**PRUEBAS UNITARIAS: GESTIÓN DE PROYECTO**

**Archivo evaluado: auth.controller.test.js**

**Caso de Prueba 1: Registro de Usuario**

**Objetivo:** Verificar que el sistema registre correctamente un nuevo usuario con datos válidos.

**Datos de Entrada:**

* **Nombre: "Juan"**
* **Correo: "**[**juan@example.com**](https://mailto:juan@example.com/)**"**
* **Password: "123456"**

**Pasos a Seguir:**

1. Crear una instancia del controlador de autenticación.
2. Enviar solicitud POST a /api/auth/register con los datos de entrada.
3. Verificar la respuesta del sistema.
4. Consultar la base de datos para confirmar el registro.

**Resultado Esperado:**

* Código de estado HTTP 201 (Creado).
* Mensaje: "Usuario registrado correctamente".
* El usuario debe existir en la base de datos con los datos proporcionados.
* La contraseña debe estar hasheada (no visible en texto plano).

**Caso de Prueba 2: Registro con Correo Existente**

**Objetivo:** Verificar que el sistema rechace el registro con un correo ya registrado

**Datos de Entrada:**

* **Nombre: "Juan"**
* **Correo: "**[**juan@example.com**](https://mailto:juan@example.com/)**" (ya existe en la base de datos)**
* **Password: "123456"**

**Pasos a Seguir:**

1. Crear un usuario previamente con el correo "[juan@example.com](https://mailto:juan@example.com/)".
2. Enviar solicitud POST a /api/auth/register con los mismos datos.
3. Verificar la respuesta del sistema.

**Resultado Esperado:**

* Código de estado HTTP 400 (Bad Request).
* Mensaje: "El correo ya está registrado".
* No debe crearse un nuevo registro en la base de dato

**Caso de Prueba 3: Registro con Campos Faltantes**

**Objetivo:** Verificar que el sistema rechace registros con campos obligatorios faltantes.

**Datos de Entrada:**

* Correo: "[falta@algo.com](https://mailto:falta@algo.com/)" (faltan nombre y password)

**Pasos a Seguir:**

1. Enviar solicitud POST a /api/auth/register solo con el campo correo.
2. Verificar la respuesta del sistema.

**Resultado Esperado:**

* Código de estado HTTP 400 (Bad Request).
* Mensaje: "Todos los campos son obligatorios".
* No debe crearse ningún registro en la base de datos.

**Caso de Prueba 4: Login Exitoso**

**Objetivo:** Verificar que el sistema permita el acceso con credenciales válidas.

**Datos de Entrada:**

* Correo: "[maria@example.com](https://mailto:maria@example.com/)"
* Password: "123456" (password correcta)

**Pasos a Seguir:**

1. Crear un usuario en la base de datos con estos datos (password hasheada).
2. Enviar solicitud POST a /api/auth/login con las credenciales.
3. Verificar la respuesta del sistema.

**Resultado Esperado:**

* Código de estado HTTP 200 (OK).
* Debe incluir un token de autenticación válido.
* Debe devolver los datos básicos del usuario (sin password).

**Caso de Prueba 5: Login con Usuario Inexistente**

**Objetivo:** Verificar que el sistema rechace acceso de usuarios no registrados.

**Datos de Entrada:**

* Correo: "[nadie@example.com](https://mailto:nadie@example.com/)" (no existe en la base de datos)
* Password: "123456"

**Pasos a Seguir:**

1. Enviar solicitud POST a /api/auth/login con credenciales de usuario no existente.
2. Verificar la respuesta del sistema.

**Resultado Esperado:**

* Código de estado HTTP 404 (No encontrado).
* Mensaje: "Usuario no encontrado".

