**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 backend 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 Aplicación (Casos de Uso)**, 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: Gestión de Disponibilidad (availability)**

**Componente Probado: CreateAvailabilityUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 14. Creación exitosa | - DTO con specialist\_id y rango de tiempo válido.  - Mock de repo.findManyBySpecialistId devuelve []. | El caso de uso debe llamar a repo.create y devolver el nuevo bloque. | Pasa | Aprobada |
| 15. Falla por solapamiento | - DTO con rango de tiempo que entra en conflicto con un horario existente.  - Mock de repo.findManyBySpecialistId devuelve un bloque que se solapa. | Debe lanzar un Error: "El nuevo horario se solapa...". El método create no debe ser llamado. | Pasa | Aprobada |
| 16. Falla por falta de ID | - DTO sin specialist\_id. | Debe lanzar un Error: "El ID del especialista es requerido.". Ningún método del repositorio debe ser llamado. | Pasa | Aprobada |

**Componente Probado: CreateWeeklyScheduleUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 17. Creación de horario semanal | - DTO con specialist\_id y un objeto schedule con días y horas.  - Mock de repo.findManyBySpecialistId devuelve []. | Debe calcular las fechas para la próxima semana, generar los slots y llamar a repo.createMany. Devuelve { count: X }. | Pasa | Aprobada |
| 18. Falla por solapamiento semanal | - DTO con schedule que genera un slot conflictivo.  - Mock de repo.findManyBySpecialistId devuelve un bloque que se solapa. | Debe lanzar un Error: "...se solapa con un bloque ya existente.". El método createMany no debe ser llamado. | Pasa | Aprobada |
| 19. Falla por horario vacío | - DTO con specialist\_id pero un objeto schedule vacío. | Debe lanzar un Error: "El horario proporcionado no contiene ningún bloque de tiempo válido.". | Pasa | Aprobada |

**Componente Probado: DeleteAvailabilityUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 20. Eliminación exitosa | - id de un bloque de horario existente y no reservado (is\_booked: false).  - Mock de repo.findById devuelve el bloque. | Debe llamar a repo.delete y devolver el objeto del bloque eliminado. | Pasa | Aprobada |
| 21. Falla por no encontrar | - id de un bloque que no existe.  - Mock de repo.findById devuelve null. | Debe lanzar un Error: "Bloque de disponibilidad no encontrado.". | Pasa | Aprobada |
| 22. Falla por estar reservado | - id de un bloque ya reservado (is\_booked: true).  - Mock de repo.findById devuelve el bloque reservado. | Debe lanzar un Error: "No se puede eliminar un horario que ya tiene una cita agendada.". | Pasa | Aprobada |

**Componente Probado:**UpdateAvailabilityUseCase

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 23. Actualización exitosa | - id y DTO de un bloque existente y no reservado.  - Mock de repo.findById devuelve el bloque.  - Mock de repo.findManyBySpecialistId devuelve []. | Debe llamar a repo.update y devolver el objeto del bloque actualizado. | Pasa | Aprobada |
| 24. Falla por estar reservado | - id de un bloque ya reservado (is\_booked: true).  - Mock de repo.findById devuelve el bloque reservado. | Debe lanzar un Error: "No se puede modificar un horario que ya tiene una cita agendada.". | Pasa | Aprobada |
| 25. Falla por solapamiento | - id y DTO con un rango de tiempo que entra en conflicto con *otro* horario.  - Mock de repo.findManyBySpecialistId devuelve un bloque diferente que se solapa. | Debe lanzar un Error: "El horario actualizado se solapa con otro...". | Pasa | Aprobada |

**Componente Probado: GetAvailabilitiesBySpecialistUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 26. Obtener disponibilidad | - specialistId y un rango de fechas.  - Mock de repo.findManyBySpecialistId devuelve un array de bloques. | Debe llamar al repositorio con los parámetros correctos y devolver el array de bloques de disponibilidad. | Pasa | Aprobada |
| 27. Obtener disponibilidad vacía | - specialistId y rango de fechas.  - Mock de repo.findManyBySpecialistId devuelve []. | Debe llamar al repositorio y devolver un array vacío. | Pasa | Aprobada |

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Módulo: Gestión de Especialistas (specialist)**

