Documento de Pruebas Unitarias

“SKIPUR”

Integrantes:

Chavez Oscullo Klever Enrique

Guacan Rivera Alexander David

Trejo Duque Alex Fernando

Fecha: 2025-07-29

1. Introducción y Metodología de Pruebas Unitarias

El presente documento detalla los casos de prueba unitarios diseñados e implementados para el frontend del proyecto SKIPUR. El objetivo de estas pruebas es verificar la correctitud lógica de los componentes individuales del sistema de forma aislada, garantizando que cada pieza de la lógica de negocio funcione según lo especificado.

Se ha utilizado el framework de pruebas **Vitest**, en conjunto con la librería **vitest-mock-extended**, para llevar a cabo las pruebas en un entorno de Node.js con TypeScript. La metodología se centra en la **capa de Servicios**, ya que esta contiene la lógica de negocio más pura y crítica del sistema.

Cada prueba sigue rigurosamente el patrón **Arrange-Act-Assert (AAA)**:

- **Arrange (Preparar):** Se inicializa el entorno de la prueba, creando mocks (simulaciones) de las dependencias externas (como los repositorios de base de datos o los servicios de correo) para aislar el componente bajo prueba.
- **Act (Actuar):** Se ejecuta el método o función que se está probando, pasándole las entradas de prueba definidas.
- **Assert (Verificar):** Se comprueba que el resultado obtenido (el valor de retorno o un error lanzado) coincide con el resultado esperado.

2. Casos de Prueba Unitarios por Módulo

A continuación, se documentan los casos de prueba para cada módulo funcional.

Módulo: Autentificación

Caso de Prueba	Entrada de Prueba (Arrange)	Resultado Esperado (Assert)	Resultado Obtenido	Estado
1. Login exitoso devuelve usuario autenticado	- loginDto con email y password válidos - loginRequest devuelve token y user DTO - mapFromUserDto devuelve user mapeado - setToken guarda el token	Se llama a loginRequest, setToken, mapFromUserDto, y se retorna el usuario mapeado	Coincide con el resultado esperado	Aprobada
2. Login falla por credenciales inválidas	- loginDto con email y password incorrectos - loginRequest lanza error	Lanza excepción con mensaje "Email o contraseña incorrectos"	Coincide con el resultado esperado	Aprobada
3. Logout elimina el token	- Se espía removeToken- Se llama a logout()	removeToken debe ser llamado	Coincide con el resultado esperado	Aprobada
4. getAuthUser con token válido	- getToken retorna un token - jwtDecode retorna payload válido - mapFromAccessTokenPayload mapea a AuthUser	Se llama a getToken, jwtDecode, mapFromAccessTokenPayload y retorna el usuario autenticado	Coincide con el resultado esperado	Aprobada
5. getAuthUser sin token	- getToken retorna null	Retorna null	Coincide con el resultado esperado	Aprobada
6. getAuthUser con token inválido	- getToken retorna un token - jwtDecode lanza excepción	Retorna null	Coincide con el resultado esperado	Aprobada

Módulo: Registro de paciente

Caso de Prueba	Entrada de Prueba (Arrange)	Resultado Esperado (Assert)	Resultado Obtenido	Estado
7. Registro exitoso de paciente	- Objeto registration con datos válidos - mapToRegistratePatientDto retorna un DTO válido - registerPatientRequest resuelve sin error	Se llama a mapToRegistratePatientDto y a registerPatientRequest con el DTO correspondiente	Coincide con el resultado esperado	Aprobada
8. Error al registrar paciente por fallo en el mapeo	- mapToRegistratePatientDto lanza error	Se lanza excepción con mensaje: "El usuario no pudo ser registrado"	Coincide con el resultado esperado	Aprobada

