



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA



Departamento de Informática
Universidad Técnica Federico Santa María

Documentación

Web de tienda en línea

Fashion Trend

Integrantes	Denyss Carcamo	202073035-0
	Gastón Quevedo	201830002-0
	Jorge Roldán	202073116-0
Fecha	22/05/2024	
Profesor	Oscar Reyes H.	
Asignatura	Pruebas de Software	
Grupo	5	

INF-331

Índice

1. Introducción	2
2. Tecnologías	2
3. Alcances de la herramienta	3
4. Descripción	4
4.1. Backend	4
4.2. Frontend	6
4.3. Dependencias herramienta con aplicación	8
5. Testing	9
6. Problemas encontrados y resueltos	12

1. Introducción

La tienda en línea "Fashion Trend" se especializa en la venta de ropa y accesorios de moda, pero ha enfrentado varios desafíos que afectan tanto la experiencia del cliente como la eficiencia operativa interna. Los clientes frecuentemente tienen dificultades para encontrar productos específicos y obtener información detallada, lo que impacta negativamente en la satisfacción del cliente y las ventas. Además, la gestión manual del inventario y los detalles de los productos ha resultado ser un proceso lento y propenso a errores.

Para abordar estos problemas, "Fashion Trend" ha decidido desarrollar un sistema de gestión de inventario y catálogo en línea. Este sistema permitirá a los clientes buscar productos por categoría, tipo, tamaño y color, además de ver la disponibilidad de productos en tiempo real y obtener detalles completos sobre cada artículo, incluyendo imágenes, descripciones y precios.

El sistema también ofrecerá una interfaz de administración para el equipo de la tienda. Desde esta interfaz, los empleados podrán agregar nuevos productos al catálogo, editar detalles existentes, gestionar cantidades en inventario y eliminar elementos que ya no estén disponibles. Esta funcionalidad mejorará significativamente la eficiencia operativa y reducirá los errores asociados con la gestión manual.

2. Tecnologías

En el panorama digital actual, es crucial desarrollar una aplicación web escalable y de alto rendimiento para manejar las crecientes demandas de los usuarios y garantizar una experiencia de usuario perfecta. La elección de tecnologías como MongoDB, Fastify, React y Jest garantiza que la aplicación no solo sea eficaz, sino también mantenible y fácil de ampliar a medida que evolucionan los requisitos.

- **MongoDB:** Es una base de datos NoSQL conocida por su escalabilidad y flexibilidad. Utiliza un modelo sin esquema, que permite un rápido desarrollo e iteración de aplicaciones. MongoDB almacena datos en un formato similar a JSON (BSON), lo que lo convierte en una opción natural para las aplicaciones web modernas y permite una integración perfecta con backends y frontends basados en JavaScript. El alto rendimiento de la base de datos se logra mediante funciones de indexación, replicación y equilibrio de carga, lo que la hace adecuada para aplicaciones que necesitan manejar grandes volúmenes de datos y altas tasas de transacciones.
- **Fastify:** Es un framework moderno para Node.js que enfatiza la velocidad y los bajos gastos generales. Fastify, conocido por su alto rendimiento, está diseñado para manejar una gran cantidad de solicitudes de manera eficiente, lo que lo hace ideal para el desarrollo backend donde el rendimiento es crítico. La arquitectura de plugins

de Fastify permite una amplia personalización y extensión de la funcionalidad sin sacrificar el rendimiento.