**Componente Probado: CreateSpecialistUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 28. Creación exitosa de especialista | - DTO con datos válidos (email, full\_name, etc.).  - Mock de userRepository.findByEmail devuelve null.  - Mock de bcrypt.hash devuelve un hash simulado. | Se debe crear el especialista, hashear la contraseña, llamar a emailService.sendSpecialistWelcomeEmail y devolver el nuevo objeto de usuario. | Pasa | Aprobada |
| 29. Falla por email duplicado | - DTO con un email que ya existe.  - Mock de userRepository.findByEmail devuelve un usuario existente. | Debe lanzar un Error: "El correo electrónico ya está en uso.". Los métodos create del repositorio y send...Email del servicio no deben ser llamados. | Pasa | Aprobada |

**Componente Probado: DeactivateSpecialistUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 30. Desactivación exitosa | - id de un especialista existente y activo.  - Mock de specialistRepository.findById devuelve un especialista con is\_active: true. | Debe llamar a specialistRepository.deactivate y devolver el objeto de usuario actualizado con is\_active: false. | Pasa | Aprobada |
| 31. Falla por no encontrar | - id de un especialista que no existe.  - Mock de specialistRepository.findById devuelve null. | Debe lanzar un Error: "Especialista no encontrado.". | Pasa | Aprobada |
| 32. Falla por estar ya inactivo | - id de un especialista ya inactivo.  - Mock de specialistRepository.findById devuelve un especialista con is\_active: false. | Debe lanzar un Error: "Este especialista ya se encuentra inactivo.". | Pasa | Aprobada |

**Componente Probado: UpdateSpecialistUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 33. Actualización exitosa | - id de un especialista existente y un DTO con los datos a cambiar.  - Mock de specialistRepository.findById devuelve el especialista. | Debe llamar a specialistRepository.update con el id y el DTO, y devolver el objeto del especialista actualizado. | Pasa | Aprobada |
| 34. Falla por no encontrar | - id de un especialista que no existe.  - Mock de specialistRepository.findById devuelve null. | Debe lanzar un Error: "Especialista no encontrado.". El método update no debe ser llamado. | Pasa | Aprobada |

**Componente Probado: GetAllSpecialistsUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Prueba** | **Entrada de Prueba (Arrange)** | **Resultado Esperado (Assert)** | **Resultado Obtenido** | **Estado** |
| 35. Obtener solo activos (por defecto) | - Sin opciones de entrada (execute()).  - Mock de specialistRepository.findAll devuelve una lista de especialistas activos. | Debe llamar al repositorio con el parámetro onlyActive = true y devolver la lista de especialistas activos. | Pasa | Aprobada |
| 36. Obtener todos (incluyendo inactivos) | - Con la opción { includeInactive: true }.  - Mock de specialistRepository.findAll devuelve una lista de todos los especialistas. | Debe llamar al repositorio con el parámetro onlyActive = false y devolver la lista completa. | Pasa | Aprobada |

**Componente Probado: GetSpecialistByIdUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 37. Obtener especialista por ID | - id de un especialista existente.  - Mock de specialistRepository.findById devuelve el objeto del especialista. | Debe llamar al repositorio con el id correcto y devolver el objeto completo del especialista encontrado. | Pasa | Aprobada |
| 38. Falla por no encontrar ID | - id de un especialista que no existe.  - Mock de specialistRepository.findById devuelve null. | Debe llamar al repositorio y luego lanzar un Error: "Especialista no encontrado.". | Pasa | Aprobada |

**A screenshot of a computer

AI-generated content may be incorrect.**

**Módulo: Gestión de Especialidades (specialty)**

**Componente Probado: CreateSpecialtyUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 1. Creación exitosa | - DTO con name válido (ej. "Psicología Infantil").  - Mock de repo.findByName devuelve null. | El caso de uso debe llamar a repo.create y devolver el nuevo objeto de especialidad. | Pasa | Aprobada |
| 2. Falla por nombre duplicado | - DTO con un name que ya existe.  - Mock de repo.findByName devuelve una especialidad existente. | Debe lanzar un Error: "Ya existe una especialidad con este nombre.". El método create no debe ser llamado. | Pasa | Aprobada |

**Componente Probado: DeactivateSpecialtyUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 3. Desactivación exitosa | - id de una especialidad existente y activa.  - Mock de repo.findById devuelve una especialidad con is\_active: true. | Debe llamar a repo.deactivate y devolver el objeto actualizado con is\_active: false. | Pasa | Aprobada |
| 4. Falla por no encontrar | - id de una especialidad que no existe.  - Mock de repo.findById devuelve null. | Debe lanzar un Error: "Especialidad no encontrada.". | Pasa | Aprobada |
| 5. Falla por estar ya inactiva | - id de una especialidad ya inactiva.  - Mock de repo.findById devuelve una especialidad con is\_active: false. | Debe lanzar un Error: "Esta especialidad ya se encuentra inactiva.". | Pasa | Aprobada |

