# UNIVERSIDAD SANTO TOMAS DE AQUINO

# CALIDAD DE SOFTWARE

# PRUEBAS DE TESTING AUTOMATIZADAS Y UNITARIAS

# NOMBRES:

# XIMENA PATRICIA GARCIA ESPINEL

# CRISTIAN SANTIAGO BARRERA TORRES

# SEBASTIAN ALEJANDRO RINCON BURGOS

# ANDRES FELIPE WILCHES SALAS

# TUNJA 2025

# 

# 1 Tests para crearUsuarioAdmin

## 1. Objetivo del Test

Verificar que la función `crearUsuarioAdmin` crea un usuario administrador si este no existe en la base de datos y maneja correctamente los casos de usuario existente o errores.

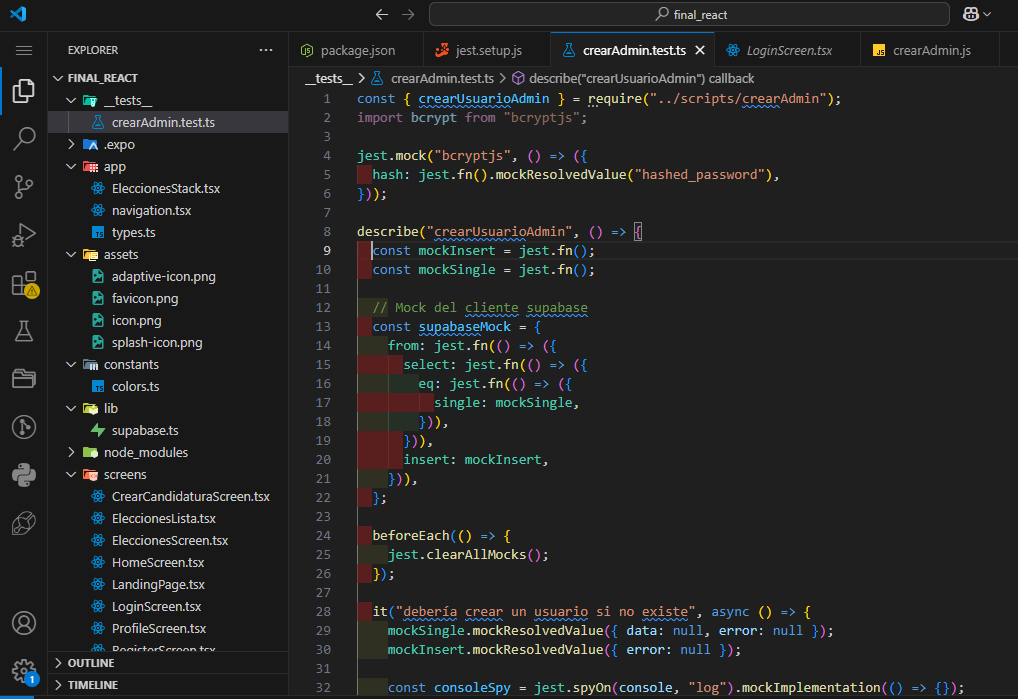
## 2. Descripción del Test

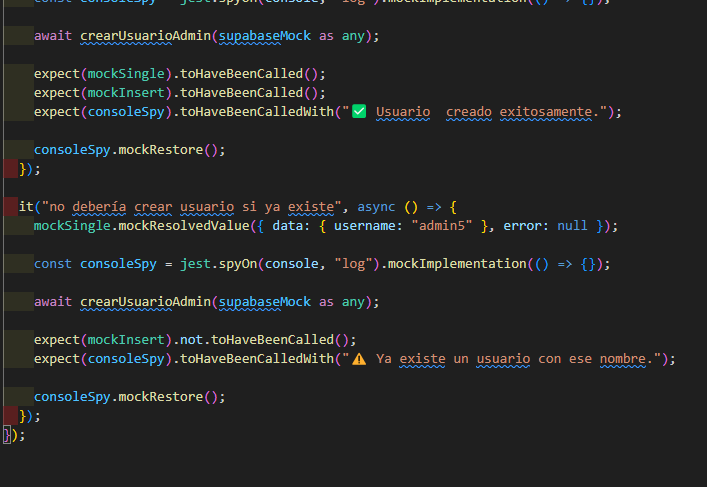
El test ejecuta la función `crearUsuarioAdmin` simulando la base de datos con mocks para:  
- Verificar que se inserta un nuevo usuario cuando no existe.  
- Confirmar que no se inserta un usuario si ya existe.  
- Comprobar que los mensajes de consola sean los correctos.