- **React:** Es una biblioteca de JavaScript popular para crear interfaces de usuario, particularmente aplicaciones de una sola página donde las actualizaciones de datos en tiempo real son cruciales. La arquitectura basada en componentes de React promueve la reutilización y la modularidad, lo que permite a los desarrolladores crear interfaces de usuario complejas componiendo componentes más pequeños y aislados. El uso de un DOM virtual mejora el rendimiento al actualizar solo las partes de la interfaz de usuario que han cambiado, en lugar de volver a representar toda la página. El sólido apoyo de la comunidad de React y su extenso ecosistema de bibliotecas y herramientas lo convierten en una opción confiable para el desarrollo frontend.
- **Jest:** Es un framework de prueba integral para JavaScript, diseñado para garantizar la calidad y confiabilidad del código. Funciona de inmediato con una configuración mínima, lo que permite a los desarrolladores escribir y ejecutar pruebas rápidamente. Jest es conocido por su velocidad y confiabilidad, con características como la ejecución de pruebas en paralelo y la observación inteligente de pruebas para optimizar el proceso de prueba. Admite pruebas de instantáneas, que capturan la salida renderizada de los componentes de la interfaz de usuario para realizar un seguimiento de los cambios a lo largo del tiempo. Las sólidas capacidades de mocking de Jest y su extensa biblioteca de aserciones lo convierten en una herramienta poderosa para pruebas unitarias y de integración.

3. Alcances de la herramienta

- Catálogo de productos:
 - Vista principal con todos los productos: Se muestra cada producto con su nombre, su precio y una foto que lo acompaña.
 - Vista particular con detalles de cada producto: Indica cada una de las características del producto.
- Gestión de inventario:
 - Añadir, editar y eliminar: Los empleados que hayan iniciado sesión pueden añadir productos, editar los ya existentes o eliminar alguno.
 - Visualización y búsqueda de productos registrados: Los productos son mostrados en una lista y pueden ser encontrados con una barra de búsqueda.
- Gestión de Usuarios:

- Creación usuarios con distintos roles: Se tienen 3 roles, cada uno con funciones diferentes.
- Rol Administrador: Es capaz de crear empleados, ver una lista completa de los usuarios registrados, editarlos o eliminarlos.
- Rol Empleado: Pueden agregar, editar o eliminar productos, también pueden ver una lista con cada uno de los productos existentes y buscarlos con la barra de búsqueda.
- Rol Cliente: Pueden ver los productos del catálogo y no tienen permiso para modificarlos de ninguna forma.

4. Descripción

4.1. Backend

El desarrollo del backend consta de tres tareas principales: inicialización, desarrollo de funciones principales y refinamiento de funciones principales. Cada tarea se basa en la anterior para garantizar un sistema backend robusto y escalable.

1. Inicializar el servidor

- a) Configurar la estructura del proyecto: Crear la estructura de carpetas para el backend. Compuesta principalmente por modelos, controladores, rutas, y middleware.

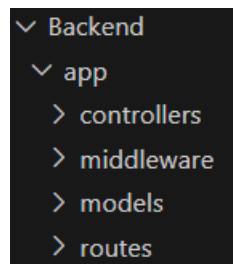


Figura 1: Estructura del backend

- b) Instalar Fastify y otras dependencias: Instalar Fastify y otras dependencias necesarias para el proyecto.

```
"dependencies": {  
  "@fastify/cors": "^9.0.1",  
  "@fastify/env": "^4.3.0",  
  "@fastify/jwt": "^8.0.1",  
  "@fastify/static": "^7.0.4",  
  "bcryptjs": "^2.4.3",  
  "dotenv": "^16.4.5",  
  "dotenv-expand": "^11.0.6",  
  "fastify": "^4.27.0",  
  "fastify-multer": "^2.0.3",  
  "fs": "^0.0.1-security",  
  "mongoose": "^8.3.4",  
  "path": "^0.12.7",  
  "uuid": "^9.0.1"  
}
```

Figura 2: Dependencias del backend

- c) Crear archivo de servidor: Crear el archivo del servidor principal y configurarlo para iniciar el servidor.
- d) Configurar variables de entorno: Configurar variables de entorno para la configuración del servidor.

```
HOST = "127.0.0.1"  
PORT = 8030  
MONGODB_URI = 'mongodb://127.0.0.1:27017/mydb'  
JWT_SECRET = "_Fashion.jwt-secret-key.Trend_"  
JWT_EXPIRATION = "4h"  
JWT_REFRESH_EXPIRATION = 86400  
FRONT_URL_CORS = "http://127.0.0.1:3003"  
ADMIN_USER_NAME = "admin"  
ADMIN_USER_EMAIL = "admin@gmail.com"  
ADMIN_USER_PASSWORD = "123admin"
```

