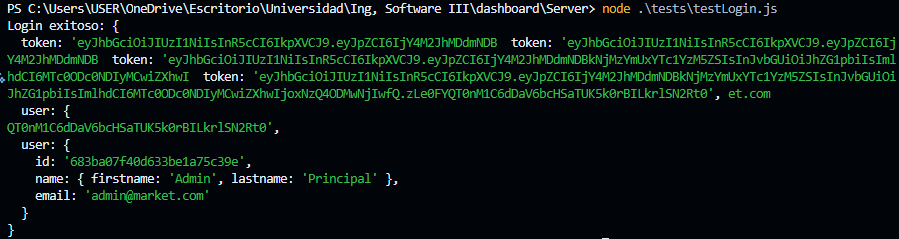
Notas:

ST = Server Test (Backend)

CT = Client Test (Frontend)



|  |  |
| --- | --- |
| ID | ST-01 |
| Descripción | Verificar que un usuario con credenciales válidas pueda autenticarse correctamente y reciba un token JWT junto con la información básica de usuario. |
| Condiciones Previas | - El servidor debe estar en ejecución en http://localhost:5000.- Debe existir un usuario con email admin@market.com y contraseña 123456789 en la base de datos.- La ruta /api/auth/login debe estar correctamente implementada y accesible. |
| Pasos de la prueba | 1. Ejecutar el script testLogin.js usando Node.js.2. El script enviará una petición POST a /api/auth/login con el email y contraseña del usuario admin.3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un objeto JSON que contiene:- Un campo token con un JWT válido.- Un campo user con los datos básicos del usuario autenticado (id, nombre, email). |
| Resultados ideales | json<br>{<br> "token": "&lt;jwt\_token&gt;",<br> "user": {<br> "id": "683ba07f40d63b3e1a75c39e",<br> "name": { "firstname": "Admin", "lastname": "Principal" },<br> "email": "admin@market.com"<br> }<br>} |
| Estado de la prueba | Exitosa |
| Nota adicional | Ninguna |



| ID | ST-02 |
| --- | --- |
| Descripción | Verificar que el sistema rechace la autenticación de un usuario que proporciona una contraseña incorrecta y devuelva un mensaje de error apropiado. |
| Condiciones Previas | - El servidor debe estar en ejecución en [http://localhost:5000](http://localhost:5000/).- Debe existir un usuario con email admin@market.com y contraseña 123456789 en la base de datos.- La ruta /api/auth/login debe estar correctamente implementada y accesible. |
| Pasos de la prueba | 1. Ejecutar el script testLoginFail.js usando Node.js.2. El script enviará una petición POST a /api/auth/login con el email válido y una contraseña incorrecta.3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un objeto JSON que contiene un mensaje de error indicando que la contraseña es incorrecta. |
| Resultados ideales | json{  "msg": "Contraseña incorrecta"} |
| Estado de la prueba | Exitosa |
| Nota adicional | La prueba confirma que el sistema maneja correctamente errores de autenticación por contraseña inválida. |



| ID | ST-03 |
| --- | --- |
| Descripción | Verificar que el acceso a una ruta protegida sea denegado si se proporciona un token JWT inválido. |
| Condiciones Previas | - El servidor debe estar en ejecución en http://localhost:5000.- La ruta protegida /api/users debe requerir autenticación mediante token JWT.- El middleware de validación de token debe estar correctamente implementado. |
| Pasos de la prueba | 1. Ejecutar el script testProtectedInvalidToken.js usando Node.js.2. El script enviará una petición GET a /api/users con un token inválido en la cabecera Authorization.3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un objeto JSON que indica que el token proporcionado es inválido. |
| Resultados ideales | json{  "error": "Token inválido"} |
| Estado de la prueba | Exitosa |
| Nota adicional | La prueba confirma que el middleware de autenticación rechaza correctamente tokens inválidos, asegurando la protección de rutas privadas. |



| ID | ST-04 |
| --- | --- |
| Descripción | Verificar que el acceso a una ruta protegida sea denegado si no se proporciona un token JWT. |
| Condiciones Previas | - El servidor debe estar en ejecución en http://localhost:5000.- La ruta protegida /api/users debe requerir autenticación mediante token JWT.- El middleware de validación debe manejar correctamente la ausencia del token. |
| Pasos de la prueba | 1. Ejecutar el script testProtectedNoToken.js usando Node.js.2. El script enviará una petición GET a /api/users sin incluir el header Authorization.3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un objeto JSON que indique que el token es requerido para acceder a la ruta. |
| Resultados ideales | json{  "error": "Token requerido"} |
| Estado de la prueba | Exitosa |
| Nota adicional | Esta prueba garantiza que el sistema protege adecuadamente las rutas privadas, incluso ante la omisión del token. |