Módulo: Gestión de especialidades

Caso de Prueba	Entrada de Prueba (Arrange)	Resultado Esperado (Assert)	Resultado Obtenido	Estado
9. Obtener especialidades mapeadas	- getSpecialtiesRequest devuelve lista de SpecialtyDto - mapFromSpecialtyDto convierte cada DTO a Specialty	Se llama a getSpecialtiesRequest, se mapea correctamente y retorna lista de especialidades	Coincide con el resultado esperado	Aprobada
10. Crear una especialidad correctamente	- Objeto CreateSpecialty válido - mapToCreateSpecialtyDto retorna DTO - createSpecialtyRequest retorna SpecialtyDto - mapFromSpecialtyDto retorna Specialty	Se llama a funciones de mapeo y API, y se retorna especialidad mapeada	Coincide con el resultado esperado	Aprobada
11. Actualizar una especialidad correctamente	- ID válido y UpdateSpecialty con datos - mapToUpdateSpecialtyDto genera DTO - updateSpecialtyRequest responde con SpecialtyDto	Se llama a updateSpecialtyRequest con ID y DTO, se mapea y retorna la especialidad	Coincide con el resultado esperado	Aprobada
12. Eliminar una especialidad por ID	- ID válido - deleteSpecialtyRequest no lanza error	Se llama a deleteSpecialtyRequest con el ID indicado	Coincide con el resultado esperado	Aprobada
13. Obtener especialidad por ID	- ID válido - getSpecialtyByIdRequest retorna SpecialtyDto - mapFromSpecialtyDto transforma a Specialty	Se llama a getSpecialtyByIdRequest, se mapea y se retorna la especialidad	Coincide con el resultado esperado	Aprobada

Módulo: Gestión de especialistas

Caso de Prueba	Entrada de Prueba (Arrange)	Resultado Esperado (Assert)	Resultado Obtenido	Estado
14. Obtener especialistas mapeados	- getSpecialistsRequest devuelve una lista de SpecialistDto - mapFromSpecialistDto transforma cada DTO a Specialist	Se llaman las funciones correspondientes y se retorna lista de especialistas mapeados	Coincide con el resultado esperado	Aprobada
15. Crear un especialista correctamente	- Objeto CreateSpecialist válido	Se ejecutan funciones de mapeo y API y se retorna especialista mapeado	Coincide con el resultado esperado	Aprobada

	- mapToCreateSpecialistDto devuelve DTO - createSpecialistRequest devuelve SpecialistDto - mapFromSpecialistDto devuelve Specialist			
16. Actualizar un especialista correctamente	- ID y UpdateSpecialist válidos - mapToUpdateSpecialistDto genera DTO - updateSpecialistRequest responde con SpecialistDto	Se llaman funciones de mapeo y API, y se retorna especialista actualizado	Coincide con el resultado esperado	Aprobada
17. Eliminar un especialista por ID	- ID válido - deleteSpecialistRequest no lanza error	Se llama a deleteSpecialistRequest con el ID indicado	Coincide con el resultado esperado	Aprobada

Módulo: Gestión de especialistas

Caso de Prueba	Entrada de Prueba (Arrange)	Resultado Esperado (Assert)	Resultado Obtenido	Estado
18. Obtener disponibilidad por especialista (con fechas explícitas)	- specialistId válido - Fechas start y end definidas - getAvailabilityBySpecialistIdRequest retorna lista de AvailabilityDto	Llama correctamente a la API y retorna disponibilidades mapeadas	Coincide con el resultado esperado	Aprobada
19. Obtener disponibilidad por especialista (usando fechas por defecto)	- specialistId válido - No se pasan fechas- getStartWeek y getEndWeek retornan fechas por defecto	Se usan fechas por defecto, se llama a la API y se retorna lista vacía	Coincide con el resultado esperado	Aprobada
20. Error al obtener disponibilidades	- specialistId válido - La API lanza error	Se lanza excepción con mensaje: "El usuario no tiene registrado ningun horario"	Coincide con el resultado esperado	Aprobada
21. Crear una disponibilidad correctamente	- Objeto CreateAvailability válido - DTO generado por mapToCreateAvailabilityDto - API responde con AvailabilityDto	Se mapea correctamente y retorna la disponibilidad creada	Coincide con el resultado esperado	Aprobada
22. Error al crear disponibilidad	- Objeto CreateAvailability válido - API lanza error	Se lanza excepción con mensaje: "No se pudo registrar el horario"	Coincide con el resultado esperado	Aprobada
23. Actualizar disponibilidad correctamente	- ID y objeto UpdateAvailability válidos - API responde con AvailabilityDto	Se mapea y retorna la disponibilidad actualizada	Coincide con el resultado esperado	Aprobada

