**PRUEBA TÉCNICA - MERCADOLIBRE**

**Descripción del problema**: Desarrollar una aplicación con tres componentes gráficos para la búsqueda, visualización de resultados y descripción del detalle de producto. Consumible a través de un servidor con los Endpoints solicitados. La información debe ser consumida a través del API proporcionada de Mercadolibre.

**API a consumir:** [https://api.mercadolibre.com/](https://api.mercadolibre.com/sites/MLA/search?q=:query)

**PROPUESTA DE DESARROLLO**

**Ingeniería de Software:** Para el desarrollo de la propuesta se realizará un análisis, en primera instancia, directamente desde la ingeniería de software para abordar el diseño, casos de uso, arquitectura y organización de la aplicación. Para fines netamente técnicos, no se abordará la descripción desde un modelo de vistas de arquitectura (4+1), sino una abreviación contando con diagramas de casos de uso, diagramas de secuencia y diagrama de despliegue (Vista lógica, de desarrollo y escenario).

**Levantamiento de Requerimientos:**

**Requerimientos Funcionales:**

* Construcción de tres componentes (vistas) para la aplicación: Búsqueda, resultados y detalle de producto.
* Cada componente debe tener una ruta diferente pasando información en la Url.
* En la caja de búsqueda se puede ingresar cualquier información y debe traer resultados en una vista aparte.
* La búsqueda debe traer únicamente 4 productos en la vista de resultados.
* Cada resultado debe ser accesible para poder ingresar a la vista de detalle.
* El usuario debe poder ver un ítem con mayor información al hacer clic sobre él.
* La respuesta del API debe estar simplificada e incluir los datos del desarrollador.
* El usuario puede ingresar directamente desde la vista de vista de detalle el ID como un queryParam y acceder a la información.

**Requerimientos No Funcionales:**

* La aplicación debe estar orientada a la escalabilidad y rendimiento.
* Debe ser intuitiva.
* Se deben manejar las tecnologías en Frontend de CSS (o SASS), HTML y JS (Deseable React).
* En la parte del servidor se debe utilizar NodeJs y Express como librería de servicio web.

**Casos de Uso:**

* CU-01: Visualización Caja de Búsqueda, resultados y detalle de producto.
* CU-02: Buscar producto.
* CU-03: Visualizar lista de productos.
* CU-04: Visualizar producto específico.
* CU-05: Consultar producto por URL.
* CU-06: Estandarizar la información del API de Mercadolibre.

