|  |
| --- |
|  |
| Prueba de Caja Blanca |
| ***“SKIPUR”*** |
|  |
|  |
| **Integrantes:**  **Chavez Oscullo Klever Enrique**  **Guacan Rivera Alexander David**  **Trejo Duque Alex Fernando**  **Fecha 2025-07-08** |
|  |

**Prueba caja blanca para el Requisito Funcional: Crear una Nueva Especialidad**

El propósito de esta prueba es analizar la estructura interna del código responsable de la funcionalidad de creación de especialidades médicas. Se busca garantizar que todas las rutas lógicas, condiciones y flujos de control dentro del módulo de frontend sean evaluadas para identificar posibles errores y asegurar su correcto funcionamiento al momento de enviar el formulario desde la interfaz.

1. **CÓDIGO FUENTE**

El fragmento de código seleccionado para el análisis corresponde a la función handleSubmit definida dentro del componente SpecialtyForm. Esta función encapsula la lógica que se ejecuta cuando el usuario envía el formulario para crear una nueva especialidad, controlando el estado del formulario, reiniciando los campos si es necesario y gestionando errores.

// src/components/forms/SpecialtyForm.tsx

const handleSubmit = async (

values: FormValues,

{ setSubmitting, resetForm }: FormikHelpers<FormValues>

) => {

try {

await onSubmit?.(values)

if (mode === 'add') {

resetForm()

}

} catch {

onError?.()

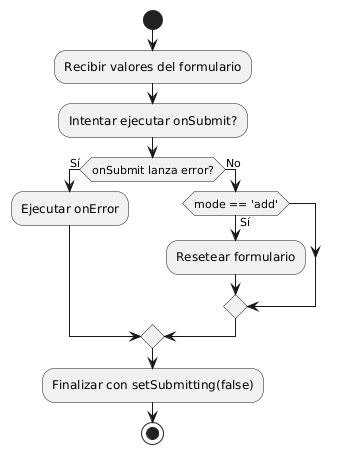
} finally {

setSubmitting(false)

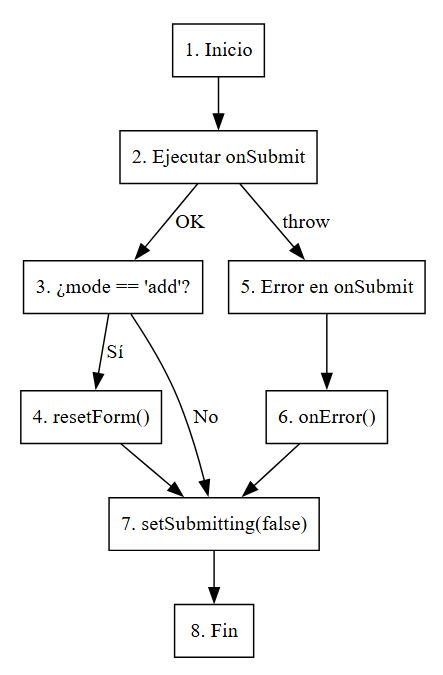
}

}

1. **DIAGRAMA DE FLUJO (DF)**



1. **GRAFO DE FLUJO (GF)**



1. **IDENTIFIACCIÒN DE LAS RUTAS (Camino básico)**

En base al grafo de flujo, se identifican los caminos lógicamente independientes:

* R1 (Camino de Éxito con modo "add"):

1 → 2 → 3 (true) → 4 → 7 → 8

Descripción: Se ejecuta onSubmit, el modo es "add", se resetea el formulario, finaliza.

* R2 (Camino de Éxito con modo "edit"):

1 → 2 → 3 (false) → 7 → 8

Descripción: Se ejecuta onSubmit, el modo es "edit", no se resetea el formulario, finaliza.

* R3 (Error en envío):

1 → 2 (throw) → 5 → 6 → 7 → 8

Descripción: onSubmit lanza un error, se ejecuta onError, se finaliza.

1. **COMPLEJIDAD CICLOMÁTICA**

La complejidad ciclomática **V(G)** mide el número de caminos independientes que deben ser probados para garantizar la cobertura completa.