**Componente Probado: UpdateSpecialtyUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 6. Actualización exitosa | - id de una especialidad existente y DTO con datos válidos.  - Mock de repo.findById devuelve la especialidad.  - Mock de repo.findByName devuelve null. | Debe llamar a repo.update con el id y el DTO, y devolver el objeto de la especialidad actualizado. | Pasa | Aprobada |
| 7. Falla por no encontrar | - id de una especialidad que no existe.  - Mock de repo.findById devuelve null. | Debe lanzar un Error: "Especialidad no encontrada.". El método update no debe ser llamado. | Pasa | Aprobada |
| 8. Falla por nombre duplicado | - DTO con un name que ya pertenece a *otra* especialidad.  - Mock de repo.findById devuelve la especialidad a actualizar.  - Mock de repo.findByName devuelve la otra especialidad con el mismo nombre. | Debe lanzar un Error: "Ya existe otra especialidad con ese nombre.". | Pasa | Aprobada |

**Componente Probado: GetAllSpecialtiesUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 9. Obtener solo activas (por defecto) | - Sin opciones de entrada (execute()).  - Mock de repo.findAll devuelve una lista de especialidades activas. | Debe llamar al repositorio con el parámetro onlyActive = true y devolver la lista de especialidades activas. | Pasa | Aprobada |
| 10. Obtener todas (incluyendo inactivas) | - Con la opción { includeInactive: true }.  - Mock de repo.findAll devuelve una lista de todas las especialidades. | Debe llamar al repositorio con el parámetro onlyActive = false y devolver la lista completa. | Pasa | Aprobada |

**Componente Probado: GetSpecialtyByIdUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 11. Obtener especialidad por ID | - id de una especialidad existente.  - Mock de repo.findById devuelve el objeto de la especialidad. | Debe llamar al repositorio con el id correcto y devolver el objeto completo de la especialidad encontrada. | Pasa | Aprobada |
| 12. Falla por no encontrar ID | - id de una especialidad que no existe.  - Mock de repo.findById devuelve null. | Debe llamar al repositorio y luego lanzar un Error: "Especialidad no encontrada.". | Pasa | Aprobada |
| 13. Obtener especialidad inactiva | - id de una especialidad existente con is\_active: false.  - Mock de repo.findById devuelve el objeto de la especialidad inactiva. | Debe devolver el objeto de la especialidad inactiva, ya que la lógica actual lo permite. | Pasa | Aprobada |

**A screen shot of a computer

AI-generated content may be incorrect.**

**Módulo: Autenticación (auth)**

**Componente Probado: RegisterUserUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 39. Registro exitoso de cliente | - DTO con datos válidos para un nuevo cliente y su paciente.  - Mock de userRepository.findByEmail devuelve null.  - Mock de bcrypt.hash devuelve un hash simulado. | Se debe autogenerar una contraseña, hashearla, llamar a userRepository.create, luego a emailService.sendWelcomeEmail y finalmente devolver el nuevo objeto de usuario. | Pasa | Aprobada |
| 40. Falla por email duplicado | - DTO con un email que ya existe en la base de datos.  - Mock de userRepository.findByEmail devuelve un usuario existente. | Debe lanzar un Error: "El correo electrónico ya está en uso.". No se debe llamar a bcrypt.hash, create ni sendWelcomeEmail. | Pasa | Aprobada |

**Componente Probado: LoginUserUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 41. Login exitoso | - DTO con email y password correctos.  - Mock de userRepository.findByEmail devuelve el usuario.  - Mock de bcrypt.compare devuelve true.  - Mock de jwt.sign devuelve un token. | Debe llamar a todos los mocks en secuencia y devolver un objeto con el token y los datos del user. | Pasa | Aprobada |
| 42. Falla por usuario no encontrado | - DTO con un email que no existe.  - Mock de userRepository.findByEmail devuelve null. | Debe lanzar un Error: "Credenciales inválidas.". No debe llamar a bcrypt.compare ni a jwt.sign. | Pasa | Aprobada |
| 43. Falla por contraseña incorrecta | - DTO con password incorrecta.  - Mock de userRepository.findByEmail devuelve el usuario.  - Mock de bcrypt.compare devuelve false. | Debe lanzar un Error: "Credenciales inválidas.". No debe llamar a jwt.sign. | Pasa | Aprobada |
| 44. Falla por falta de JWT Secret | - DTO con credenciales correctas.  - Variable de entorno JWT\_SECRET no está definida. | Debe lanzar un Error: "La clave secreta para JWT no está configurada.". No debe llamar a jwt.sign. | Pasa | Aprobada |