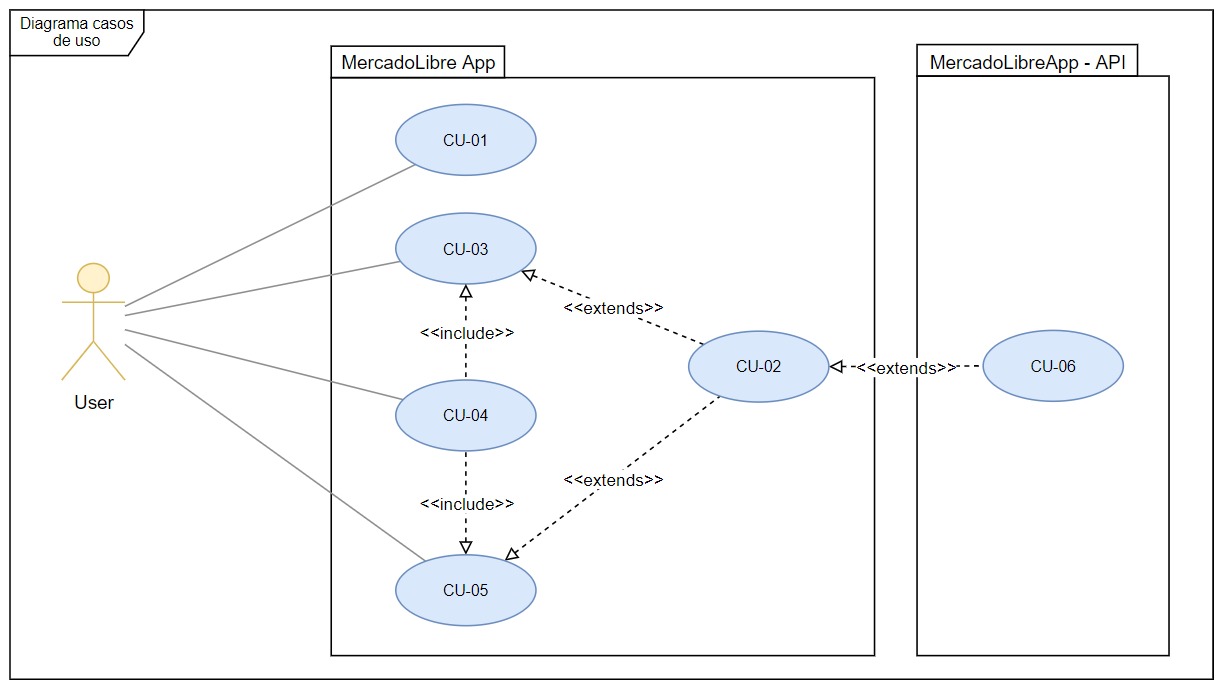


Ilustración . Diagrama de Casos de Uso.

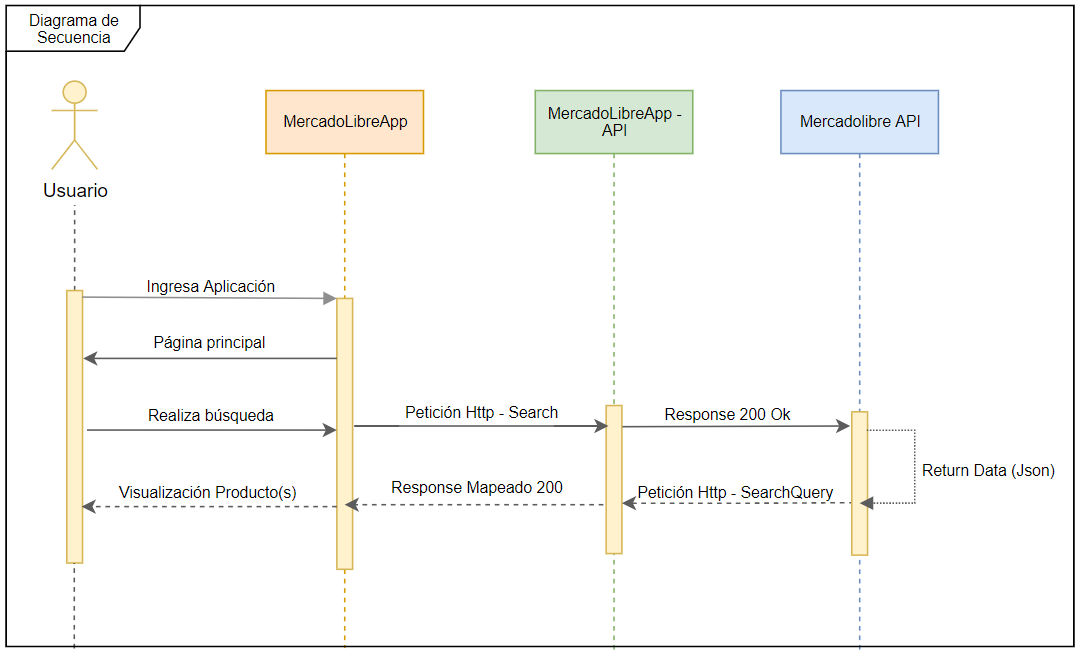


Ilustración . Diagrama de Secuencia (Flujo principal)