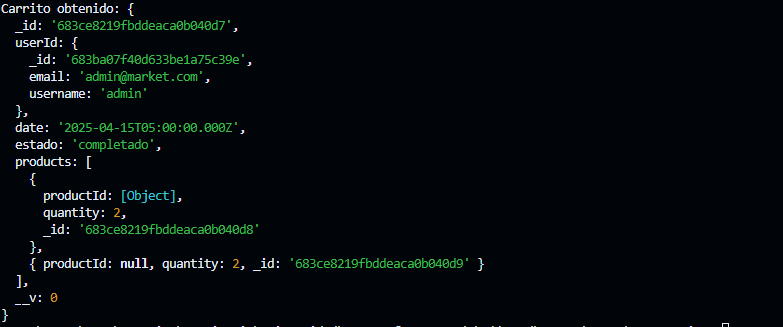
Texto

El contenido generado por IA puede ser incorrecto.

| ID | ST-05 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado pueda crear un nuevo carrito de compras mediante una petición POST a la ruta correspondiente. |
| Condiciones Previas | - El servidor debe estar en ejecución en http://localhost:5000.- Debe existir el usuario con ID 683ba07f40d633be1a75c39e en la base de datos.- Debe existir el producto con ID 683b9dd07a1837853e8e3209.- La ruta /api/carts debe estar implementada y protegida mediante autenticación JWT válida. |
| Pasos de la prueba | 1. Ejecutar el script testCreateCart.js usando Node.js.2. El script enviará una petición POST a /api/carts con un carrito que incluye un producto.3. Se incluye en la cabecera un token JWT válido del usuario admin.4. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un objeto JSON que representa el nuevo carrito creado, incluyendo los campos userId, products, date, estado, \_id. |
| Resultados ideales | json{  "userId": "683ba07f40d633be1a75c39e",  "products": [    { "productId": "683b9dd07a1837853e8e3209", "quantity": 2, "\_id": "<id>" }  ],  "date": "<fecha\_ISO>",  "estado": "pendiente",  "\_id": "<id>",  "\_\_v": 0} |
| Estado de la prueba | Exitosa |
| Nota adicional | La prueba confirma que los carritos pueden ser creados correctamente por usuarios autenticados y se asigna automáticamente el estado pendiente. |



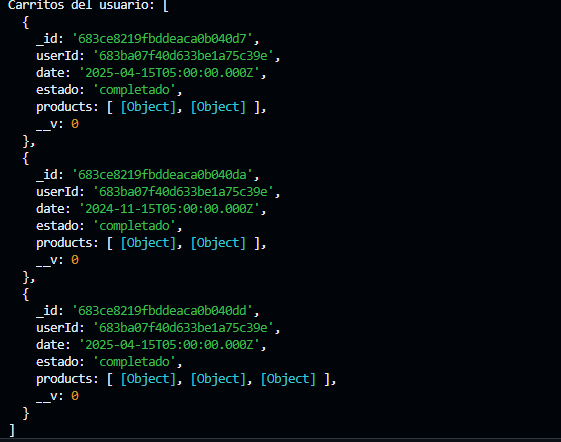
| ID | ST-06 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado (con rol adecuado) pueda eliminar un carrito existente mediante una petición DELETE a la API. |
| Condiciones Previas | - El servidor debe estar corriendo en http://localhost:5000.- Debe existir un carrito con ID 683d07560397a5a6ba225169 en la base de datos.- El usuario que realiza la petición debe tener un token JWT válido y permiso para eliminar el carrito.- La ruta /api/carts/:id debe estar correctamente implementada y protegida. |
| Pasos de la prueba | 1. Ejecutar el script testDeleteCart.js usando Node.js.2. El script enviará una petición DELETE a /api/carts/683d07560397a5a6ba225169.3. La petición incluye un token JWT válido en la cabecera.4. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un objeto JSON que confirme la eliminación del carrito. |
| Resultados ideales | json{  "message": "Carrito eliminado"} |
| Estado de la prueba | Exitosa |
| Nota adicional | La prueba confirma que la funcionalidad de eliminación de carritos está operando correctamente con control de autenticación y permisos. |