* **Fórmula por nodos predicado**:
  + Nodos de decisión: mode === 'add', try/catch
  + P = 2
  + V(G) = P + 1 = 2 + 1 = 3
* **Fórmula por aristas y nodos**:
  + N = 8 (nodos identificados)
  + A = 9 (aristas: 1→2, 2→3, 3→4, 3→7, 2→5, 4→7, 5→6, 6→7, 7→8)
  + V(G) = A - N + 2 = 9 - 8 + 2 = 3

Se requieren al menos 3 pruebas para cubrir todas las rutas posibles.

**Prueba de Caja Blanca para el Requisito Funcional: Crear Disponibilidad Horaria**

El propósito de esta prueba es analizar la estructura interna del código responsable de crear un nuevo bloque de disponibilidad para un especialista. El enfoque principal es validar el flujo de control que previene la creación de horarios que se solapan con bloques ya existentes, garantizando la integridad de la agenda del especialista.

1. **CÓDIGO FUENTE**

El fragmento de código seleccionado es el método execute de la clase CreateAvailabilityUseCase. Este método contiene la lógica para verificar si un nuevo bloque de tiempo entra en conflicto con otros antes de proceder con su creación.

**// src/application/use-cases/availability/create-availability.use-case.ts**

**import { IAvailabilityRepository } from "../../../domain/repositories/availability.repository";**

**import { CreateAvailabilityDto } from "../../dtos/availability.dtos";**

**export class CreateAvailabilityUseCase {**

**constructor(private readonly availabilityRepository: IAvailabilityRepository) {}**

**async execute(dto: CreateAvailabilityDto) {**

**const { specialist\_id, start\_time, end\_time } = dto;**

**if (!specialist\_id) {**

**throw new Error("El ID del especialista es requerido.");**

**}**

**// 1. Convertir fechas a objetos Date**

**const startDate = new Date(start\_time);**

**const endDate = new Date(end\_time);**

**// 2. Verificar si ya existe un bloque que se solape con el nuevo**

**const existingAvailabilities = await this.availabilityRepository.findManyBySpecialistId(specialist\_id, startDate, endDate);**

**const isOverlapping = existingAvailabilities.some(existing =>**

**(startDate < new Date(existing.end\_time)) && (endDate > new Date(existing.start\_time))**

**);**

**if (isOverlapping) {**

**throw new Error("El nuevo horario se solapa con un bloque de disponibilidad existente.");**

**}**

**// 3. Crear el nuevo bloque de disponibilidad**

**return this.availabilityRepository.create({**

**specialist\_id,**

**start\_time,**

**end\_time,**

**});**

**}**

**}**

1. **DIAGRAMA DE FLUJO (DF)**

A diagram of a diagram

AI-generated content may be incorrect.

1. **GRAFO DE FLUJO (GF)**

A diagram of a algorithm

AI-generated content may be incorrect.

1. **IDENTIFIACCIÒN DE LAS RUTAS (Camino basico)**

Se identifican los caminos linealmente independientes a través del Grafo de Flujo.

* **R1 (Camino de Éxito):** 1 -> 2 -> 3 -> 4 -> 5 -> 6
  + *Descripción:* Se provee un specialist\_id, no se encuentra ningún horario que se solape, y el nuevo bloque de disponibilidad se crea exitosamente.
* **R2 (Error - Horario se solapa):** 1 -> 2 -> 3 -> 4 -> 6
  + *Descripción:* Se provee un specialist\_id, pero la búsqueda en la base de datos encuentra al menos un bloque de horario existente que entra en conflicto con las nuevas fechas.
* **R3 (Error - Falta ID de especialista):** 1 -> 2 -> 6
  + *Descripción:* No se proporciona el specialist\_id en los datos de entrada, por lo que la operación se detiene antes de consultar la base de datos.

1. **COMPLEJIDAD CICLOMÁTICA**

Se calcula la Complejidad Ciclomática V(G) para determinar la complejidad del código.