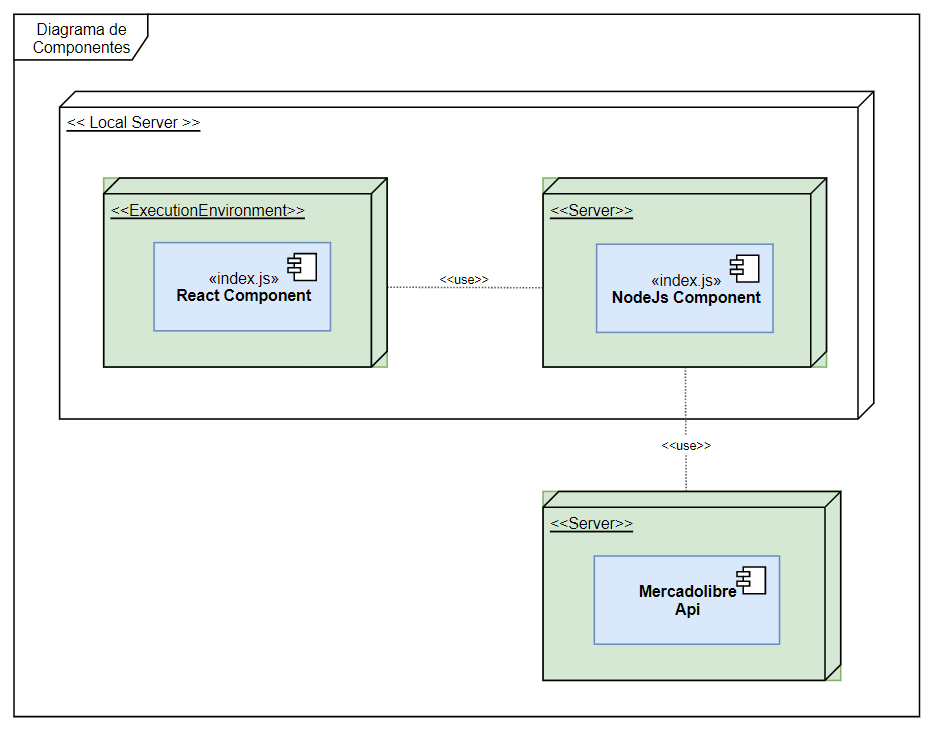


Ilustración . Diagrama de Componentes

**BACKEND**

El desarrollo del Backend consta de un servidor instanciado en NodeJs con ayuda de la librería Express y corriendo en el lenguaje Typescript, las peticiones Http fueron realizadas con la librería Axios y se utilizó la librería dotenv para ocultar la configuración de ambientes (Se dejó el archivo público en el repositorio para esta prueba). Finalmente, para temas de desarrollo se utilizó la librería nodemon para tener un Live-server. Se utilizaron las siguientes versiones de las librerías:

* **NodeJs**: 14.15.4
* Typescript: 4.2.4
* Express: 4.17.1
* Axios: 0.21.1
* Dotenv: 8.2.0
* Nodemon: 2.0.7
* Jest: 26.0.23
* Mocha: 8.2.2

**Aspectos importantes:** La aplicación tiene una arquitectura desacoplada por componentes para poder dar escalabilidad y mantenimiento más fácil.

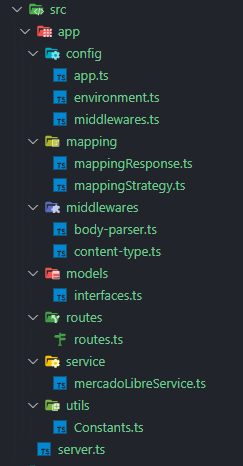


Ilustración . Estructura de aplicación Backend

En el apartado de configuración se centraliza la implementación de ambientes, middlewares y del servidor Express. En la carpeta de mapping se realizó un patrón de diseño comportamental Strategy para definir los mapeos según cada endpoint de la aplicación (Obtener todos los productos o un solo producto).

La carpeta Middlewares contiene la información correspondiente a los interceptores de la aplicación, no se agregó ninguno nuevo. En models se encuentran las interfaces o modelos de datos que tiene cada response del API, con el objetivo de mantener un estándar en los datos. En routes se tiene la instancia de las rutas de la aplicación: Default, obtener ítems y obtener un ítem en específico, así como su redirección. En service se encuentra el llamado Http a la API de MercadoLibre, así como la instancia de la clase Strategy para definir el mapeo a realizar y dejar modificable la opción para escalar en caso que en un futuro existan más métodos de mapeo.

También se encuentran métodos auxiliares para realizar la consulta Http de Categorías y Descripción, así como un catch de errores por cada método. Finalmente, una carpeta de Utils donde están las constantes utilizadas como las URL.

**Endpoints:** Todas las operaciones se ejecutan bajo métodos GET.

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Descripción** | **Ejemplo** |
| {{localhost}} / | Ruta por defecto que retorna un mensaje plano. | http://localhost:3001/ |
| {{localhost}} /api/items | Consulta todos los elementos dado un valor. Se deben enviar por query en la URL con valor “q” el objeto de búsqueda. | <http://localhost:3001/api/items?q=batman> |
| {{localhost}} /api/item/{{id}} | Consulta un producto en específico dado su ID. Se debe enviar como un queryParam. | http://localhost:3001/api/items/MC321654 |
| {{localhost}} /\* | Cualquier otra ruta que se ingrese en el API será redirigida a la URL inicial. | <http://localhost:3001/nuevaRuta> |

**Ejecución:** Para ejecutar el API por separado (Sin el Docker-compose), se debe ejecutar el comando npm run dev y el servicio será levantado en el port 3001.