| ID | ST-07 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado pueda obtener un carrito específico mediante su ID utilizando una petición GET. |
| Condiciones Previas | - El servidor debe estar en ejecución en http://localhost:5000.- Debe existir un carrito con ID 683ce8219fbddeaca0b040d7 en la base de datos.- El usuario debe estar autenticado con un token JWT válido.- La ruta /api/carts/:id debe estar correctamente implementada y protegida. |
| Pasos de la prueba | 1. Ejecutar el script testGetCartById.js con Node.js.2. El script realiza una petición GET a /api/carts/683ce8219fbddeaca0b040d7 con el token en los headers.3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un objeto JSON que contiene los detalles del carrito correspondiente al ID. |
| Resultados ideales | json{  "\_id": "683ce8219fbddeaca0b040d7",  "userId": { "\_id": "683ba07f40d633be1a75c39e", "email": "[admin@market.com](mailto:admin@market.com)", "username": "admin" },  "date": "2025-04-15T05:00:00.000Z",  "estado": "completado",  "products": [ ... ]} |
| Estado de la prueba | Exitosa |
| Nota adicional | El campo productId de uno de los productos aparece como null, lo que podría ser validado como un caso aparte si se desea comprobar integridad de los datos. |



| ID | ST-08 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado pueda obtener todos los carritos mediante una petición GET a la ruta /api/carts. |
| Condiciones Previas | - El servidor debe estar en ejecución en [http://localhost:5000](http://localhost:5000/). - Deben existir carritos en la base de datos. - El usuario debe estar autenticado con un token JWT válido. - La ruta /api/carts debe estar correctamente implementada y protegida. |
| Pasos de la prueba | 1. Ejecutar el script testGetCarts.js con Node.js. 2. El script realiza una petición GET a /api/carts con el token en los headers. 3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un arreglo JSON que contiene los detalles de todos los carritos existentes. Cada carrito debe incluir su \_id, userId (con datos básicos del usuario), date, estado, y el arreglo products. |
| Resultados ideales | json { "\_id": "683ce8219fbddeaca0b04113", "userId": { "\_id": "683ba40e58cb411f6ebc674e", "email": "user8@example.com", "username": "user8" }, "date": "2024-10-15T05:00:00.000Z", "estado": "completado", "products": [ ... ], "\_\_v": 0 } |
| Estado de la prueba | Exitosa |
| Nota adicional | La respuesta muestra múltiples carritos con la estructura esperada, aunque la consola solo muestra objetos resumidos para products. |



| ID | ST-09 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado pueda obtener todos los carritos asociados a un usuario específico mediante una petición GET a la ruta /api/carts/user/:userId. |
| Condiciones Previas | - El servidor debe estar en ejecución en [http://localhost:5000](http://localhost:5000/). - Deben existir carritos en la base de datos asociados al usuario con ID 683ba07f40d633be1a75c39e. - El usuario debe estar autenticado con un token JWT válido. - La ruta /api/carts/user/:userId debe estar correctamente implementada y protegida. |
| Pasos de la prueba | 1. Ejecutar el script testGetCartsByUser.js con Node.js. 2. El script realiza una petición GET a /api/carts/user/683ba07f40d633be1a75c39e con el token en los headers. 3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un arreglo JSON que contiene los detalles de todos los carritos asociados al usuario indicado. Cada carrito incluye su \_id, userId, date, estado y el arreglo products. |
| Resultados ideales | json [ { "\_id": "683ce8219fbddeaca0b040d7", "userId": "683ba07f40d633be1a75c39e", "date": "2025-04-15T05:00:00.000Z", "estado": "completado", "products": [ ... ], "\_\_v": 0 }, ... ] |
| Estado de la prueba | Exitosa |
| Nota adicional | La respuesta muestra múltiples carritos asociados al usuario, con productos en forma de objetos resumidos. |

Texto

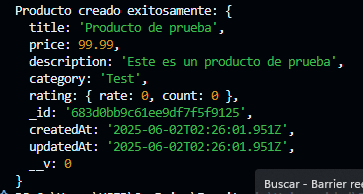
El contenido generado por IA puede ser incorrecto.