## 3. Herramientas utilizadas

- \*\*Jest\*\*: Framework para pruebas unitarias en JavaScript/TypeScript.  
- \*\*Jest Mocks\*\*: Para simular llamadas a Supabase y funciones de bcrypt.  
- \*\*Fake Timers\*\* (opcional en futuros tests): Para controlar temporizadores y simulaciones asíncronas.

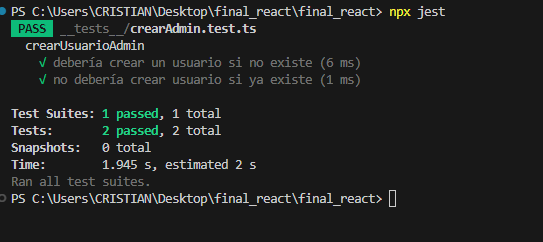
## 4. Código del Test





## 5. Resultados esperados

- El usuario es insertado en la tabla `users` solo si no existe previamente.  
- La función imprime en consola el mensaje correcto.  
- En caso de usuario existente, no se realiza la inserción y se muestra una advertencia.



## 6. Estado actual

- Test aprobado exitosamente en entorno local con Jest.

# 2 Tests para Login Usuario

## 1. Objetivo del Test

Validar la función loginUsuario para asegurar que maneja correctamente el inicio de sesión de usuarios, verificando la autenticidad del nombre de usuario y la contraseña.

## 2. Descripción del Test

Se realizaron tres pruebas principales:

* **Validación correcta**: Verificar que un usuario con nombre y contraseña válidos pueda iniciar sesión exitosamente.
* **Usuario no existe**: Verificar que la función retorne un mensaje adecuado cuando el usuario no se encuentra en la base de datos.
* **Contraseña incorrecta**: Comprobar que la función rechace el acceso cuando la contraseña es incorrecta.

Para lograrlo, se simularon (mockearon) las dependencias externas:

* Supabase (simulación de la consulta a la base de datos).
* bcrypt (simulación de la comparación de contraseñas).
* AsyncStorage (simulación del almacenamiento local).