**Pruebas unitarias:** Se realizaron pruebas unitarias para cubrir la principal funcionalidad del API que es el mapeo de información. Se obtuvo un Coverage de 63.55% que debe ser completado.

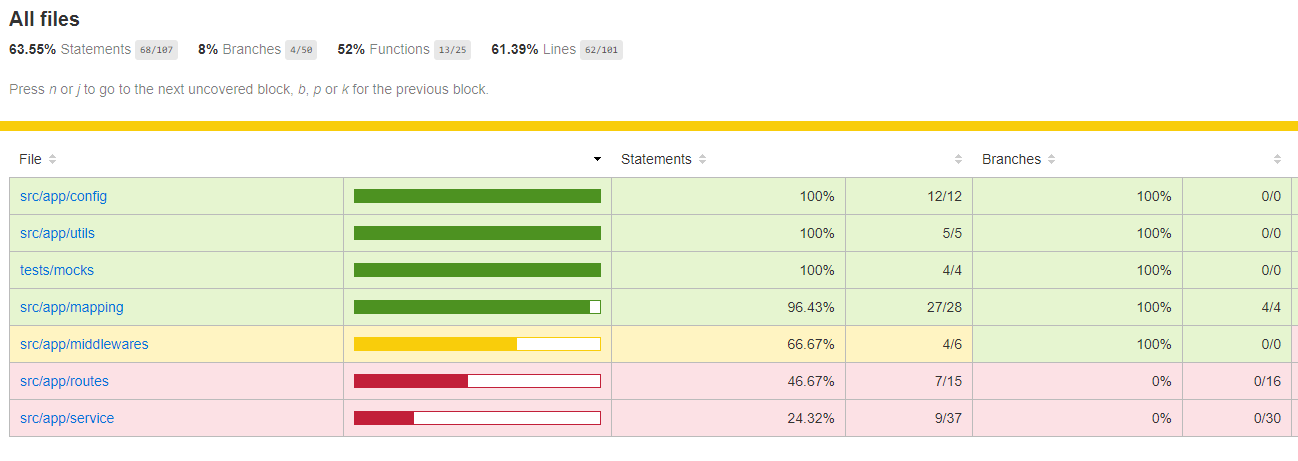


Ilustración 5. Pruebas unitarias, cobertura: 63.55%

**FRONTEND**

En el lado de la aplicación, se realizó a través de ReactJS y NextJs para tener la escalabilidad y SEO disponible desde el principio. Se utilizó Javascript como lenguaje de programación, HTML y SCSS. Se utilizaron Hooks propios y de la aplicación bajo la filosofía “Keep it simple”; así como helpers (servicios y pipes) y componentes siguiendo una tendencia de desarrollo Frontend de tipo atómico (Atomic Desing). Se utilizaron las siguientes tecnologías:

* **NodeJs**: 14.15.4
* ReactJs: 17
* NextJs: Latest
* Sass: 1.32
* React Lottie: 1.2.3
* Jest: 26.6.3
* React Testing Library: 11.2.7
* Identity-obj-proxy: 3.0.0

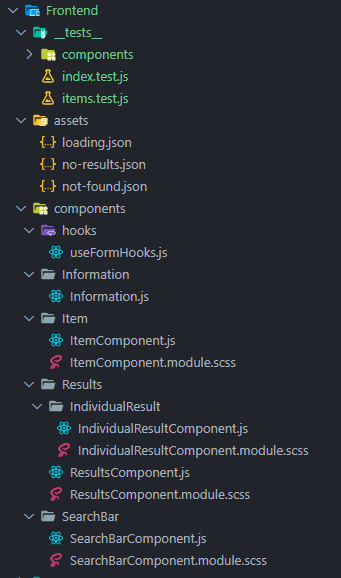
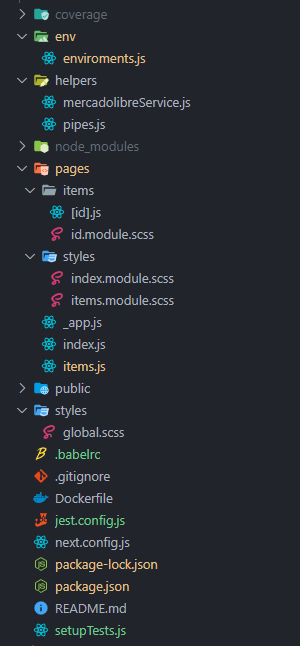
 

Ilustración 6. Estructura de aplicación Frontend

En primer lugar se tienen los assets generales, conformados por las animaciones lottie correspondiente a la pantalla de bienvenida, una búsqueda sin resultados y una página no encontrada. Seguida de los componentes donde se modularizó cada parte de la aplicación. Por un lado se tiene el componente “Information,” el cual contiene la meta información para el renderizado del lado del servidor y favorecer el SEO.