Figura 3: Ejemplo de variables de entorno

- e) Conectarse a MongoDB: Configurar una conexión a MongoDB usando Mongoose.
- f) Inicializar usuario administrador y roles: Crear el usuario administrador inicial y definir los roles de los usuarios (user, employee, y admin).

2. Desarrollar funciones principales

- a) Registrar plugins para Fastify: Registrar los complementos (plugins) necesarios para que Fastify amplíe su funcionalidad.
- b) Crear modelos: rol, refresh token, usuario, producto, e imagen
- c) Implementar autenticación: Implementar la autenticación de usuarios mediante JSON web tokens (JWT) y tokens de actualización.

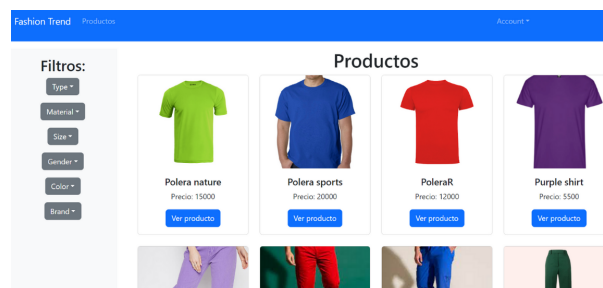
- d) Desarrollar APIs CRUD de usuario y producto: Crear APIs CRUD (Create, Read, Update, Delete) para administrar usuarios y productos.
- e) Crear rutas de usuario, producto, y autenticación: Desarrollar las rutas para las distintas operaciones que hay sobre los usuarios, productos, y autenticación (registro, inicio de sesión y actualización de tokens).
- f) Implementar hook para rutas de usuarios y productos: Utilizar middleware como hook para proteger las rutas de usuarios y productos al requerir encabezados de autorización y validar tokens JWT.

3. Refinar funciones principales

- a) Implementar Role-Based Access Control (RBAC): Mejorar la seguridad de la aplicación implementando un control de acceso basado en roles para restringir el acceso a ciertas API según los roles de los usuarios.
- b) Mejorar el manejo de errores: Mejorar el manejo de errores para proporcionar mensajes de error más informativos y garantizar respuestas de error consistentes.

4.2. Frontend

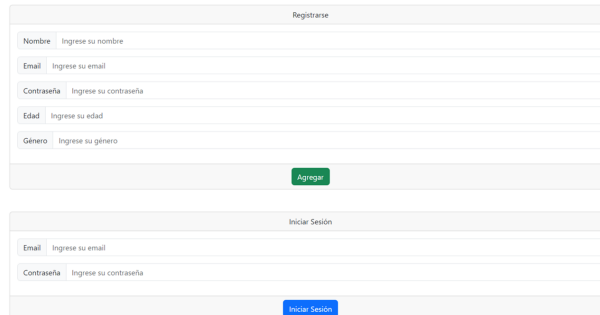
Tenemos una vista principal, a la izquierda están los filtros por cada característica de los productos y a la derecha se muestra cada producto con su nombre y su precio:



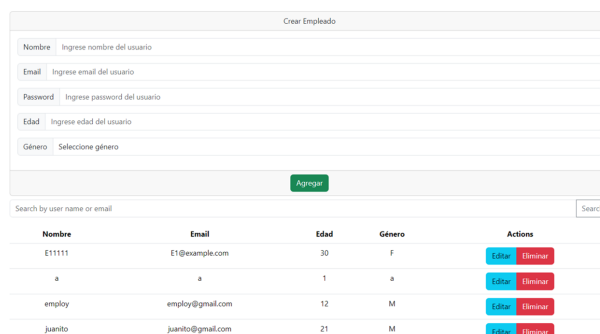
Al apretar el botón 'ver producto' se accede a una vista que contiene los detalles de este y su imagen en un mayor tamaño para que el cliente tenga la información necesaria para realizar una compra informada:



