor a la hora de rellenar ciertos formularios, o lo contrario, confirmaciones de que se ha creado correctamente la entidad del formulario

⁸ [Esta](#) es la documentación de la api de Google charts