Se tiene una carpeta de Hooks custom (Utilizados en la barra de navegación), y los componentes aislados para su integración. Para mayor información de los componentes, remitirse al GitHub.

En la carpeta ENV se encuentra un archivo de configuración para cuando la aplicación deba ser escalada y tenga diferentes ambientes, poderlos reemplazar para compilar versiones y que no se exponga información confidencial cuando se suba a algún repositorio público. La carpeta pages contiene las direcciones de la aplicación, los cuales son utilizados para redirigir la información, se aplicaron conceptos como shallow-routing y dynamicRouting.

Cada componente/page, contiene un archivo correspondiente a estilos en SASS y a nivel de la aplicación se tiene un archivo de configuración de estilos general y transversal a toda la aplicación.

La carpeta \_\_test\_\_ contiene las pruebas unitarias de la aplicación, a modo general se encargaron de cubrir el renderizado de los componentes y de establecer la suit de pruebas para posteriores desarrollos.

**Evaluación de Rendimiento:** Para la finalización del ejercicio práctico se realizó una auditoria de interfaz a través de LightHouse por cada pantalla, obteniendo los siguientes resultados:



Ilustración 7. Pantalla inicial

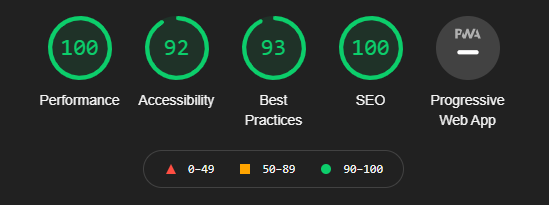
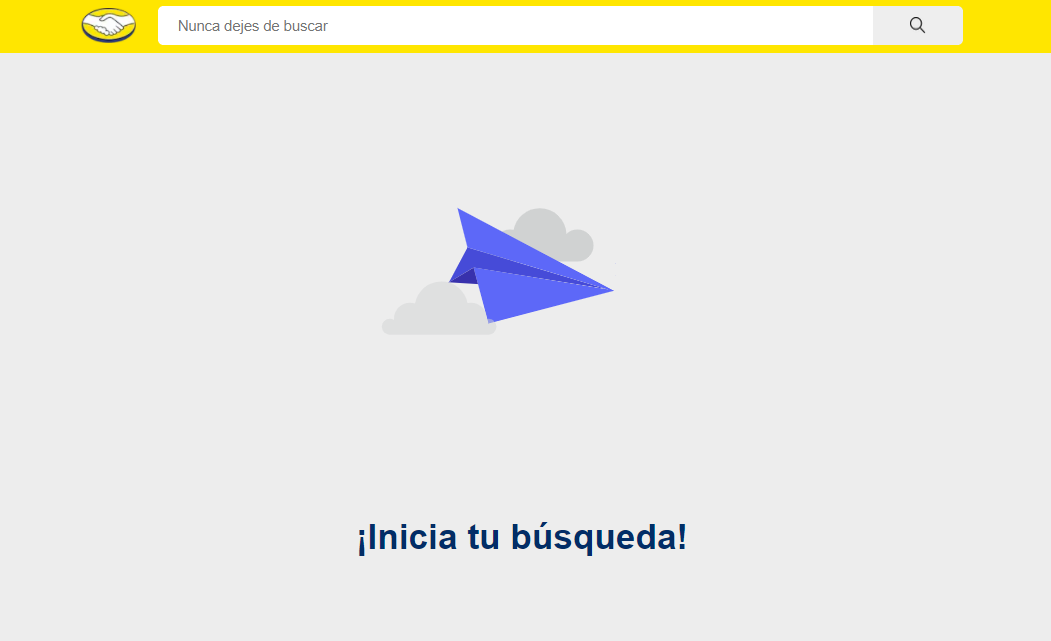