| ID | ST-10 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado pueda obtener estadísticas mensuales de ventas mediante una petición GET a la ruta /api/carts/stats/sales-by-month. |
| Condiciones Previas | - El servidor debe estar en ejecución en [http://localhost:5000](http://localhost:5000/). - Deben existir datos de ventas en la base de datos. - El usuario debe estar autenticado con un token JWT válido. - La ruta /api/carts/stats/sales-by-month debe estar correctamente implementada y protegida. |
| Pasos de la prueba | 1. Ejecutar el script testSalesByMonth.js con Node.js. 2. El script realiza una petición GET a /api/carts/stats/sales-by-month con el token en los headers. 3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con un arreglo JSON donde cada objeto representa un mes (\_id) y muestra totalOrders (cantidad total de pedidos) y totalProducts (cantidad total de productos vendidos) para ese mes. |
| Resultados ideales | json [ { "\_id": 1, "totalOrders": 17, "totalProducts": 44 }, { "\_id": 2, "totalOrders": 18, "totalProducts": 52 }, ... ] |
| Estado de la prueba | Exitosa |
| Nota adicional | La respuesta muestra estadísticas completas para los 12 meses del año, indicando que el cálculo de ventas por mes funciona correctamente. |

Texto

El contenido generado por IA puede ser incorrecto.

| ID | ST-11 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado pueda actualizar el contenido de un carrito específico mediante una petición PUT. |
| Condiciones Previas | - El servidor debe estar en ejecución en [http://localhost:5000](http://localhost:5000/). - Debe existir un carrito con ID 683ce8219fbddeaca0b040d7 en la base de datos. - El usuario debe estar autenticado con un token JWT válido. - La ruta /api/carts/:id debe aceptar peticiones PUT para actualización y estar protegida. |
| Pasos de la prueba | 1. Ejecutar el script testUpdateCart.js con Node.js. 2. El script realiza una petición PUT a /api/carts/683ce8219fbddeaca0b040d7 con el token en los headers y un nuevo arreglo de productos en el body. 3. Observar la respuesta impresa en consola. |
| Resultados esperados | El servidor responde con el objeto JSON actualizado del carrito, reflejando el nuevo contenido en products. |
| Resultados ideales | json { "\_id": "683ce8219fbddeaca0b040d7", "userId": "683ba07f40d633be1a75c39e", "date": "2025-04-15T05:00:00.000Z", "estado": "completado", "products": [ { "productId": "683b9dd07a1837853e8e3209", "quantity": 5, "\_id": "683d0b47c61ee9df7f5f9121" } ], "\_\_v": 0 } |
| Estado de la prueba | Exitosa |
| Nota adicional | Se confirma que la actualización reemplaza correctamente los productos del carrito con los nuevos datos enviados. |



| ID | ST-12 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado pueda crear un nuevo producto mediante una petición POST. |
| Condiciones Previas | - El servidor debe estar activo en [http://localhost:5000](http://localhost:5000/). - La ruta /api/products debe aceptar peticiones POST para creación de productos y estar protegida con autenticación JWT. - El token JWT debe ser válido y autorizado para crear productos. |
| Pasos de la prueba | 1. Ejecutar el script testCreateProduct.js con Node.js. 2. El script hace una petición POST a /api/products con los datos: título, precio, categoría y descripción. 3. Observar la respuesta JSON que contiene el nuevo producto creado. |
| Resultados esperados | El servidor responde con el objeto JSON del producto recién creado, incluyendo ID, timestamps y valores por defecto (como rating). |
| Resultados ideales | json { "title": "Producto de prueba", "price": 99.99, "description": "Este es un producto de prueba", "category": "Test", "rating": { "rate": 0, "count": 0 }, "\_id": "683d0bb9c61ee9df7f5f9125", "createdAt": "2025-06-02T02:26:01.951Z", "updatedAt": "2025-06-02T02:26:01.951Z", "\_\_v": 0 } |
| Estado de la prueba | Exitosa |
| Nota adicional | Se confirma que el producto se crea correctamente con los datos enviados y se generan valores por defecto para rating. |