Desde la barra de navegación se puede acceder al login, en este se encuentra un formulario para registro y otro para inicio de sesión, este registro genera cuentas con el rol de cliente:

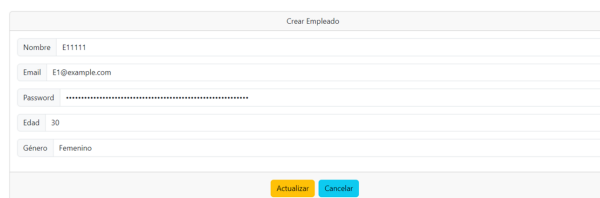


Si un administrador hace login es dirigido a una vista de administrador, en esta puede crear cuentas para empleados, ver todos los usuarios existentes y utilizar una barra de búsqueda para encontrar alguno:



Nombre	Email	Edad	Género	Actions
E11111	E1@example.com	30	F	Editar Eliminar
a	a	1	a	Editar Eliminar
employ	employ@gmail.com	12	M	Editar Eliminar
juanito	juanito@gmail.com	21	M	Editar Eliminar

Por cada usuario tiene la opción de eliminarlo o editarlo en caso de ser necesario:



Si un empleado realiza login tiene la habilidad de registrar nuevos productos en el sistema indicando cada una de sus características y una imagen:

Agregar Producto

Nombre

Descripción

Precio

Talla

Tipo

Color

Material

Genero

Marca

Cantidad

Costo

Imagen No file chosen

Estos productos se pueden visualizar en una lista y cada uno tiene la opción de ser eliminado o actualizar alguno de sus campos mediante un formulario que es desplegado al apretar el botón 'actualizar'. En esta página también se encuentra una barra de búsqueda que permite encontrar productos más fácilmente:

Search by product name

#	Nombre	Descripción	Precio	Tipo	Talla	Color	Material	Genero	Marca	Cantidad	Costo	
1	Polera nature	Polera verde bonita	15000	polera	S	verde	poliester	F	ZARA	500	150	Editar Eliminar
2	Polera sports	Polera deportiva azul	20000	polera	M	azul	poliester	F	ZARA	500	150	Editar Eliminar
3	PoleraR	polera roja	12000	polera	L	rojo	algodon	Unisex	GUCCI	35	101	Editar Eliminar
4	Purple shirt	Polera morada 100% algodón	5500	polera	S	morado	algodon	F	GUCCI	500	150	Editar Eliminar
5	pantalon	pantalon morado 100% algodón	9500	pantalon	M	morado	algodon	F	CHANEL	500	150	Editar Eliminar

Edit Product

Nombre

Descripción

Tipo

Material

Talla

Genero

Color

Marca

Cantidad

Costo

Precio

4.3. Dependencias herramienta con aplicación

La infraestructura de nuestra aplicación se dividió en tres partes, base de datos, backend y frontend.

1. Base de Datos:

a) Herramienta: MongoDB

- b) Descripción: Almacenamiento de la aplicación, aquí se guarda toda la información necesaria para el funcionamiento de la aplicación
- c) Pruebas: Aunque Jest ofrece la opción de hacer pruebas directas a MongoDB se decidió realizar pruebas a un siguiente nivel, ya que se considero que las pruebas del Backend cubrían lo necesario para probar también la base de datos.

2. Backend:

- a) Herramienta: Fastify
- b) Descripción: Sección encargada del manejo de la información, tanto almacenada como a almacenar, es la logica de la aplicación.
- c) Pruebas: Se hicieron suites de pruebas en este nivel, considerando las conexiones al nivel anterior.