* **V(G) = P + 1** (Número de nodos predicado + 1)
  + Nodos predicado (decisiones): **Nodo 2** (specialist\_id == null?) y **Nodo 4** (isOverlapping?).
  + P = 2
  + **V(G) = 2 + 1 = 3**
* **V(G) = A – N + 2** (Número de aristas - Número de nodos + 2)
  + Número de Nodos (N) = 6 (nodos del 1 al 6).
  + Número de Aristas (A) = 7 (las flechas en el grafo).
  + V(G) = 7 - 6 + 2 = 1 + 2
  + **V(G) = 3**

**Prueba de Caja Blanca para el Requisito Funcional: Desactivar una Especialidad**

El objetivo de esta prueba es analizar la lógica interna del caso de uso encargado de desactivar una especialidad. El análisis se centra en las validaciones previas que aseguran que la especialidad exista y que no se intente desactivar una que ya está inactiva, garantizando así la consistencia del estado del sistema.

1. **CÓDIGO FUENTE**

El fragmento de código a analizar es el método execute de la clase DeactivateSpecialtyUseCase.

// src/application/use-cases/specialty/deactivate-specialty.use-case.ts

import { ISpecialtyRepository } from "../../../domain/repositories/specialty.repository";

export class DeactivateSpecialtyUseCase {

constructor(private readonly specialtyRepository: ISpecialtyRepository) {}

async execute(id: string) {

// 1. Verificar que la especialidad exista

const specialty = await this.specialtyRepository.findById(id);

if (!specialty) {

throw new Error("Especialidad no encontrada.");

}

// 2. Verificar que no esté ya inactiva

if (!specialty.is\_active) {

throw new Error("Esta especialidad ya se encuentra inactiva.");

}

// 3. Llamar al repositorio para realizar la desactivación (actualización)

return this.specialtyRepository.deactivate(id);

}

}

1. **DIAGRAMA DE FLUJO (DF)**

A diagram with text and words

AI-generated content may be incorrect.

1. **GRAFO DE FLUJO (GF)**

A diagram of a flowchart

AI-generated content may be incorrect.

1. **IDENTIFIACCIÒN DE LAS RUTAS (Camino basico)**

Los caminos linealmente independientes a través del Grafo de Flujo son:

* **R1 (Camino de Éxito):** 1 -> 2 -> 3 -> 4 -> 5 -> 6
  + *Descripción:* Se encuentra la especialidad, está activa, se procede a desactivarla y la operación finaliza con éxito.
* **R2 (Error - Especialidad no encontrada):** 1 -> 2 -> 3 -> 6
  + *Descripción:* El id proporcionado no corresponde a ninguna especialidad en la base de datos.
* **R3 (Error - Especialidad ya inactiva):** 1 -> 2 -> 3 -> 4 -> 6
  + *Descripción:* Se encuentra la especialidad, pero su estado is\_active ya es false.

1. **COMPLEJIDAD CICLOMÁTICA**

Se calcula la Complejidad Ciclomática V(G) del fragmento de código.

* **V(G) = P + 1** (Número de nodos predicado + 1)
  + Nodos predicado (decisiones): **Nodo 3** (specialty == null?) y **Nodo 4** (!is\_active?).
  + P = 2
  + **V(G) = 2 + 1 = 3**
* **V(G) = A – N + 2** (Número de aristas - Número de nodos + 2)
  + Número de Nodos (N) = 6.
  + Número de Aristas (A) = 7.
  + V(G) = 7 - 6 + 2 = 1 + 2
  + **V(G) = 3**

**Prueba de Caja Blanca para el Requisito Funcional: Iniciar Sesión de Usuario**

El objetivo de esta prueba es analizar el flujo lógico del caso de uso responsable de la autenticación de usuarios. Se examinarán las validaciones de credenciales, incluyendo la verificación de la existencia del usuario por su correo electrónico y la comparación de la contraseña proporcionada con su hash almacenado, para garantizar un proceso de inicio de sesión seguro y correcto.

1. **CÓDIGO FUENTE**

El fragmento de código a analizar es el método execute de la clase LoginUserUseCase.