Ilustración 8. Rendimiento pantalla inicial



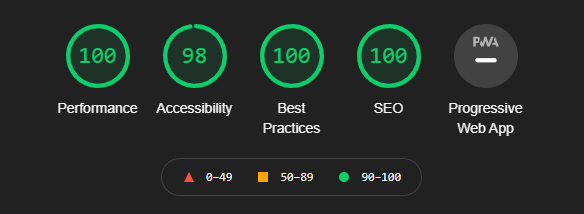
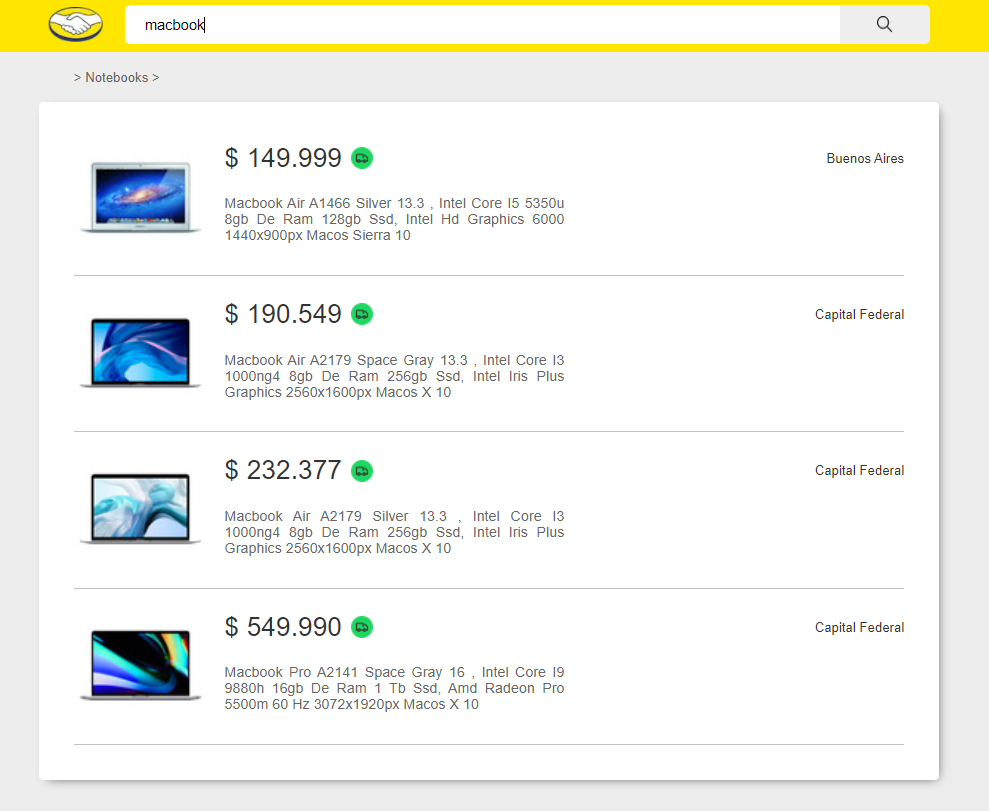