3. Frontend:

- a) Herramienta: React
- b) Descripción: Encargada de entregar una interfaz para permitir la interacción con el usuario
- c) Pruebas: Actualmente no hay pruebas implementadas en esta sección, pero se encuentran planificadas y en desarrollo.

5. Testing

El desarrollo del testing se dividió en dos partes, backend y frontend. En la sección de backend se uso Jest combinado con Fastify para realizar las pruebas, en el transcurso se probaron diversos métodos, funciones y librerías auxiliares que permitieran ejecutar las pruebas, en lo cual se uso finalmente el método de inyección otorgado por Fastify, el cual consiste entregar el método y el cuerpo de la solicitud HTTP y la librería comprueba su ejecución, pero estas ejecuciones no tienen ninguna repercusión fuera de la suite, solo son validas dentro.

En este proceso se desarrollaron tres suite de casos de pruebas, la primera enfocada en el endpoint inicial, la segunda en el endpoint de productos y finalmente el endpoint de usuarios.

```
PS C:\Users\Jorge\Documents\PdS\Proyecto\backend> npm test
npm info using npm@10.7.0
npm info using node@v22.2.0

> backend@1.0.0 test
> jest

PASS tests/welcome.test.js (5.275 s)
PASS tests/products.test.js (5.176 s)
PASS tests/users.test.js (5.184 s)
A worker process has failed to exit gracefully and has been force exited. This is likely due to
testOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was
called on any active timers.

File                                     % Stmts   % Branch   % Funcs   % Lines   Uncovered Line #s
-----
All files                               22.19     0         2.22    23.18
Backend                                55.17     0        14.28    56.14    65-69,84-88,97-134
server.js                              5.17     0         0         6.18
Backend/app/controllers                 5.76     0         0        18.18    7-37
auth.controller.js                     4.25     0         0         4.59    7-162
product.controller.js                  4.34     0         0         4.7     7-148
Backend/app/middleware                  22.22     0         0        22.22
authenticate.js                        22.22     0         0        22.22    2-10
Backend/app/models                      72.41    100         0        72.41
image.model.js                         100     100        100        100
product.model.js                       100     100        100        100
refreshToken.model.js                  46.66    100         0        46.66    14-26,30
role.model.js                          100     100        100        100
user.model.js                          100     100        100        100
Backend/app/routes                      22.85     0         0        23.18
auth.routes.js                         12.5     0         0        12.82    7-72
product.routes.js                      40       100         0         40     8-13,19-28
user.routes.js                         30       100         0         30     7-14

Test Suites: 3 passed, 3 total
Tests:       22 passed, 22 total
Snapshots:   0 total
Time:        7.025 s, estimated 89 s
Ran all test suites.
npm info ok
```

En la figura se aprecia la ejecución de las tres suites, además se muestra una tabla que corresponde a la ejecución de las pruebas en todos los archivos, mostrando que líneas no son ejecutadas, porcentaje de ejecución, funciones, entre otra información que puede resultar útil al momento de analizar las pruebas

Para el endpoint inicial existían dos casos posibles, que la petición viniera con un cuerpo o no, y justamente eso fueron los casos desarrollados en esta suite.

```
PASS tests/welcome.test.js
  GET /
    ✓ without body (18 ms)
    ✓ with body (1 ms)

File                                     % Stmts   % Branch   % Funcs   % Lines   Uncovered Line #s
-----
All files                               22.19     0         2.22    23.18
Backend                                55.17     0        14.28    56.14    65-69,84-88,97-134
server.js                              5.17     0         0         6.18
Backend/app/controllers                 5.76     0         0        18.18    7-37
auth.controller.js                     4.25     0         0         4.59    7-162
product.controller.js                  4.34     0         0         4.7     7-148
Backend/app/middleware                  22.22     0         0        22.22
authenticate.js                        22.22     0         0        22.22    2-10
Backend/app/models                      72.41    100         0        72.41
image.model.js                         100     100        100        100
product.model.js                       100     100        100        100
refreshToken.model.js                  46.66    100         0        46.66    14-26,30
role.model.js                          100     100        100        100
user.model.js                          100     100        100        100
Backend/app/routes                      22.85     0         0        23.18
auth.routes.js                         12.5     0         0        12.82    7-72
product.routes.js                      40       100         0         40     8-13,19-28
user.routes.js                         30       100         0         30     7-14

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        3.892 s, estimated 5 s
Ran all test suites matching /welcome.test.js/
```