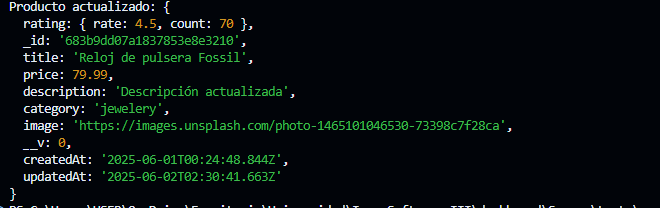


| ID | ST-13 |
| --- | --- |
| Descripción | Verificar que un usuario autenticado pueda eliminar un producto específico mediante una petición DELETE. |
| Condiciones Previas | - El servidor debe estar activo en [http://localhost:5000](http://localhost:5000/). - La ruta /api/products/:id debe aceptar peticiones DELETE y estar protegida con autenticación JWT. - El producto con ID 683b9dd07a1837853e8e320b debe existir en la base de datos antes de la prueba. - El token JWT debe ser válido y autorizado para eliminar productos. |
| Pasos de la prueba | 1. Ejecutar el script testDeleteProduct.js con Node.js. 2. El script realiza una petición DELETE a /api/products/683b9dd07a1837853e8e320b. 3. Observar la respuesta JSON del servidor confirmando la eliminación. |
| Resultados esperados | El servidor responde con un mensaje confirmando la eliminación del producto. |
| Resultados ideales | json { "message": "Producto eliminado" } |
| Estado de la prueba | Exitosa |
| Nota adicional | La prueba confirma que la eliminación funciona correctamente con el token proporcionado. |

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

| ID | ST-14 |
| --- | --- |
| Descripción | Verificar que el endpoint /api/products retorne la lista completa de productos. |
| Condiciones Previas | - El servidor debe estar corriendo en http://localhost:5000. - La base de datos debe contener productos con sus respectivos campos. |
| Pasos de la prueba | 1. Ejecutar la función testGetProducts. 2. Se realiza una petición GET a /api/products. 3. Verificar que la respuesta contenga un array con los productos. |
| Resultados esperados | Se recibe un arreglo con productos, donde cada producto incluye campos como \_id, title, price, category, rating, createdAt, updatedAt. |
| Ejemplo de respuesta | json [{ "\_id": "683cde2047d290a6b8d6bc6c", "title": "iphone 30", "price": 999, "category": "electronics", "rating": {"rate": 0, "count": 0}, "createdAt": "...", "updatedAt": "..." }, {...}] |
| Estado de la prueba | Exitosa |
| Notas adicionales | No se requiere token para este endpoint en esta prueba, según el código. |



| ID | ST-15 |
| --- | --- |
| Descripción | Verificar que se pueda actualizar uno o varios campos de un producto existente mediante su ID. |
| Condiciones Previas | - El servidor debe estar corriendo en http://localhost:5000. - El producto con ID 683b9dd07a1837853e8e3210 debe existir. - Token válido con permisos de administrador. |
| Pasos de la prueba | 1. Ejecutar la función testUpdateProduct. 2. Se hace una petición PUT a /api/products/683b9dd07a1837853e8e3210 con los campos a actualizar (ejemplo: price y description). 3. Verificar que la respuesta incluya los campos actualizados correctamente. |
| Resultados esperados | El producto se actualiza y retorna el objeto con los nuevos valores en los campos indicados. |
| Ejemplo de respuesta | json { "\_id": "683b9dd07a1837853e8e3210", "title": "Reloj de pulsera Fossil", "price": 79.99, "description": "Descripción actualizada", "category": "jewelery", "rating": { "rate": 4.5, "count": 70 }, "image": "...", "createdAt": "...", "updatedAt": "..." } |
| Estado de la prueba | Exitosa |
| Notas adicionales | Solo se envían los campos a modificar, el resto permanecen igual. |



| ID | ST-16 |
| --- | --- |
| Descripción | Verificar que se pueda eliminar un usuario existente mediante su ID. |
| Condiciones Previas | - El servidor debe estar ejecutándose en http://localhost:5000. - Debe existir un usuario con ID 683ba40e58cb411f6ebc674a. - El token JWT debe pertenecer a un usuario con rol de admin. |
| Pasos de la prueba | 1. Ejecutar la función testDeleteUser. 2. Se realiza una petición DELETE al endpoint /api/users/683ba40e58cb411f6ebc674a. 3. Se verifica la respuesta del servidor. |
| Resultado Esperado | Se devuelve un mensaje confirmando que el usuario fue eliminado: { message: 'Usuario eliminado' }. |
| Respuesta de ejemplo | json { "message": "Usuario eliminado" } |
| Estado de la prueba | Exitosa |
| Notas adicionales | Esta operación debe estar protegida para que solo usuarios administradores puedan ejecutarla. De lo contrario, debe retornarse un código 403 (Forbidden). |

Texto

El contenido generado por IA puede ser incorrecto.