24. Error al actualizar disponibilidad	- ID válido - API lanza error	Se lanza excepción con mensaje: "No se pudo actualizar el horario del especialista"	Coincide con el resultado esperado	Aprobada
25. Eliminar disponibilidad correctamente	- ID válido - deleteAvailabilityRequest no lanza error	Se llama correctamente a la API con el ID	Coincide con el resultado esperado	Aprobada
26. Error al eliminar disponibilidad	- ID válido - API lanza error	Se lanza excepción con mensaje: "No se pudo eliminar el horario de disponibilidad"	Coincide con el resultado esperado	Aprobada

Resultados y conclusiones

Las pruebas unitarias realizadas abarcan los módulos de autenticación, registro de pacientes, especialidades, especialistas y disponibilidad horaria. En general, se validó exitosamente el correcto funcionamiento de las operaciones principales de cada servicio, incluyendo flujos positivos (como creación, obtención y actualización de datos) y negativos (manejo de errores y excepciones). Se comprobó que los datos son mapeados correctamente entre DTOs y entidades internas, y que las respuestas de la API se integran adecuadamente con la lógica de negocio.

En cuanto al manejo de errores, se verificó que las excepciones son capturadas y transformadas en mensajes claros para el usuario. Esto garantiza un comportamiento robusto y predecible del sistema ante fallas externas o entradas inválidas. En conclusión, las pruebas unitarias reflejan una buena cobertura y confiabilidad en la implementación de los servicios, asegurando una base sólida para futuras integraciones y mantenimientos del sistema.

Anexos

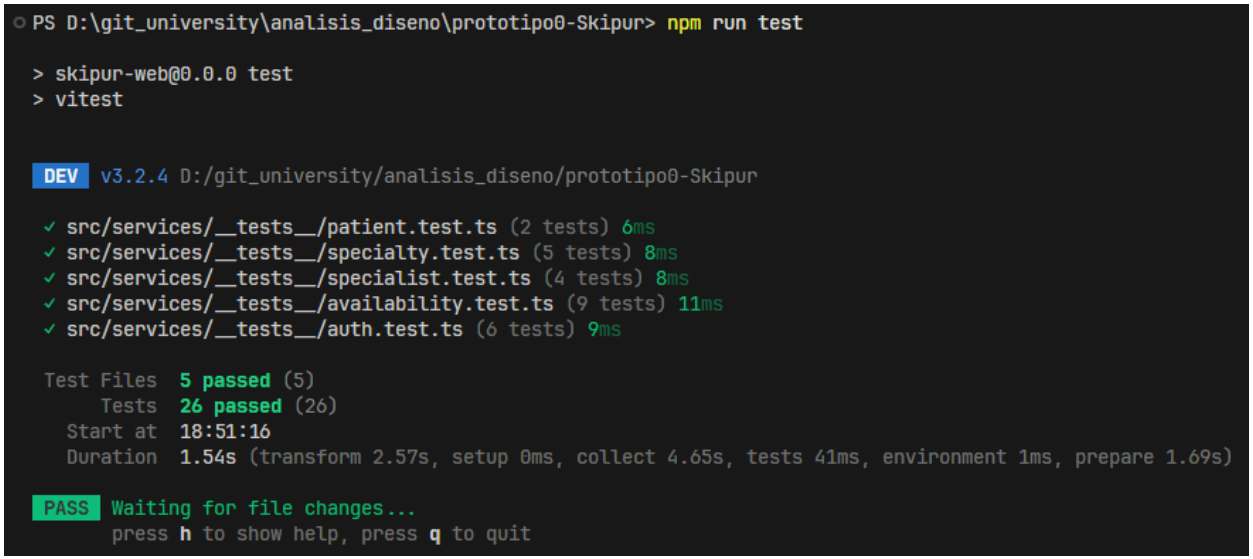

Figura 1. Ejecución de pruebas exitosas