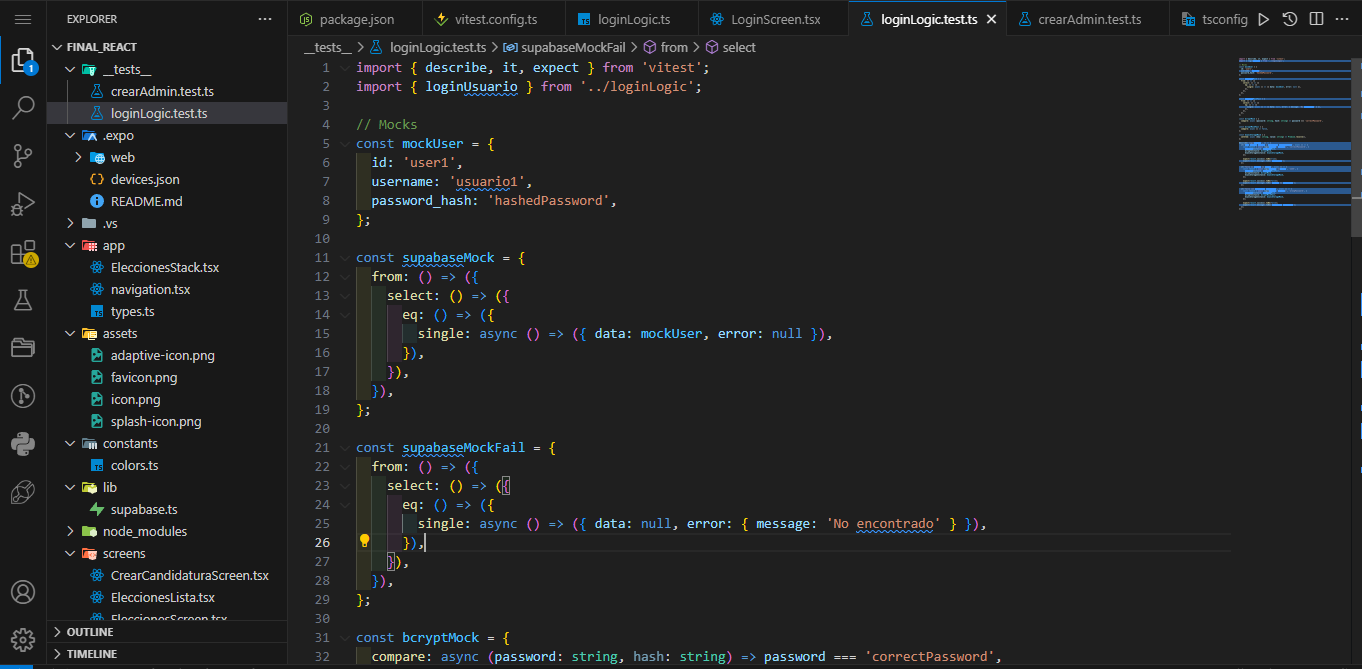
## 3. Herramientas utilizadas

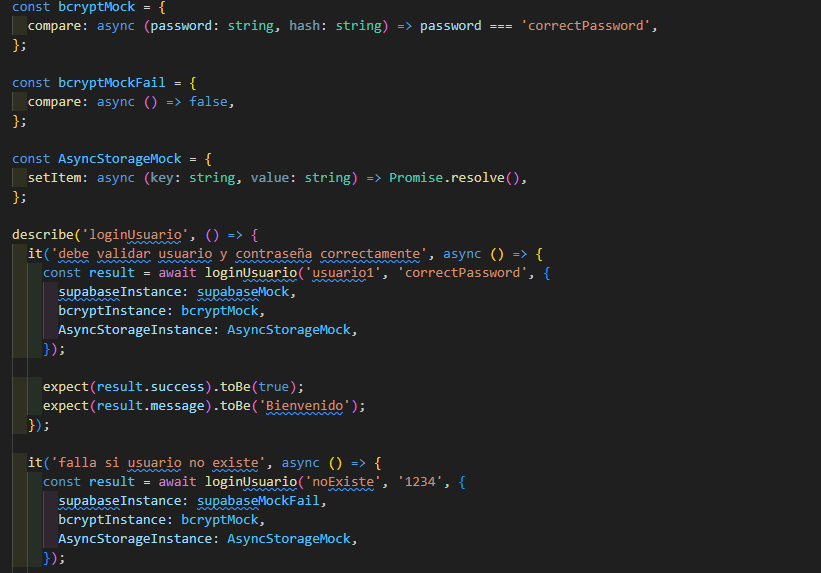
-  **Vitest**: Framework de testing para ejecutar y organizar las pruebas.

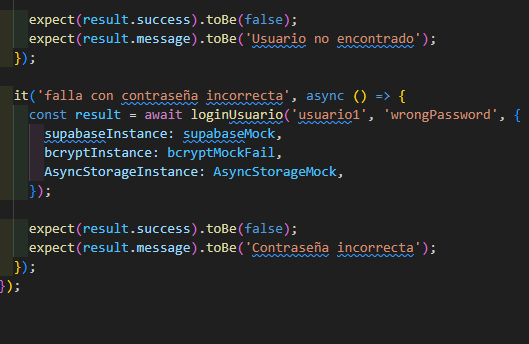
 **Mocking manual** para dependencias como Supabase, bcrypt y AsyncStorage.

 **TypeScript** (por los tipos en los mocks y la función).

## 4. Código del Test







## 5. Resultados esperados

 Cuando el usuario y contraseña son correctos, se espera un objeto con { success: true, message: 'Bienvenido' }.

 Cuando el usuario no existe, se espera { success: false, message: 'Usuario no encontrado' }.

 Cuando la contraseña es incorrecta, se espera { success: false, message: 'Contraseña incorrecta' }.

## 6. Estado actual

Las tres pruebas pasaron correctamente al ejecutar npx vitest run \_\_tests\_\_/loginLogic.test.ts, indicando que la función loginUsuario está comportándose como se espera bajo los escenarios probados.

# 3 Tests para calidad de codigo

# 1. Objetivo del Test

Validar la calidad del código y detectar errores comunes, variables no usadas, definiciones ausentes y malas prácticas en el proyecto React Native (Expo) usando ESLint, así como asegurar que las pruebas unitarias para funciones clave se implementan correctamente.

## 2. Descripción del Test

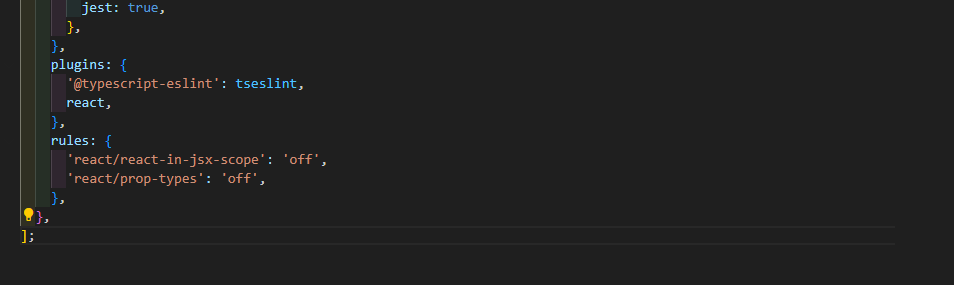
Se ejecutó ESLint en todo el proyecto para revisar el código fuente con extensiones .js, .jsx, .ts, .tsx. También se probaron las pruebas unitarias implementadas con Jest y AVA para las funciones críticas, como crearUsuarioAdmin y loginLogic. Se aplicaron correcciones automáticas disponibles mediante la opción --fix de ESLint y se revisaron los errores restantes que requieren atención manual.