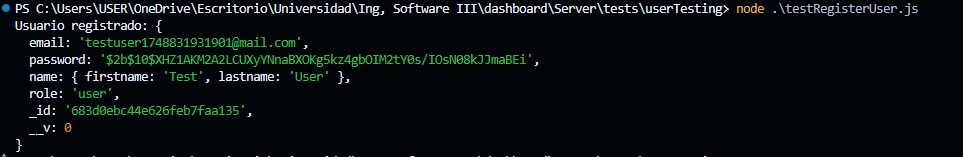
| ID | ST-17 |
| --- | --- |
| Descripción | Verificar que se pueda obtener correctamente la información de un usuario por su ID. |
| Condiciones Previas | - El servidor debe estar en ejecución en http://localhost:5000. - Debe existir un usuario con ID 683ba40e58cb411f6ebc674d. - Se debe proporcionar un token JWT válido de un usuario con permisos (por ejemplo, rol admin). |
| Pasos de la prueba | 1. Ejecutar la función testGetUserById. 2. Se realiza una petición GET a /api/users/683ba40e58cb411f6ebc674d. 3. Se verifica que el cuerpo de la respuesta contenga los datos del usuario esperado. |
| Resultado Esperado | Se retorna un objeto con los datos completos del usuario solicitado. |
| Respuesta de ejemplo | json { "name": { "firstname": "Alice", "lastname": "Thomas" }, "address": { "geolocation": { "lat": "-10.5212", "long": "69.2303" }, "city": "Phoenix", "street": "Broadway", "number": 1574, "zipcode": "94587" }, "\_id": "683ba40e58cb411f6ebc674d", "email": "user7@example.com", "username": "user7", "phone": "555-678-7200", "role": "user", "\_\_v": 0 } |
| Estado de la prueba | Exitosa |
| Notas adicionales | Esta operación debe estar protegida para que usuarios sin permisos no puedan acceder a información sensible de otros usuarios. |



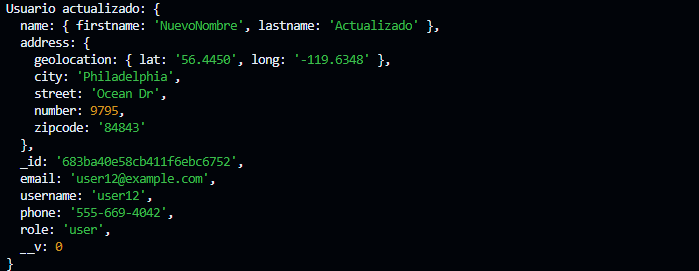
| ID | ST-18 |
| --- | --- |
| Descripción | Verificar que se pueda obtener correctamente la lista de todos los usuarios registrados en el sistema. |
| Condiciones Previas | - El servidor debe estar en ejecución en http://localhost:5000. - Deben existir usuarios registrados en la base de datos. - Se debe usar un token JWT válido con permisos de administrador (admin). |
| Pasos de la prueba | 1. Ejecutar la función testGetUsers. 2. Se realiza una petición GET a /api/users. 3. Se verifica que la respuesta sea una lista (array) de usuarios con sus datos correspondientes. |
| Resultado Esperado | Se retorna un array de objetos, cada uno representando un usuario con sus campos como name, address, email, username, phone, role, etc. |
| Respuesta de ejemplo | json [ { "name": { "firstname": "Olivia", "lastname": "Smith" }, "address": { "geolocation": { "lat": "-31.8129", "long": "62.5342" }, "city": "Chicago", "street": "Sunset Blvd", "number": 4783, "zipcode": "82195" }, "\_id": "683ba40e58cb411f6ebc67ac", "email": "user102@example.com", "username": "user102", "phone": "555-827-6124", "role": "user", "\_\_v": 0 }, ... ] |
| Estado de la prueba | Exitosa |
| Notas adicionales | Esta operación debe estar protegida y solo accesible para usuarios con el rol adecuado (admin). Se recomienda paginar o limitar la cantidad de usuarios retornados en bases de datos grandes. |