Para el endpoint de productos se realizaron mas pruebas, a continuación se describen brevemente cada caso ejecutado:

1. Se obtienen todos los productos.
2. Se obtiene un producto en base a su id.
3. Se sube un producto.
4. Se sube el mismo producto, se busca que sea rechazado por ya existir.

5. Se actualiza un producto.
6. Se actualiza un producto pero compitiendo un error en su formulario.
7. Se borra un producto.
8. Se borra un producto ya eliminado.
9. Se actualiza un producto ya eliminado.
10. Se busca un producto por su nombre

```

PS C:\Users\Jorge\Documents\PdS\Proyecto\backend> jest products.test.js
PASS tests/products.test.js (88.458 s)
  /api/products
    ✓ GET / (363 ms)
    ✓ GET /:id
    ✓ POST /
    ✓ Again POST / (1 ms)
    ✓ PUT /:id (1 ms)
    ✓ PUT with a typo in a field. Example: name as nema /:id (1 ms)
    ✓ DELETE /:id
    ✓ DELETE after DELETE /:id
    ✓ PUT after DELETE /:id (1 ms)
    ✓ GET /search/:name (1 ms)

File | % Stats | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files | 22.19 | 0 | 2.22 | 23.18 | 
Backend | 55.17 | 0 | 14.28 | 56.14 | 
  server.js | 55.17 | 0 | 14.28 | 56.14 | 65-69,84-88,97-134
Backend/app/controllers | 5.76 | 0 | 0 | 6.18 | 
  auth.controller.js | 18.18 | 0 | 0 | 18.18 | 7-37
  product.controller.js | 4.25 | 0 | 0 | 4.59 | 7-162
  user.controller.js | 4.34 | 0 | 0 | 4.7 | 7-148
Backend/app/middleware | 22.22 | 0 | 0 | 22.22 | 
  authenticate.js | 22.22 | 0 | 0 | 22.22 | 2-10
Backend/app/models | 72.41 | 100 | 0 | 72.41 | 
  image.model.js | 100 | 100 | 100 | 100 | 
  product.model.js | 100 | 100 | 100 | 100 | 
  refreshToken.model.js | 46.66 | 100 | 0 | 46.66 | 14-26,30
  role.model.js | 100 | 100 | 100 | 100 | 
  user.model.js | 100 | 100 | 100 | 100 | 
Backend/app/routes | 22.85 | 0 | 0 | 23.18 | 
  auth.routes.js | 12.5 | 0 | 0 | 12.82 | 7-72
  product.routes.js | 40 | 100 | 0 | 40 | 8-13,19-28
  user.routes.js | 30 | 100 | 0 | 30 | 7-14

Test Suites: 1 passed, 1 total
Tests: 10 passed, 10 total
Snapshots: 0 total
Time: 96.673 s
Ran all test suites matching /products.test.js/i
  