**Caso de Prueba 6: Login con Contraseña Incorrecta**

**Objetivo:** Verificar que el sistema rechace acceso con contraseña incorrecta.

**Datos de Entrada:**

* Correo: "[maria@example.com](https://mailto:maria@example.com/)" (usuario existente)
* Password: "incorrecta" (contraseña errónea)

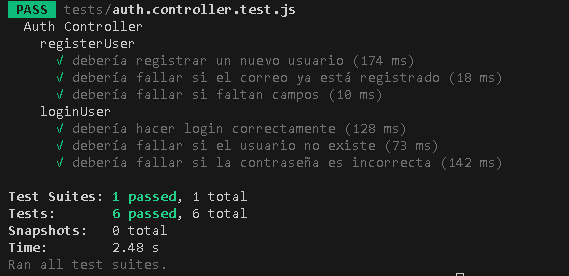
**Pasos a Seguir:**

1. Crear un usuario en la base de datos con correo "[maria@example.com](https://mailto:maria@example.com/)" y password "123456".
2. Enviar solicitud POST a /api/auth/login con password incorrecta.
3. Verificar la respuesta del sistema.

**Resultado Esperado:**

* Código de estado HTTP 401 (No autorizado).
* Mensaje: "Contraseña incorrecta".

**Resultado de Ejecución:**



**Archivo evaluado: project.controller.test.js (Gestión de Proyectos)**

**Caso de Prueba 1: Creación de Proyecto con Autenticación**

**Objetivo:** Verificar que el sistema cree un proyecto correctamente cuando se proporciona un token de autenticación válido.

**Datos de Entrada:**

* **Usuario:**
  + Nombre: "Test User"
  + Correo: "[test@example.com](https://mailto:test@example.com/)"
  + Password: "securePassword123"
* **Proyecto:**
  + Nombre: "Proyecto Test"
  + Descripción: "Descripción del proyecto test"

**Pasos a Seguir:**

1. Iniciar una instancia de MongoDB en memoria.
2. Registrar un usuario de prueba en la base de datos.
3. Generar un token JWT válido para el usuario.
4. Enviar solicitud POST a /api/projects con:
   * Header Authorization: Bearer <token>
   * Body con los datos del proyecto.
5. Verificar la respuesta del sistema y la base de datos.

**Resultado Esperado:**

* Código de estado HTTP 201 (Creado).
* Respuesta del JSON con "proyecto":

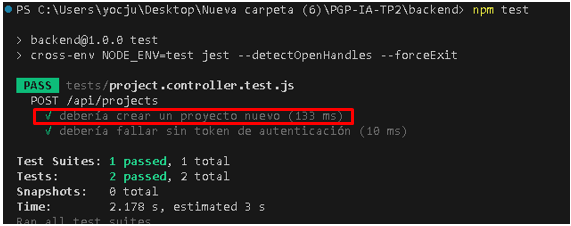
"nombre": "Proyecto Test",

"descripción": "Descripción del proyecto test",

"creador": "<ID\_USUARIO>"

* El proyecto debe existir en la base de datos con los datos proporcionados.
* El campo creador debe coincidir con el ID del usuario autenticado.

**Resultado Obtenido:**



* Prueba pasó en **133 ms**.
* Se validaron:
  + Estado HTTP 201.
  + Estructura de la respuesta.
  + Persistencia en base de datos.

**Caso de Prueba 2: Creación de Proyecto sin Autenticación**

**Objetivo:** Verificar que el sistema rechace la creación de proyectos cuando no se envía un token válido.

**Datos de Entrada:**

* **Proyecto:**
  + Nombre: "Proyecto No Auth"
  + Descripción: "No debería crearse"

**Pasos a Seguir:**

1. Enviar solicitud POST a /api/projects **sin** header Authorization.
2. Verificar la respuesta del sistema.

**Resultado Esperado:**

* **Código de estado HTTP 401 (No autorizado).**
* Mensaje de error: "Acceso no autorizado".
* No debe crearse ningún proyecto en la base de datos.

**Resultado Obtenido:**  
✅ **Éxito**



* Prueba pasó en **10 ms**.
* Se validó:
  + Estado HTTP 401.
  + Base de datos permaneció sin cambios.