Ilustración 9. Rendimiento página de búsqueda inicial



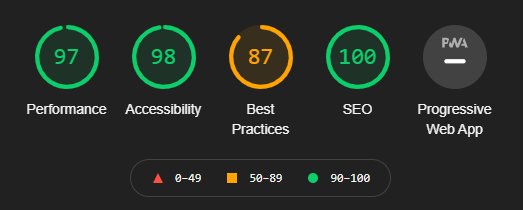
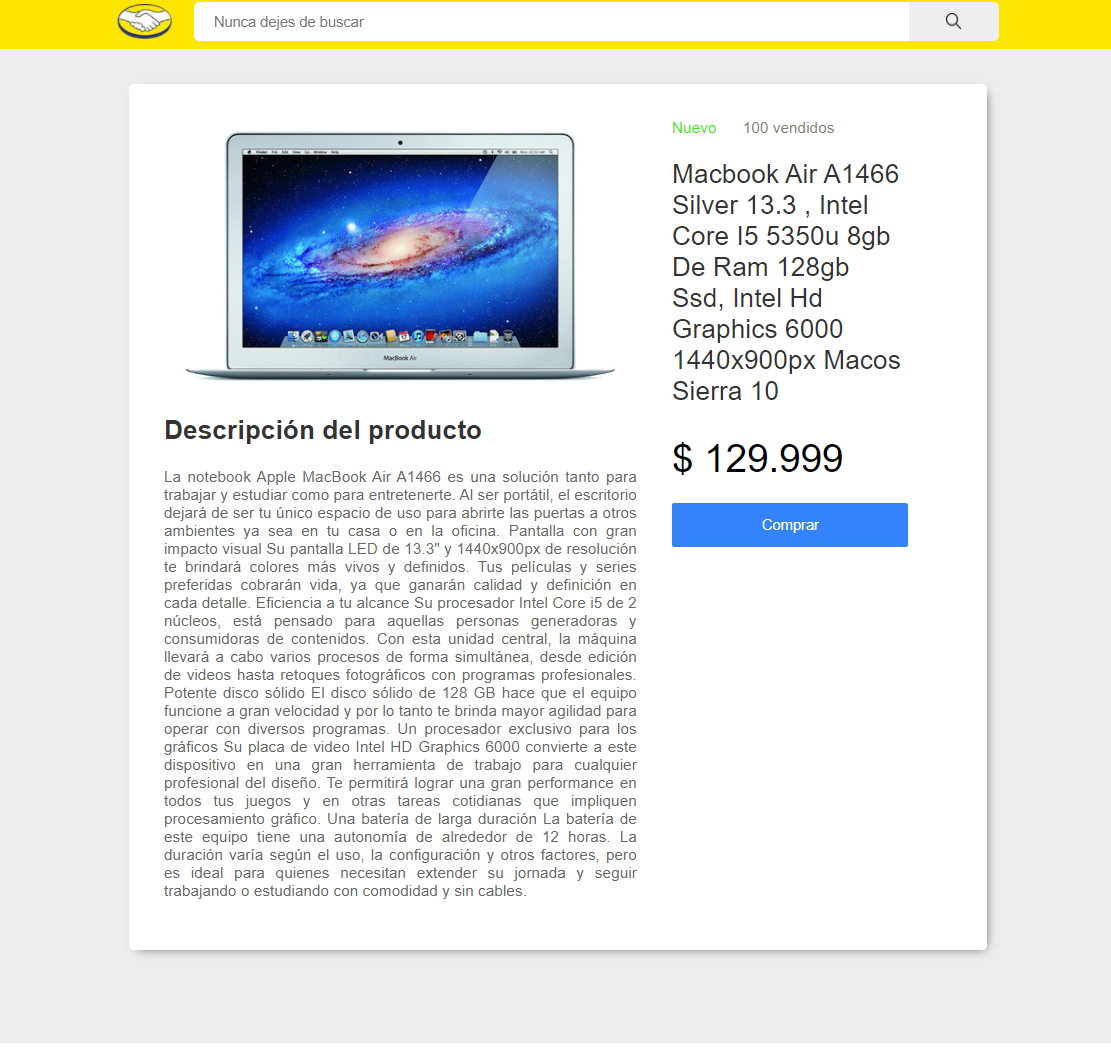


Ilustración 10. Rendimiento pantalla de búsqueda de elementos



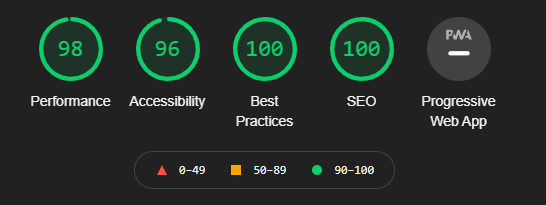
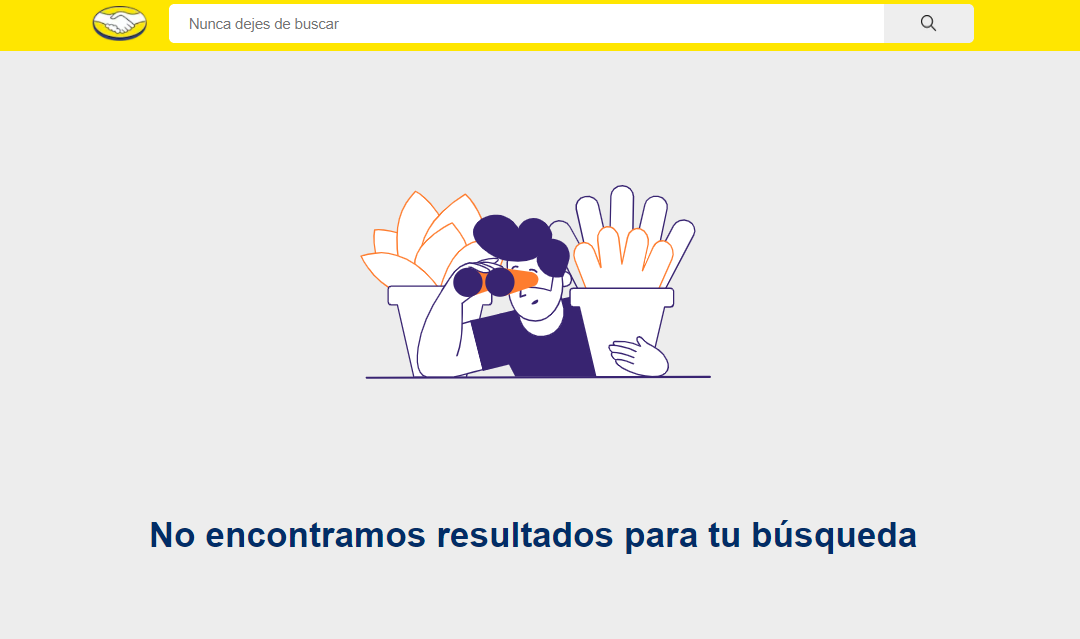