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Módulo: Gestión de Citas (appointment)**

**Componente Probado: ReserveAppointmentUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 45. Reserva exitosa | - availabilityId de un bloque existente y no reservado (is\_booked: false).  - patientId de un paciente válido.  - Mock de availabilityRepo.findById devuelve el bloque. | Debe llamar a appointmentRepo.create con los datos correctos y devolver el nuevo objeto de cita. | Pasa | Aprobada |
| 46. Falla por disponibilidad no encontrada | - availabilityId de un bloque que no existe.  - Mock de availabilityRepo.findById devuelve null. | Debe lanzar un Error: "El horario seleccionado no existe.". El método create de citas no debe ser llamado. | Pasa | Aprobada |
| 47. Falla por disponibilidad ya reservada | - availabilityId de un bloque con is\_booked: true.  - Mock de availabilityRepo.findById devuelve el bloque reservado. | Debe lanzar un Error: "Este horario ya no está disponible. Por favor, seleccione otro.". El método create no debe ser llamado. | Pasa | Aprobada |

**Componente Probado: GetMyAppointmentsUseCase**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso de Prueba | Entrada de Prueba (Arrange) | Resultado Esperado (Assert) | Resultado Obtenido | Estado |
| 48. Obtener citas exitosamente | - clientId de un usuario que tiene un paciente asociado.  - Mock de userRepo.findById devuelve el User con su Patient anidado.  - Mock de appointmentRepo.findManyByPatientId devuelve un array de citas. | Debe llamar a appointmentRepo.findManyByPatientId con el ID del paciente correcto y devolver la lista de citas. | Pasa | Aprobada |
| 49. Cliente sin citas | - clientId de un usuario con paciente, pero sin citas.  - Mock de userRepo.findById devuelve el usuario.  - Mock de appointmentRepo.findManyByPatientId devuelve []. | Debe devolver un array vacío []. | Pasa | Aprobada |
| 50. Falla por cliente no encontrado | - clientId de un usuario que no existe.  - Mock de userRepo.findById devuelve null. | Debe lanzar un Error: "No se encontró un paciente asociado a este cliente.". El repositorio de citas no debe ser llamado. | Pasa | Aprobada |
| 51. Falla por cliente sin paciente | - clientId de un usuario que existe, pero su relación representative\_for está vacía.  - Mock de userRepo.findById devuelve el usuario sin pacientes. | Debe lanzar un Error: "No se encontró un paciente asociado a este cliente.". El repositorio de citas no debe ser llamado. | Pasa | Aprobada |

A screen shot of a computer program

AI-generated content may be incorrect.

**Resultados**

La suite de pruebas unitarias implementada para el backend del sistema SKIPUR ha cubierto los módulos funcionales clave, incluyendo la autenticación de usuarios, la gestión de especialidades, el manejo de perfiles de especialistas y la compleja lógica de la disponibilidad horaria. El objetivo principal de esta fase fue validar la correctitud de la lógica de negocio de forma aislada, garantizando que cada componente se comporte según lo esperado bajo diversas condiciones.

En general, se validó exitosamente el correcto funcionamiento de las operaciones principales, abarcando no solo los flujos positivos ('happy paths'), sino también una variedad de flujos de error y casos límite. Esto incluye la validación de datos de entrada (DTOs), el manejo de credenciales inválidas en la autenticación, la prevención de registros duplicados (como correos electrónicos o especialidades con el mismo nombre), y la correcta aplicación de reglas de negocio críticas, como la detección de solapamiento de horarios y la protección contra la modificación de citas ya reservadas.

La metodología de pruebas, centrada en los Casos de Uso (Use Cases), ha permitido validar de forma efectiva la eficacia de la Clean Architecture implementada. Al simular (mockear) las dependencias de la capa de infraestructura —tales como los repositorios de base de datos (IUserRepository, ISpecialtyRepository, etc.) y los servicios externos (IEmailService)—, se ha logrado probar la lógica de negocio en completo aislamiento. Este enfoque confirma que cada componente cumple con su responsabilidad única y que el sistema de inyección de dependencias funciona correctamente.