| ID | ST-19 |
| --- | --- |
| Descripción | Verificar que el sistema pueda calcular y retornar la cantidad de usuarios registrados agrupados por mes. |
| Condiciones Previas | - El servidor debe estar en ejecución en http://localhost:5000. - Se debe usar un token JWT válido con rol de administrador (admin). - Deben existir usuarios registrados en la base de datos con la propiedad createdAt. |
| Pasos de la prueba | 1. Ejecutar la función testRegisteredByMonth. 2. Se realiza una petición GET a /api/users/stats/registered-by-month. 3. Se verifica que la respuesta sea un arreglo con estadísticas de registros por mes. |
| Resultado Esperado | La respuesta debe ser un array de objetos con propiedades como \_id (mes o código de agrupación) y total (cantidad de registros). |
| Respuesta de ejemplo | json [ { "\_id": null, "total": 122 } ] |
| Estado de la prueba | Parcialmente exitosa |
| Notas adicionales | El valor de \_id: null indica que posiblemente falta la agrupación por mes en la consulta MongoDB (por ejemplo, usando $month sobre createdAt). Se recomienda revisar la lógica de agregación en el backend para corregir la agrupación y que \_id contenga el número del mes (1–12). |



| ID | ST-20 |
| --- | --- |
| Descripción | Verificar que el sistema permita registrar un nuevo usuario correctamente. |
| Condiciones Previas | - El servidor debe estar activo en http://localhost:5000. - No se requiere autenticación previa. |
| Pasos de la prueba | 1. Ejecutar la función testRegisterUser. 2. Se realiza una petición POST a /api/users con un cuerpo JSON que contiene name, email y password. 3. Se verifica que la respuesta incluya los datos del nuevo usuario registrado. |
| Resultado Esperado | La API debe retornar un objeto con los datos del usuario, incluyendo \_id, email, name, role y una contraseña encriptada. |
| Respuesta de ejemplo | json { "email": "testuser1748831931901@mail.com", "password": "$2b$10$...", "name": { "firstname": "Test", "lastname": "User" }, "role": "user", "\_id": "683d0ebc44e626feb7faa135", "\_\_v": 0 } |
| Estado de la prueba | Exitosa |
| Notas adicionales | - La contraseña se almacena encriptada con bcrypt, lo cual es correcto. - El uso de Date.now() asegura que el email no esté duplicado entre pruebas. - El rol predeterminado es "user" como se espera. |



| ID | ST-21 |
| --- | --- |
| Descripción | Verificar que el sistema permita actualizar correctamente los datos de un usuario específico. |
| Condiciones Previas | - El servidor debe estar corriendo en http://localhost:5000. - El usuario debe existir y el token JWT debe tener permisos de administrador. |
| Pasos de la prueba | 1. Ejecutar la función testUpdateUser. 2. Se realiza una petición PUT a /api/users/:id con un cuerpo JSON que contiene name. 3. Se verifica que la respuesta contenga los datos actualizados del usuario. |
| Resultado Esperado | La API debe retornar el objeto del usuario con los datos modificados, incluyendo los campos no actualizados (como email, address, etc.). |
| Respuesta de ejemplo | json { "name": { "firstname": "NuevoNombre", "lastname": "Actualizado" }, "address": { "geolocation": { "lat": "56.4450", "long": "-119.6348" }, "city": "Philadelphia", "street": "Ocean Dr", "number": 9795, "zipcode": "84843" }, "\_id": "683ba40e58cb411f6ebc6752", "email": "user12@example.com", "username": "user12", "phone": "555-669-4042", "role": "user", "\_\_v": 0 } |
| Estado de la prueba | Exitosa |
| Notas adicionales | - Solo se actualizó el campo name, pero la respuesta incluye todo el objeto del usuario. - Es importante que el backend implemente una validación para evitar cambios no autorizados en campos protegidos como role o email. |

| Campo | Detalle |
| --- | --- |
| ID del Caso | CT-01 |
| Nombre | Inicio de sesión exitoso |
| Descripción | Verificar que el usuario administrador pueda iniciar sesión correctamente. |
| Datos de Entrada | - Correo: admin@market.com- Contraseña: 123456789 |
| Resultado Esperado | El usuario se autentica correctamente y se genera un token de acceso. |
| Resultado Obtenido | El usuario administrador inició sesión correctamente. |
| Estado | Éxito |