```

Para la ultima suite no existe mucha diferencia a nivel descriptivo con la anterior, la diferencia radica en que los datos a rellenar son distintos porque usuarios y productos son entidades distintas.

```

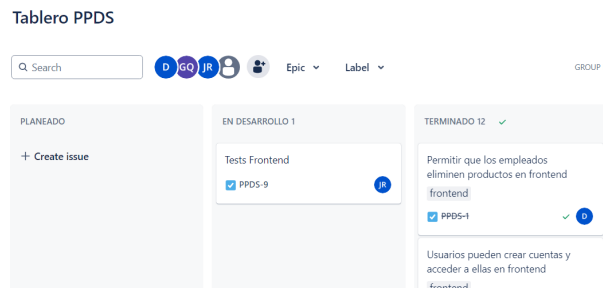
PASS tests/users.test.js
  /api/users
    ✓ GET / (30 ms)
    ✓ GET /:id (1 ms)
    ✓ POST / (1 ms)
    ✓ Again POST /
    ✓ PUT /:id (1 ms)
    ✓ PUT with a typo in a field. Example: email as ema /:id (1 ms)
    ✓ DELETE /:id
    ✓ DELETE after DELETE /:id
    ✓ PUT after DELETE /:id
    ✓ GET /search/:name (1 ms)

File | % Stats | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files | 22.19 | 0 | 2.22 | 23.18 | 
Backend | 55.17 | 0 | 14.28 | 56.14 | 
  server.js | 55.17 | 0 | 14.28 | 56.14 | 65-69,84-88,97-134
Backend/app/controllers | 5.76 | 0 | 0 | 6.18 | 
  auth.controller.js | 18.18 | 0 | 0 | 18.18 | 7-37
  product.controller.js | 4.25 | 0 | 0 | 4.59 | 7-162
  user.controller.js | 4.34 | 0 | 0 | 4.7 | 7-148
Backend/app/middleware | 22.22 | 0 | 0 | 22.22 | 
  authenticate.js | 22.22 | 0 | 0 | 22.22 | 2-10
Backend/app/models | 72.41 | 100 | 0 | 72.41 | 
  image.model.js | 100 | 100 | 100 | 100 | 
  product.model.js | 100 | 100 | 100 | 100 | 
  refreshToken.model.js | 46.66 | 100 | 0 | 46.66 | 14-26,30
  role.model.js | 100 | 100 | 100 | 100 | 
  user.model.js | 100 | 100 | 100 | 100 | 
Backend/app/routes | 22.85 | 0 | 0 | 23.18 | 
  auth.routes.js | 12.5 | 0 | 0 | 12.82 | 7-72
  product.routes.js | 40 | 100 | 0 | 40 | 8-13,19-28
  user.routes.js | 30 | 100 | 0 | 30 | 7-14

Test Suites: 1 passed, 1 total
Tests: 10 passed, 10 total
Snapshots: 0 total
Time: 2.983 s, estimated 7 s
Ran all test suites matching /users.test.js/i
  
```

6. Problemas encontrados y resueltos

1. **Organización:** El primer problema que tuvimos fue al no organizar las tareas que realizaríamos desde un inicio, esto llevó a descoordinaciones y desconocimiento sobre cómo iba el progreso del desarrollo en cada uno de los miembros del equipo. La solución fue empezar a utilizar jira, escribiendo las tareas que serían necesarias y asignándolas a cada uno de los integrantes del equipo, estas fueron organizadas en las categorías: planeado, en desarrollo y terminado. Solucionando nuestro problema de organización.



2. **Interacción entre Herramientas:** Nuestro segundo problema fue una interacción que ocurría entre nuestras herramientas involucradas, Jest, Factly y la extracción de datos de un archivo de variables de entorno (.env). Esto debido a la naturaleza de pruebas pequeñas de Jest, el cual busca la ejecución rápida de pruebas a componentes e interacciones, para lo cual siempre sugiere el uso de Mocks para omitir solicitudes e interacciones y simplemente simularlas, esto último fue la causa del problema, Factly y las variables de entorno causaban bastantes llamadas asíncronas que Jest no estaba esperando, investigando sobre esto se cambió la configuración de Jest para que reconociera la extracción de variables de entorno y las esperara y se usó una función especial de Jest que es un mock de temporizadores, inserta un temporizador falso que obliga a Jest a respetar los tiempos y nos permitió correr nuestras suites de casos de prueba correctamente.