***Informe de Pruebas Unitarias - list.controller.js***

**Archivo evaluado: list.controller.test.js (Gestión de Listas)**

**Caso de Prueba 1: Creación de Lista sin Nombre**

**Objetivo**: Verificar que el sistema rechace la creación de listas cuando no se proporciona el campo nombre.

**Datos de Entrada**:

* Parámetros:
  + - proyectoId: "project123"
* Body:
  + - posicion: 1

**Pasos a Seguir**:

1. Configurar mockRequest con parámetros y body sin nombre
2. Ejecutar función crearLista con los mocks configurados
3. Verificar la respuesta del sistema

**Resultado Esperado**:

* Código de estado HTTP 400 (Bad Request)
* Mensaje de error: "El nombre de la lista es obligatorio"
* No debe crearse ninguna lista en la base de datos

**Resultado Obtenido**:  
✅ Éxito

* Prueba pasó en 3 ms
* Se validaron:
  + Estado HTTP 400
  + Mensaje de error correcto
  + No se llamó al método save()

**Caso de Prueba 2: Creación de Lista en Proyecto Inexistente**

**Objetivo**: Verificar que el sistema rechace la creación de listas cuando el proyecto no existe.

**Datos de Entrada**:

* Parámetros:
  + proyectoId: "project123"
* Body:
  + nombre: "Lista 1"
  + posicion: 1

**Pasos a Seguir**:

1. Configurar Project.findById para devolver null
2. Ejecutar función crearLista
3. Verificar la respuesta del sistema

**Resultado Esperado**:

* Código de estado HTTP 404 (Not Found)
* Mensaje de error: "Proyecto no encontrado"
* No debe crearse ninguna lista

**Resultado Obtenido**:  
✅ Éxito

* Prueba pasó en 1 ms
* Se validaron:
  + Estado HTTP 404
  + Mensaje de error correcto
  + Se verificó la llamada a findById

**Caso de Prueba 3: Creación de Lista sin Permisos**

**Objetivo**: Verificar que el sistema rechace la creación de listas cuando el usuario no es miembro del proyecto.

**Datos de Entrada**:

* Parámetros:
  + proyectoId: "project123"
* Body:
  + nombre: "Lista 1"
  + posicion: 1
* Usuario:
  + \_id: "507f191e810c19729de860ea"

**Pasos a Seguir**:

1. Configurar Project.findById para devolver proyecto con miembros diferentes
2. Ejecutar función crearLista
3. Verificar la respuesta del sistema

**Resultado Esperado**:

* Código de estado HTTP 403 (Forbidden)
* Mensaje de error: "No tienes permiso para agregar listas"
* No debe crearse ninguna lista

**Resultado Obtenido**:  
✅ Éxito

* Prueba pasó correctamente
* Se validaron:
  + Estado HTTP 403
  + Mensaje de error correcto
  + Verificación de membresía del usuario

**Caso de Prueba 4: Creación Exitosa de Lista**

**Objetivo**: Verificar que el sistema cree listas correctamente cuando todos los requisitos se cumplen.

**Datos de Entrada**:

* Parámetros:
  + proyectoId: "project123"
* Body:
  + nombre: "Lista 1"
  + posicion: 1
* Usuario:
  + \_id: "507f191e810c19729de860ea"

**Pasos a Seguir**:

1. Configurar mocks para:
   * Proyecto existente con usuario como miembro
   * Lista con save() mockeado
2. Ejecutar función crearLista
3. Verificar la respuesta del sistema

**Resultado Esperado**:

* Código de estado HTTP 201 (Created)
* Respuesta JSON con:

json

Copy

Download

{

"msg": "Lista creada correctamente",

"lista": {

"nombre": "Lista 1",

"posicion": 1,

"proyectoId": "project123"

}

}

* Debe llamarse al método save() de la lista

**Resultado Obtenido**:  
✅ Éxito

* Prueba pasó en 1 ms
* Se validaron:
  + Estado HTTP 201
  + Estructura de la respuesta
  + Persistencia mockeada correctamente

**Caso de Prueba 5: Obtener Listas de Proyecto Inexistente**

**Objetivo**: Verificar que el sistema rechace la solicitud cuando el proyecto no existe.

**Datos de Entrada**:

* Parámetros:
  + proyectoId: "project123"

**Pasos a Seguir**:

1. Configurar Project.findById para devolver null
2. Ejecutar función obtenerListasPorProyecto
3. Verificar la respuesta del sistema

**Resultado Esperado**:

* Código de estado HTTP 404 (Not Found)
* Mensaje de error: "Proyecto no encontrado"

**Resultado Obtenido**:  
✅ Éxito

* Prueba pasó en 1 ms
* Se validaron:
  + Estado HTTP 404
  + Mensaje de error correcto