| **Campo** | **Detalle** |
| --- | --- |
| **ID del Caso** | CT-02 |
| **Nombre** | Visualización de gráficos en el Dashboard |
| **Descripción** | Verificar que los gráficos del Dashboard se visualizan correctamente al obtener datos de los endpoints correspondientes. |
| **Datos de Entrada** | - Usuario autenticado con token válido- Peticiones a los endpoints:• GET /api/products• GET /api/users• GET /api/carts |
| **Resultado Esperado** | Se muestran correctamente:• Total de productos y su distribución por categoría• Total de usuarios registrados• Ventas totales e ingresos• Gráfico de ventas por categoría• Evolución de ingresos |
| **Resultado Obtenido** | Todos los gráficos y valores numéricos fueron renderizados correctamente, con datos representados visualmente de forma precisa. |
| **Estado** | Éxito |

| **Campo** | **Detalle** |
| --- | --- |
| **ID del Caso** | CT-03 |
| **Nombre** | Completar pedidos |
| **Descripción** | Verificar que un usuario pueda completar pedidos exitosamente desde la tabla de pedidos, confirmando la acción mediante un modal. |
| **Datos de Entrada** | Click en el botón “Completar” en 3 filas distintas de la tabla de pedidos, y confirmación en el modal emergente. |
| **Resultado Esperado** | El estado del pedido cambia exitosamente (por ejemplo, desaparece de la lista o se actualiza), indicando que se completó la acción. |
| **Resultado Obtenido** | El usuario hizo clic en “Completar” para 3 pedidos, confirmó en el modal, y los pedidos fueron marcados como completados correctamente. |
| **Estado** | Éxito |

| **Campo** | **Detalle** |
| --- | --- |
| **ID del Caso** | CT-04 |
| **Nombre** | Gestión de productos |
| **Descripción** | Verificar que el usuario pueda editar, eliminar y agregar productos desde la interfaz de administración. |
| **Datos de Entrada** | 1. Hacer clic en “Editar” y modificar un producto. 2. Hacer clic en “Eliminar” y confirmar la eliminación. 3. Hacer clic en “Agregar Producto” y llenar el formulario. |
| **Resultado Esperado** | El producto se actualiza correctamente. El producto seleccionado se elimina de la lista. Un nuevo producto aparece en la tabla. |
| **Resultado Obtenido** | Las tres acciones fueron realizadas con éxito: se editó un producto, se eliminó otro y se agregó uno nuevo correctamente. |
| **Estado** | Éxito |

| **Campo** | **Detalle** |
| --- | --- |
| **ID del Caso** | CT-05 |
| **Nombre** | Gestión de usuarios |
| **Descripción** | Validar que el sistema permita editar un usuario, eliminar un usuario y buscar usuarios mediante nombre o correo. |
| **Datos de Entrada** | 1. Hacer clic en “Editar” de un usuario y modificar los datos.2. Hacer clic en “Eliminar” y confirmar la acción.3. Usar la barra de búsqueda con texto “Emma”. |
| **Resultado Esperado** | El usuario se actualiza correctamente.El usuario seleccionado se elimina de la tabla.La búsqueda filtra y muestra correctamente a “Emma Harris”. |
| **Resultado Obtenido** | Las tres acciones se completaron correctamente: edición, eliminación y filtrado de usuarios. |
| **Estado** | Éxito |
|  |  |

| **Campo** | **Detalle** |
| --- | --- |
| **ID del Caso** | CT-06 |
| **Nombre** | Actualización del perfil del usuario |
| **Descripción** | Validar que el usuario pueda actualizar su información personal (nombre, apellido, teléfono, etc.) |
| **Datos de Entrada** | Se modifica el apellido y se deja vacío el campo “Teléfono”, luego se hace clic en “Guardar”. |
| **Resultado Esperado** | El perfil se actualiza correctamente y se muestra una notificación de éxito. |
| **Resultado Obtenido** | El sistema no actualizó el perfil ni mostró mensaje alguno; no se detectó validación. |
| **Estado** | Fallido |

| **Campo** | **Detalle** |
| --- | --- |
| **ID del Caso** | CT-07 |
| **Nombre** | Cambio de contraseña |
| **Descripción** | Verificar que el usuario pueda cambiar su contraseña desde la sección de perfil. |
| **Datos de Entrada** | Se ingresa una nueva contraseña y se hace clic en “Guardar” sin confirmación adicional. |
| **Resultado Esperado** | La contraseña se actualiza y se muestra una notificación de éxito. |
| **Resultado Obtenido** | El sistema no muestra respuesta alguna; no se sabe si la contraseña fue actualizada. |
| **Estado** | Fallido |