

# **Proyecto: Iteración 3. Session Management and Testing**

## **GRUPO 03:**

Mariajose Zuloeta Brito

Adrián Lima García

Samuel Lorenzo Sánchez

29 de noviembre de 2024

## 1. Sign Up

La vista de registro de usuarios está diseñada para facilitar la creación de cuentas para los usuarios que van a comprar en la página. Se utiliza Vuetify para implementar componentes ya creados y estéticos. Los campos del formulario están etiquetados para garantizar la accesibilidad web, asegurando que los usuarios comprendan fácilmente la información requerida. Además, se añadieron validaciones básicas, como la verificación del formato del correo, para mejorar la usabilidad.

Entre las principales dificultades encontradas estuvo la integración de la funcionalidad del backend. Usar fetch para enviar los datos del formulario al servidor requirió manejar posibles errores, como respuestas no exitosas o problemas de conectividad. Además, estuvo complicado implementar la gestión del estado global con Pinia para guardar los datos del usuario después de recibir un token JWT del servidor. Decodificar el token y asegurarse de que el almacenamiento local funcione correctamente también implicó un proceso de prueba y error.

## 2. Análisis y diseño

En este sprint, las tareas incluyen la creación de vistas dinámicas para las subcategorías de productos y mascotas, así como el desarrollo de rutas de backend para gestionar datos y garantizar la funcionalidad del sistema. También se abarcan tareas, como la implementación de dashboards para distintos tipos de usuarios (administradores, proveedores y compradores), buscando un enfoque personalizado del sitio web según roles.

La modularidad destaca en este sprint porque se han incluido tareas específicas que pueden desarrollarse y probarse de manera independiente. Por ejemplo, el desarrollo de una ruta para subir imágenes y la implementación de filtros para mostrar únicamente mascotas en adopción muestran cómo se busca mejorar la interacción del usuario con la plataforma, así como la organización de los datos en el backend. Además, las tareas relacionadas con la creación de componentes para subcategorías reflejan la importancia de ofrecer una navegación más amplia dentro del sistema, mejorando la usabilidad del sitio.

## 3. Funcionalidades agregadas

### Frontend:

- Vista de subcategorías de productos
- Componente de subcategorías de productos y mascotas
- Vista de administrador(dashboard)
- Vista de comprador (dashboard)

- Vista de proveedor (dashboard)

- Vista del panel de control

#### **Backend:**

- Diseño de la estructura de carpetas.
- Ruta que permita subir imágenes de las mascotas y productos.
- Mostrar únicamente mascotas en adopción.

## **4. Implementación de las funcionalidades**

En esta sección hablaremos de lo desarrollado en cada funcionalidad, además de los problemas u obstáculos que nos encontramos en el desarrollo de estas:

## **5. Funcionalidades agregadas**

### **5.1. Vista de subcategorías de productos**

En cuanto a la vista de las subcategorías de productos, finalmente se optó por modificar la vista de los productos (donde se encuentra el listado) ajustando el grid de tal forma que éstas subcategorías aparezcan junto al filtrado (y en el caso de dispositivos con pantallas más pequeñas al inicio de la página).

Lo que se hizo fue que en lugar de redirigir a otra página, se trabaje en la misma página cambiando de forma reactiva el contenido del listado en función de la sección que se elija, reaccionando al evento emitido por el componente que ofrece las palabras clave por las que se filtrarán, pendiente a futuro de permitir en el backend el filtrado base por dichas palabras que serán pasadas por query.

### **5.2. Componente de subcategorías de productos y mascotas**

El componente fue finalmente únicamente creado para la sección de productos por decisión conjunta del equipo.

En cuanto a la creación de este componente, se decidió utilizar el componente v-list-group que provee Vuetify. Los distintos elementos (en éste caso categorías y subcategorías) se basaron en un conjunto de datos JSON el cual se definió con que cada sección tiene un nombre, un conjunto de palabras clave asociadas (para en el futuro utilizarlas como filtrado base en el backend) y un conjunto de subcategorías que contienen objetos categoría como los que se acaban de mencionar.

Se tuvieron que aplicar algunos cambios en la plantilla que se le provee al slot del componente v-list-group de tal forma que tanto el texto de cada categoría como el botón desplegable actuaran de forma independiente: el texto emite a su

En una primera versión del componente tuve un problema a la hora de iterar sobre el objeto para crear el componente de lista desplegable ya que al haber algunas subsecciones que se repiten en las distintas secciones principales (por ejemplo la subsección "Perros" se encuentra tanto en "Salud como en "Accesorios" en "Alimentación"), lo cual daba a que al hacer click en el desplegable "Perros" de una, se desplegara en la otra también simultáneamente (ya que el identificador era solamente el mismo nombre). La solución fue que los identificadores asociados a cada uno en el "value" que se le pasa a la prop del componente v-list-group, especificando en dicho value si es categoría o subcategoría y su identificador para que se diferencien `¡tipo!-índicecategoría!-índicesubcategoría!-índicesubsubcategoría!`. Gracias a ésto, el array de "values" llamado "open" que se le pasa al componente, que es el que guarda de forma reactiva los que están abiertos, puede diferenciar correctamente entre ellos.

con la lista de compradores existentes, permitiendo editar o eliminar cada uno mediante botones de acción.

Una de las principales dificultades es la validación de los campos del formulario. Por ejemplo, asegurarse de que los usuarios ingresen información correcta, como un correo electrónico válido o un número de teléfono con el formato adecuado. La comunicación con el servidor fue otra dificultad. Cuando se intenta agregar, editar o eliminar compradores, pueden surgir errores si la URL es incorrecta o si el servidor no responde con los datos esperado.

### 5.5. Vista de proveedor (dashboard)

El componente permite registrar, editar y eliminar proveedores. Incluye un formulario con campos para el nombre, usuario, contraseña, correo electrónico y teléfono del proveedor, todos con validaciones específicas. Dependiendo de si se está agregando un nuevo proveedor o editando uno existente, el formulario muestra un botón que dice Registrar.<sup>o</sup> .Actualizar”. Además, el componente cuenta con una tabla que lista todos los proveedores registrados, con opciones para buscar, editar o eliminar cada proveedor mediante botones de acción.

Entre las principales dificultades al desarrollar este componente fue manejar la edición de proveedores. Al rellenar el formulario con datos existentes, fue necesario asegurarse de que los cambios solo se enviaran si se modificaba algún campo. Esto implicó comparar el estado original del proveedor con los nuevos datos ingresados, lo que requirió lógica adicional para evitar actualizaciones innecesarias.

Otra dificultad fue implementar correctamente las validaciones, especialmente para el campo del correo electrónico y el número de teléfono. Se tuvieron que definir reglas específicas para garantizar que el formato de estos datos fuera válido, lo que involucró el uso de expresiones regulares. Además, gestionar errores durante el envío de datos al servidor, como problemas de red o respuestas inesperadas, también fue un reto importante.

### 5.6. Vista del panel de control

Este componente es un panel de control dinámico desarrollado con Vue y Vuetify, que permite a los usuarios navegar entre distintas secciones como .Analíticas”, .Añadir Comprador”, .Añadir Mascotas más, dependiendo de su tipo de usuario (administrador, comprador o proveedor). Incluye una barra superior con un buscador y una opción para mostrar información del usuario actual, además de un menú lateral que se despliega para acceder a las distintas funciones del panel. El contenido principal cambia dinámicamente según la opción seleccionada en el menú.

Una de las principales dificultades fue gestionar el contenido dinámico del panel. Cada tipo de usuario tiene acceso a diferentes opciones, por lo que fue necesario implementar lógica que selecciona automáticamente el componente adecuado al iniciar sesión. Esto requirió crear métodos para verificar el tipo de usuario y configurar el componente inicial de manera correcta.

## 5.7. Diseño de la estructura de carpetas

Para optimizar la gestión y entrega de imágenes de productos y mascotas, se ha integrado la plataforma Cloudinary en el backend de nuestra aplicación. Cloudinary actúa como un Content Delivery Network (CDN) especializado en imágenes, proporcionando funcionalidades de subida, transformación y optimización de imágenes sobre la marcha.

Las imágenes se suben a Cloudinary a través de la API REST de Cloudinary, utilizando las rutas predefinidas `/pets/pet.id/images` y `/products/product.id/images`. Una vez subidas, Cloudinary genera automáticamente varias versiones de cada imagen con diferentes tamaños y formatos, optimizando así su carga en diferentes dispositivos y pantallas.

La integración con Cloudinary se ha realizado utilizando el SDK de Node.js, lo que permite una fácil integración con nuestro backend desarrollado en Express.js. Además, se ha configurado Cloudinary para utilizar una política de seguridad robusta, asegurando que solo usuarios autorizados puedan acceder y modificar las imágenes. La limitación con la que contamos es que la licencia gratuita solo nos permite subir como mucho 100 MB de imágenes, vídeos, etc.

El principal objetivo de esta tarea fue comprender en profundidad el funcionamiento de Cloudinary, con especial énfasis en los 'upload presets' y los archivos de configuración por defecto. Se investigó cómo integrar esta herramienta en el código base existente, evaluando las modificaciones necesarias para adaptar la aplicación a las nuevas funcionalidades de Cloudinary.

## 5.8. Ruta que permita subir imágenes de las mascotas y productos

Con el objetivo de mejorar la experiencia del usuario, se habilitó la carga de imágenes tanto para productos como para mascotas. Se utilizó Multer para controlar el tamaño de las imágenes y garantizar un rendimiento óptimo. Al finalizar la carga, se genera una URL pública única para cada imagen, la cual se asocia al registro correspondiente. Esta URL permite mostrar la imagen de forma dinámica en la aplicación.