## 3. Herramientas utilizadas

- **ESLint**: Para análisis estático del código y detección de errores de sintaxis, variables no usadas, definiciones ausentes, etc.

## 4. Código del Test





## 5. Resultados esperados

 Código limpio, sin errores de variables no definidas o no usadas.

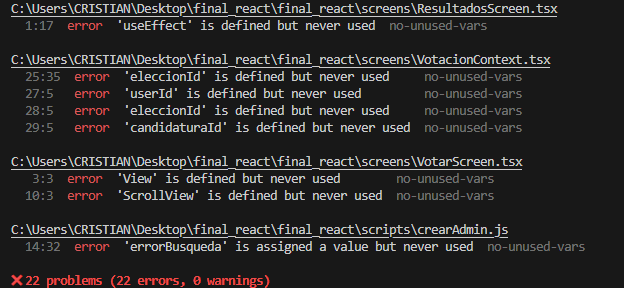
 Pruebas unitarias ejecutadas con éxito

 Código con mejores prácticas y sin warnings

 Funciones críticas, como el login y la creación de usuarios admin, correctamente testeadas y validadas.

## 

En este primero Podemos ver que hay 69 errores



Aca ya solo quedan 22 errores , pero estos son de sentencias que no se pueden borrar ya que podria afectar el funcionamiento original del proyecto

## 6. Estado actual

* ESLint reporta **22 errores** restantes que principalmente corresponden a variables definidas pero no usadas (no-unused-vars) y algunas importaciones sin uso.
* Se corrigieron automáticamente varios errores, pero algunos deben ser revisados y limpiados manualmente.
* Pruebas unitarias están implementadas pero algunas importaciones y variables dentro de ellas no se utilizan aún.
* No hay errores críticos de sintaxis o definiciones ausentes luego de aplicar --fix.
* Se recomienda eliminar variables e importaciones no usadas para mantener el código limpio y evitar advertencias.

# 4 Tests para Registro de usuarios

# 1. Objetivo del Test

Verificar que el formulario de registro de usuario funcione correctamente cuando se ingresan datos válidos, insertando correctamente el nuevo usuario en la base de datos (Supabase) y mostrando el mensaje de confirmación.

## 2. Descripción del Test

El test automatizado simula el registro de un nuevo usuario en la pantalla de registro. Llena los campos requeridos (Identificación, Nombre de usuario, Contraseña y Confirmar contraseña), envía el formulario y comprueba que aparece el mensaje: **"Usuario registrado correctamente"**. También se verifica que el backend (Supabase) devuelve una respuesta 201, confirmando la inserción exitosa del usuario.

## 3. Herramientas utilizadas

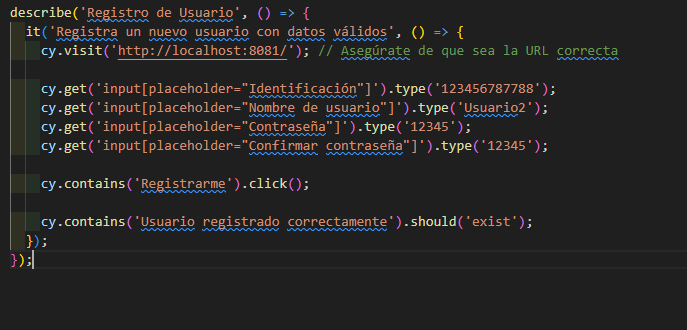
 **Cypress**: Framework de pruebas end-to-end.

 **Expo**: Plataforma para ejecutar la app React Native en desarrollo.

 **Supabase**: Backend utilizado para almacenar los usuarios registrados.

 **React Native**: Framework utilizado para construir la app.

## 4. Código del Test



## 5. Resultados esperados

 El usuario debe ser registrado correctamente con datos válidos.

 Se debe recibir una respuesta HTTP **201 Created** desde Supabase.

 El mensaje **"Usuario registrado correctamente"** debe mostrarse en pantalla.

## 

## 6. Estado actual

* El test fue ejecutado con éxito usando datos únicos en cada ejecución (evitando conflictos por duplicados).
* El mensaje esperado se muestra correctamente en pantalla.
* El test es **estable** y puede reutilizarse para futuras validaciones del flujo de registro.

# 