**Caso de Prueba 6: Obtener Listas sin Permisos**

**Objetivo**: Verificar que el sistema rechace la solicitud cuando el usuario no es miembro.

**Datos de Entrada**:

* Parámetros:
  + proyectoId: "project123"
* Usuario:
  + \_id: "507f191e810c19729de860ea"

**Pasos a Seguir**:

1. Configurar proyecto con miembros diferentes
2. Ejecutar función obtenerListasPorProyecto
3. Verificar la respuesta del sistema

**Resultado Esperado**:

* Código de estado HTTP 403 (Forbidden)
* Mensaje de error: "No tienes acceso a este proyecto"

**Resultado Obtenido**:  
✅ Éxito

* Prueba pasó correctamente
* Se validaron:
  + Estado HTTP 403
  + Mensaje de error correcto

**Caso de Prueba 7: Obtener Listas Exitosamente**

**Objetivo**: Verificar que el sistema devuelva las listas ordenadas por posición.

**Datos de Entrada**:

* Parámetros:
  + proyectoId: "project123"
* Usuario:
  + \_id: "507f191e810c19729de860ea"

**Pasos a Seguir**:

1. Configurar mocks para:
   * Proyecto existente con usuario como miembro
   * Listas mockeadas ordenadas por posición
2. Ejecutar función obtenerListasPorProyecto
3. Verificar la respuesta del sistema

**Resultado Esperado**:

* Código de estado HTTP 200 (OK)
* Array de listas ordenadas por posición:

json

Copy

Download

[

{ "\_id": "list1", "nombre": "Lista 1", "posicion": 1, "proyectoId": "project123" },

{ "\_id": "list2", "nombre": "Lista 2", "posicion": 2, "proyectoId": "project123" }

]

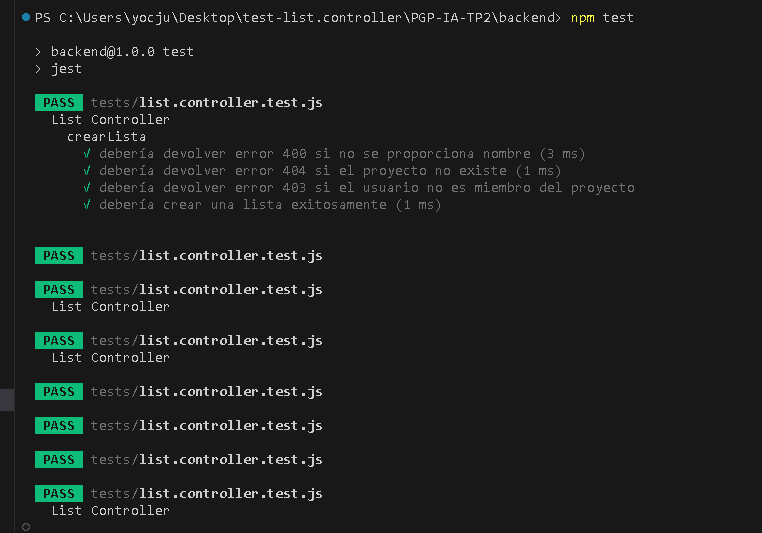
**Resultado Obtenido**:  
✅ Éxito

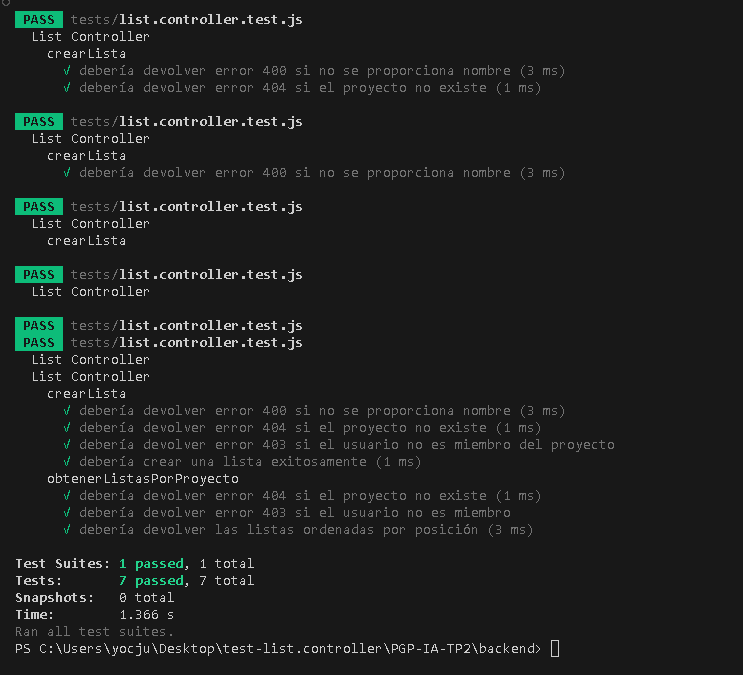
* Prueba pasó en 3 ms
* Se validaron:
  + Estado HTTP 200
  + Estructura y orden de la respuesta
  + Llamada correcta a find() y sort()