## 6. Mostrar únicamente mascotas en adopción

Se implementó un nuevo endpoint, `/pets/adoptionpets`, con el objetivo de filtrar y mostrar únicamente las mascotas disponibles para adopción. Este endpoint itera sobre todos los administradores, recopilando la información de sus mascotas. Al realizar esta consulta, se garantiza que solo se muestren las mascotas que no tienen un dueño asignado.

## 7. Implementación de los tests

Se implementaron en la iteración anterior.

## 8. Panel del sprint 2 y backlog del spring 3

En el spring anterior:

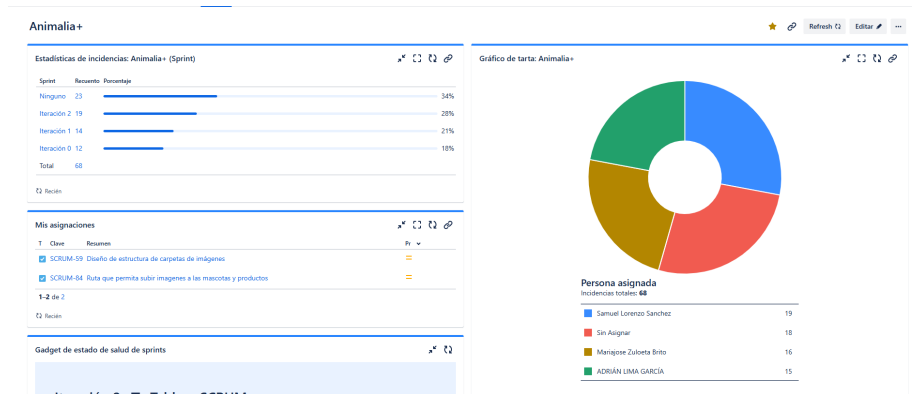


Figura 1: Imagen que muestra las cargas de trabajo de cada miembro del equipo, tareas asignadas al usuario y el estado de salud del spring

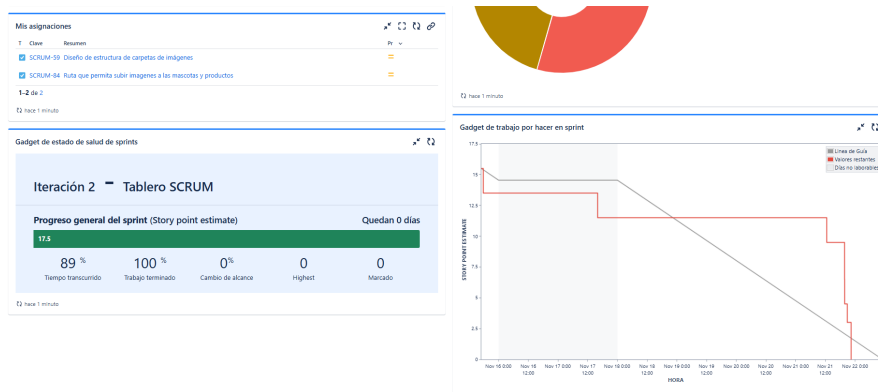


Figura 2: Imagen que muestra el spring burndown chart

Mientras que el backlog de la iteración 1 es:

Iteración 3

22 Nov – 29 Nov

(11 issues)

2

12.5

0

Complete sprint

...









<input checked="" type="checkbox"/>	SCRUM-59	Diseño de estructura de carpetas de imágenes	BASE DE DATOS	IN PROGRESS ▾	3	MB
<input checked="" type="checkbox"/>	SCRUM-77	Habilitar ruta para GET de productos y mascotas que traiga un orden	BACKEND	TO DO ▾	1	
<input checked="" type="checkbox"/>	SCRUM-79	Vistas de subcategorías de productos	FRONT END	IN PROGRESS ▾	2	
<input checked="" type="checkbox"/>	SCRUM-80	Vistas de subcategorías de mascotas	FRONT END	IN PROGRESS ▾	2	
<input checked="" type="checkbox"/>	SCRUM-81	Componente de subcategorías de productos y mascotas	FRONT END	IN PROGRESS ▾	0.5	
<input checked="" type="checkbox"/>	SCRUM-62	Vista de administrador (dashboard)	FRONT END	IN PROGRESS ▾	1	
<input checked="" type="checkbox"/>	SCRUM-84	Ruta que permita subir imagenes a las mascotas y productos	BASE DE DATOS	TO DO ▾	1	MB
<input checked="" type="checkbox"/>	SCRUM-85	Mostrar únicamente mascotas en adopción	BACKEND	TESTING ▾	1	MB
<input checked="" type="checkbox"/>	SCRUM-60	Vista de comprador (dashboard)	FRONT END	TESTING ▾	1	
<input checked="" type="checkbox"/>	SCRUM-61	Vista de proveedor (dashboard)	FRONT END	TESTING ▾	1	
<input checked="" type="checkbox"/>	SCRUM-69	Vista del panel de control	FRONT END	TESTING ▾	1	

Figura 3: Imagen que muestra el Backlog del spring 1