Ilustración 11. Pantalla detalle de producto



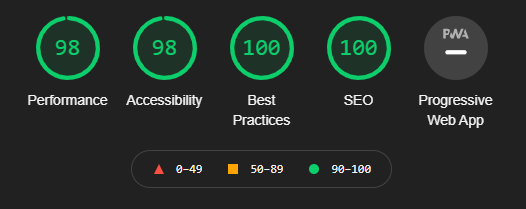


Ilustración 12. Pantallas de animación para resultados

**Ejecución:** Para ejecutar la aplicación debe realizarse con los comandos “npm run build” y “npm run start”, sin embargo, también puede realizarse desde el archivo Docker-compose en el directorio raíz del repositorio de GIthub.

**Pruebas Unitarias:** Se realizaron pruebas unitarias a nivel de integridad del renderizado de los componentes, sin embargo, es necesario consolidar algunas pruebas, agregar specs y consolidar un poco más cada prueba.

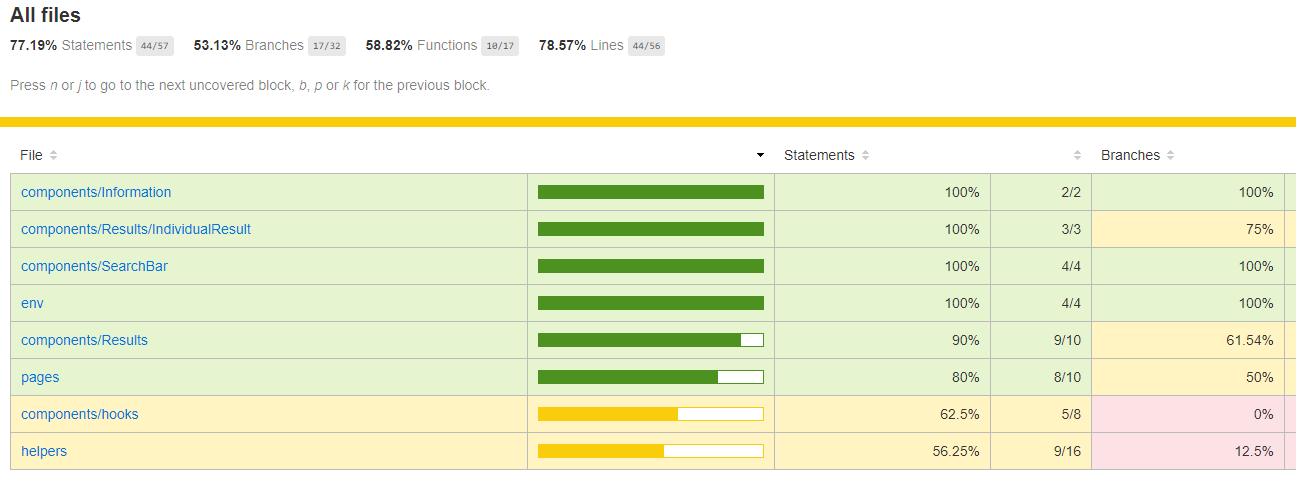


Ilustración . Cobertura Mercadolibre-app: 77.19%

**EJECUCIÓN**

Para levantar los servicios y probar la aplicación debe ejecutarse el archivo Docker-Compose con el siguiente comando:

**OPORTUNIDADES DE MEJORA EN GENERAL:**

* + Incrementar la cobertura de pruebas unitarias y mejorar la robustez de las mismas.
  + Ajuste de estilos.
  + Inclusión de Redux (Para escalabilidad).
  + Extensión a otros endpoints (Backend).