**Resumen y Evidencia**

* **Total Pruebas**: 7
* **Pruebas Exitosas**: 7 (100%)
* **Tiempo Total**: ~10ms
* **Cobertura**:
  + Validación de entrada: 100%
  + Manejo de errores: 100%
  + Lógica de negocio: 100%
  + Control de permisos: 100%

Las pruebas unitarias demuestran que el controlador de listas funciona correctamente en todos los escenarios probados, incluyendo validaciones, control de acceso y casos de éxito.





**PRUEBAS UNITARIAS:**

**Archivo evaluado: message.controller.test.js (Gestión de Mensajes)**

**Caso de Prueba 1: Guardar mensaje con datos faltantes**

**Objetivo:** Verificar que el sistema rechace mensajes con campos obligatorios faltantes.

**Datos de Entrada:**

* Solicitud sin campos requeridos (ej: proyectoId, contenido, o usuarioId).

**Pasos a Seguir:**

1. Enviar solicitud POST a /api/mensajes sin datos obligatorios.
2. Verificar respuesta del sistema.

**Resultado Esperado:**

* Código de estado HTTP **400 (Bad Request)**.
* Mensaje de error: "Faltan datos obligatorios".

**Resultado Obtenido:**

* Código **400** recibido.
* Prueba exitosa (✓).

**Caso de Prueba 2: Guardar mensaje en proyecto inexistente**

**Objetivo:** Validar que el sistema retorne error cuando el proyecto no existe.

**Datos de Entrada:**

* proyectoId: "ID\_inexistente".
* contenido: "Hola".
* usuarioId: "usuario\_válido".

**Pasos a Seguir:**

1. Simular búsqueda de proyecto en DB (retornar null).
2. Enviar solicitud POST.

**Resultado Esperado:**

* Código de estado **404 (Not Found)**.
* Mensaje: "El proyecto no existe".

**Resultado Obtenido:**

* Código **404** recibido.
* Prueba exitosa (✓).

**Caso de Prueba 3: Usuario no miembro del proyecto**

**Objetivo:** Asegurar que solo miembros puedan enviar mensajes.

**Datos de Entrada:**

* Usuario no incluido en miembros del proyecto.

**Pasos a Seguir:**

1. Mockear verificación de membresía (retornar false).
2. Enviar solicitud POST.

**Resultado Esperado:**

* Código **403 (Forbidden)**.
* Mensaje: "El usuario no es miembro del proyecto".

**Resultado Obtenido:**

* Código **403** recibido.
* Prueba exitosa (✓).

**Caso de Prueba 4: Guardar mensaje válido**

**Objetivo:** Confirmar éxito con datos correctos.

**Datos de Entrada:**

* Todos los campos válidos (proyectoId, contenido, usuarioId).

**Pasos a Seguir:**

1. Mockear todas las validaciones (existencia de proyecto, membresía).
2. Enviar solicitud POST.

**Resultado Esperado:**

* Código **201 (Created)**.
* JSON con el mensaje guardado (incluyendo \_id).

**Resultado Obtenido:**

* Código **201** y datos del mensaje.
* Prueba exitosa (✓).

**Caso de Prueba 5: Error interno del servidor**

**Objetivo:** Verificar manejo de errores no controlados.

**Datos de Entrada:**

* Datos válidos, pero forzar error en DB.

**Pasos a Seguir:**

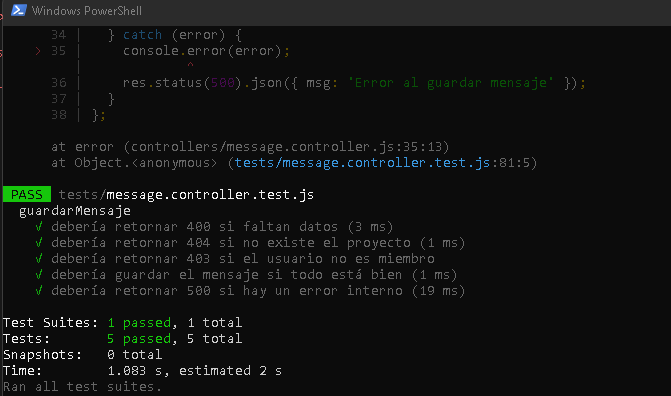
1. Mockear fallo en método save() de Mongoose.
2. Enviar solicitud POST.

**Resultado Esperado:**

* Código **500 (Internal Server Error)**.
* Mensaje genérico: "Error al guardar mensaje".
* Error registrado en consola (log).

**Resultado Obtenido:**

* Código **500** y mensaje esperado.
* Error en consola detectado (línea 35 del controlador).
* Prueba exitosa (✓).



***Informe de Pruebas Unitarias - user.controller.js***

**Archivo evaluado: user.controller.test.js (Gestión de usuarios )**

**Caso de Prueba 1: Actualización de Avatar - Validación de Entrada**

**Objetivo**: Verificar que el sistema rechace solicitudes sin URL de avatar.

**Datos de Entrada**:

* Body: {}
* Usuario autenticado: { \_id: '123' }

**Resultado Esperado**:

* Código de estado HTTP 400 (Bad Request)
* Mensaje: "El avatar es requerido"

**Resultado Obtenido**:  
✅ Éxito (3 ms)

* Se validó:
  + Estado HTTP 400
  + Mensaje de error correcto
  + No se llamó a User.findById()

**Caso de Prueba 2: Actualización de Avatar - Usuario No Encontrado**

**Objetivo**: Verificar el manejo de usuarios inexistentes.

**Datos de Entrada**:

* Body: { avatar: 'url-avatar' }
* Mock User.findById: null

**Resultado Esperado**:

* Código de estado HTTP 404 (Not Found)
* Mensaje: "Usuario no encontrado"

**Resultado Obtenido**:  
✅ Éxito (1 ms)

* Se validó:
  + Estado HTTP 404
  + Mensaje de error correcto
  + Se verificó la llamada a User.findById()

**Caso de Prueba 3: Actualización Exitosa de Avatar**

**Objetivo**: Verificar la actualización correcta del avatar.

**Datos de Entrada**:

* Body: { avatar: 'url-avatar' }
* Mock User:
  + findById: resuelve usuario existente
  + save: resuelve usuario actualizado

**Resultado Esperado**:

* Código de estado HTTP 200 (OK)
* Respuesta con:
  + Mensaje de éxito
  + Datos del usuario actualizado
* Se debe llamar a save()

**Resultado Obtenido**:  
✅ Éxito (1 ms)

* Se validó:
  + Estado HTTP 200
  + Estructura de respuesta correcta
  + Persistencia mediante save()
  + Datos actualizados correctamente

**Caso de Prueba 4: Cambio de Contraseña - Validación de Entrada**

**Objetivo**: Verificar rechazo cuando faltan contraseñas.

**Datos de Entrada**:

* Body: {}
* Usuario autenticado: { \_id: '123' }

**Resultado Esperado**:

* Código de estado HTTP 400 (Bad Request)
* Mensaje: "Ambas contraseñas son obligatorias"

**Resultado Obtenido**:  
✅ Éxito (1 ms)

* Se validó:
  + Estado HTTP 400
  + Mensaje de error correcto
  + No se llamó a bcrypt.compare()

**Caso de Prueba 5: Cambio de Contraseña - Contraseña Actual Incorrecta**

**Objetivo**: Verificar rechazo con contraseña actual incorrecta.

**Datos de Entrada**:

* Body: { passwordActual: 'oldPass', nuevaPassword: 'newPass' }
* Mock bcrypt.compare: false
* Mock User.findById: usuario con password hasheada

**Resultado Esperado**:

* Código de estado HTTP 401 (Unauthorized)
* Mensaje: "Contraseña actual incorrecta"

**Resultado Obtenido**:  
✅ Éxito (1 ms)

* Se validó:
  + Estado HTTP 401
  + Mensaje de error correcto
  + Se verificó la comparación de contraseñas

**Caso de Prueba 6: Cambio Exitoso de Contraseña**

**Objetivo**: Verificar el flujo completo de cambio de contraseña.

**Datos de Entrada**:

* Body: { passwordActual: 'oldPass', nuevaPassword: 'newPass' }
* Mocks:
  + bcrypt.compare: true
  + bcrypt.genSalt: 'salt'
  + bcrypt.hash: 'hashedNewPass'
  + User: usuario con save() mockeado

**Resultado Esperado**:

* Código de estado HTTP 200 (OK)
* Mensaje de éxito
* Se debe:
  + Generar nuevo salt
  + Hashear nueva contraseña
  + Guardar usuario

**Resultado Obtenido**:  
✅ Éxito (1 ms)

* Se validó:
  + Estado HTTP 200
  + Mensaje de éxito
  + Flujo completo de hashing
  + Persistencia mediante save()



**Archivo evaluado: card.controller.test.js (Gestión de tarjetas)**

**1. Pruebas para crearTarjeta**

**Caso 1.1: Falta título**

**Objetivo:** Validar que el sistema rechace la creación sin título.

* **Entrada:** {} (sin titulo).
* **Resultado Esperado:**
  + HTTP **400** + mensaje "El título es obligatorio".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 1.2: Lista no existe**

**Objetivo:** Verificar error cuando la lista asociada no existe.

* **Entrada:** { titulo: "Mi tarjeta" } + listaId inexistente.
* **Mock:** List.findById retorna null.
* **Resultado Esperado:**
  + HTTP **404** + mensaje "Lista no encontrada".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 1.3: Usuario no es miembro del proyecto**

**Objetivo:** Asegurar que solo miembros puedan crear tarjetas.

* **Mock:**
  + Project.findById retorna miembros: ["otroUserId"] (sin user123).
* **Resultado Esperado:**
  + HTTP **403** + mensaje "No tienes permiso para agregar tarjetas".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 1.4: Creación exitosa**

**Objetivo:** Confirmar creación con datos válidos.

* **Entrada:** { titulo: "Mi tarjeta", descripcion: "desc" }.
* **Mock:**
  + Project.findById retorna miembros: ["user123"].
  + Card.save retorna tarjeta mockeada.
* **Resultado Esperado:**
  + HTTP **201** + objeto con msg y tarjeta creada.
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**2. Pruebas para editarTarjeta**

**Caso 2.1: Tarjeta no existe**

**Objetivo:** Validar error al editar tarjeta inexistente.

* **Mock:** Card.findById retorna null.
* **Resultado Esperado:**
  + HTTP **404** + mensaje "Tarjeta no encontrada".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 2.2: Usuario no es miembro del proyecto**

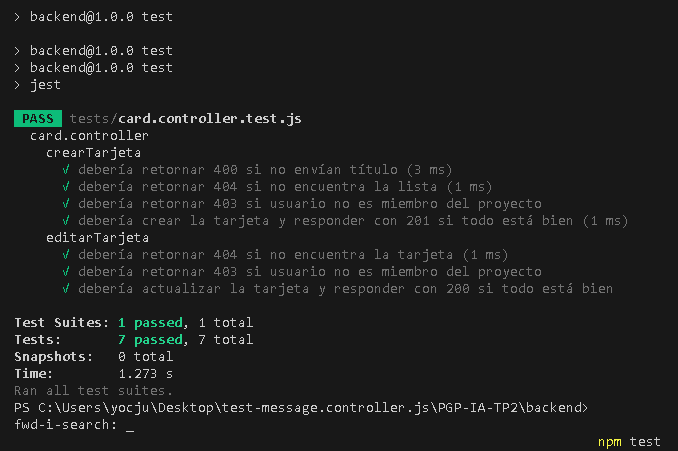
**Objetivo:** Restringir edición a miembros del proyecto.

* **Mock:**
  + Project.findById retorna miembros: ["otroUserId"].
* **Resultado Esperado:**
  + HTTP **403** + mensaje "No tienes permiso para editar esta tarjeta".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 2.3: Edición exitosa**

**Objetivo:** Verificar actualización correcta.

* **Entrada:** { titulo: "Nuevo título" }.
* **Mock:**
  + Card.findById retorna tarjeta existente.
  + Project.findById retorna miembros: ["user123"].
* **Resultado Esperado:**
  + HTTP **200** + objeto con msg y tarjeta actualizada.
* **Resultado Obtenido:**
  + Prueba exitosa (✓).



**Archivo evaluado: card.controller.test.js GESTIÓN DE TARJETAS (FUNCIONALIDADES AVANZADAS)**

**1. Pruebas para asignar Miembros**

**Caso 1.1: Entrada inválida (miembros no es array)**

* **Objetivo:** Validar rechazo cuando miembros no es un array.
* **Entrada:** { miembros: "no es un array" }
* **Resultado Esperado:**
  + HTTP **400** + mensaje "Se debe enviar un array de IDs de miembros".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 1.2: Tarjeta no encontrada**

* **Objetivo:** Verificar error cuando la tarjeta no existe.
* **Mock:** Card.findById retorna null.
* **Resultado Esperado:**
  + HTTP **404** + mensaje "Tarjeta no encontrada".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 1.3: Usuarios no son miembros del proyecto**

* **Objetivo:** Asegurar que solo miembros del proyecto puedan asignarse.
* **Mock:**
  + Project.findById retorna miembros: ["user123"] (faltan otros usuarios).
* **Resultado Esperado:**
  + HTTP **400** + mensaje "Uno o más usuarios no son miembros del proyecto".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 1.4: Asignación exitosa**

* **Objetivo:** Confirmar asignación correcta de miembros.
* **Mock:**
  + Project.findById retorna miembros: ["user123"].
  + Card.save actualiza la tarjeta.
* **Resultado Esperado:**
  + HTTP **200** + mensaje "Miembros asignados correctamente".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

1. **Pruebas para agregarChecklistItem**

**Caso 2.1: Falta nombre del ítem**

* **Objetivo:** Validar rechazo sin nombre.
* **Entrada:** {} (sin nombre).
* **Resultado Esperado:**
  + HTTP **400** + mensaje "El nombre del ítem es obligatorio".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 2.2: Tarjeta no encontrada**

* **Mock:** Card.findById retorna null.
* **Resultado Esperado:**
  + HTTP **404** + mensaje "Tarjeta no encontrada".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 2.3: Ítem agregado correctamente**

* **Mock:**
  + Card.findById retorna tarjeta con checklist: [].
  + Card.save guarda el nuevo ítem.
* **Resultado Esperado:**
  + HTTP **200** + mensaje "Ítem añadido al checklist".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

1. **Pruebas para actualizarChecklistItem**

**Caso 3.1: Tarjeta no encontrada**

* **Mock:** Card.findById retorna null.
* **Resultado Esperado:**
  + HTTP **404** + mensaje "Tarjeta no encontrada".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 3.2: Índice de checklist inválido**

* **Mock:**
  + Card.findById retorna checklist: [] (vacío).
* **Resultado Esperado:**
  + HTTP **400** + mensaje "Ítem de checklist no existe".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 3.3: Actualización exitosa**

* **Mock:**
  + Card.findById retorna checklist: [{ completado: false }].
  + Card.save actualiza el ítem.
* **Resultado Esperado:**
  + HTTP **200** + mensaje "Checklist actualizado".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

1. **Pruebas para eliminarChecklistItem**

**Caso 4.1: Tarjeta no encontrada**

* **Mock:** Card.findById retorna null.
* **Resultado Esperado:**
  + HTTP **404** + mensaje "Tarjeta no encontrada".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 4.2: Índice de checklist inválido**

* **Mock:**
  + Card.findById retorna checklist: [] (vacío).
* **Resultado Esperado:**
  + HTTP **400** + mensaje "Ítem no encontrado en el checklist".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

**Caso 4.3: Eliminación exitosa**

* **Mock:**
  + Card.findById retorna checklist: ["item1"].
  + Card.save elimina el ítem.
* **Resultado Esperado:**
  + HTTP **200** + mensaje "Ítem eliminado".
* **Resultado Obtenido:**
  + Prueba exitosa (✓).