// src/application/use-cases/auth/login-user.use-case.ts

import { IUserRepository } from "../../../domain/repositories/user.repository";

import { LoginUserDto } from "../../dtos/auth.dtos";

import \* as bcrypt from 'bcryptjs';

import \* as jwt from 'jsonwebtoken';

export class LoginUserUseCase {

constructor(private readonly userRepository: IUserRepository) {}

async execute(dto: LoginUserDto): Promise<{ token: string; user: { id: string, email: string, full\_name: string, role: string } }> {

// 1. Buscar al usuario por su email

const user = await this.userRepository.findByEmail(dto.email);

if (!user) {

throw new Error("Credenciales inválidas."); // Mensaje genérico por seguridad

}

// 2. Comparar la contraseña enviada con el hash guardado en la BD

const isPasswordValid = await bcrypt.compare(dto.password, user.password\_hash);

if (!isPasswordValid) {

throw new Error("Credenciales inválidas."); // Mismo mensaje genérico

}

// 3. Generar el JSON Web Token (JWT)

const jwtSecret = process.env.JWT\_SECRET;

if (!jwtSecret) {

throw new Error("La clave secreta para JWT no está configurada.");

}

const payload = { id: user.id, email: user.email, role: user.role };

const token = jwt.sign(payload, jwtSecret, { expiresIn: '7d' });

// 4. Devolver el token y la información básica del usuario

return {

token,

user: { id: user.id, email: user.email, full\_name: user.full\_name, role: user.role }

};

}

}

1. **DIAGRAMA DE FLUJO (DF)**

A diagram of a user use case

AI-generated content may be incorrect.

1. **GRAFO DE FLUJO (GF)**

A diagram of a software program

AI-generated content may be incorrect.

1. **IDENTIFIACCIÒN DE LAS RUTAS (Camino basico)**

Los caminos linealmente independientes a través del Grafo de Flujo son:

* **R1 (Camino de Éxito):** 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8
  + *Descripción:* El usuario existe, la contraseña es correcta, la clave JWT está configurada y se genera y devuelve el token exitosamente.
* **R2 (Error - Usuario no encontrado):** 1 -> 2 -> 3 -> 8
  + *Descripción:* El correo electrónico proporcionado no corresponde a ningún usuario en la base de datos.
* **R3 (Error - Contraseña incorrecta):** 1 -> 2 -> 3 -> 4 -> 8
  + *Descripción:* El usuario existe, pero la contraseña proporcionada no coincide con el hash almacenado.
* **R4 (Error - Clave JWT no configurada):** 1 -> 2 -> 3 -> 4 -> 5 -> 8
  + *Descripción:* Las credenciales son válidas, pero el servidor no tiene configurada la variable de entorno JWT\_SECRET, lo que impide la creación del token.

1. **COMPLEJIDAD CICLOMÁTICA**

Se calcula la Complejidad Ciclomática V(G) del fragmento de código.

* **V(G) = P + 1** (Número de nodos predicado + 1)
  + Nodos predicado (decisiones): **Nodo 3** (user == null?), **Nodo 4** (!isPasswordValid?) y **Nodo 5** (!jwtSecret?).
  + P = 3
  + **V(G) = 3 + 1 = 4**
* **V(G) = A – N + 2** (Número de aristas - Número de nodos + 2)
  + Número de Nodos (N) = 8.
  + Número de Aristas (A) = 10.
  + V(G) = 10 - 8 + 2 = 2 + 2
  + **V(G) = 4**

**Prueba caja blanca de**  describa el requisito funcional

1. **CÓDIGO FUENTE**

Pegar el trozo de código fuente que se requiere para el caso de prueba

1. **DIAGRAMA DE FLUJO (DF)**

Realizar un DF del código fuente del numeral 1

1. **GRAFO DE FLUJO (GF)**

Realizar un GF en base al DF del numeral 2

1. **IDENTIFIACCIÒN DE LAS RUTAS (Camino basico)**

Determinar en base al GF del numeral 4

**RUTAS**

**R1:** 1

**R2:**

1. **COMPLEJIDAD CICLOMÁTICA**

Se puede calcular de las siguientes formas:

* V(G) = número de nodos predicados(decisiones)+1

V(G)=

* V(G) = A – N + 2

V(G)=

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos