Facultad de ingeniería

UNIVERSIDAD NACIONAL DE COLOMBIA

Entrega 06

Documentación Pruebas Unitarias

INGENIERÍA DE SOFTWARE 1

Maria Catalina Rodriguez Cardona

Julian David Velandia Neuta

Julian David Albarracin Galindo

Daniel Estiven Blanco Diaz

Diego Alejandro Arevalo Guevara

Profesor:

Oscar Eduardo Alvarez Rodriguez

Noviembre 16 de 2025

Introducción

En este documento presentamos las pruebas unitarias realizadas para el proyecto **Rocky**, nuestra aplicación pensada para acompañar a personas que están iniciando en el gimnasio. Rocky permite llevar registro del peso, estatura, rutinas y progreso, para que el usuario pueda ver sus avances y mantenerse motivado.

Las pruebas se enfocan en funcionalidades que consideramos esenciales dentro del sistema:

- Validación de los datos del perfil del usuario
- Creación del perfil, cálculo del IMC y registro del primer peso
- Validación del registro de usuarios (correo, contraseña, etc.)

Todas las pruebas se desarrollaron con **JUnit 5** y **Mockito** dentro de un proyecto Spring Boot, y se pueden ejecutar directamente desde el entorno del proyecto usando Maven o un IDE.

Entorno y herramientas utilizadas

- *Lenguaje:* Java
- *Framework:* Spring Boot
- *Pruebas:* JUnit 5
- *Mocks:* Mockito
- *Construcción:* Maven

Funcionalidades probadas

Área	Descripción
Validación de perfil	Reglas de nombres, edad, peso, estatura, etc.
Registro del perfil	Creación del perfil, IMC inicial y peso inicial.
Registro de usuario	Validación de email, duplicados y contraseña.
Contexto de Spring	Verificación de que el proyecto carga sin errores.

Pruebas realizadas

Pruebas sobre RegistroDTO

Clase de test: **RegistroDTOTest**
Clase bajo prueba: **RegistroDTO**

UT-DTO-01 – DTO válido sin errores

Método: dtoValido_noDebeTenerErrores()
Un DTO completo y correcto no debe generar violaciones de validación.

Resultado esperado: Sin errores.
Valores probados:
nombre = "Juan"
apellido = "Perez"
edad = 25
sexo = MASCULINO
peso = 70.5f
estatura = 1.75f
objetivo = MANTENERSE_EN_FORMA
rachaDeseada = 3

UT-DTO-02 – Nombre con caracteres inválidos

Método: nombreConNumeros_debeGenerarErrorDePattern()
Cuando el nombre contiene números, debe aparecer un error en ese campo.

Resultado esperado: Error en **nombre**.
Valor límite usado: nombre = "Ju4n"
Resto del DTO era válido.

UT-DTO-03 – Edad menor que el mínimo

Método: edadMenorQueMinimo_debeGenerarError()
La edad mínima es 14 años, así que 10 debe fallar.

Resultado esperado: Error en **edad**.
Valor límite usado: edad = 10

UT-DTO-04 – Peso menor que 10 kg

Método: pesoMenorQueDiez_debeGenerarError()
El peso mínimo aceptado es 10 kg.

Resultado esperado: Error en **peso**.
Valor límite usado: peso = 9.5f

UT-DTO-05 – Estatura mayor a 2.5 m

Método: estaturaMayorQueMaximo_debeGenerarError()
Una estatura de 2.8 m debe fallar, pues el máximo permitido es 2.5 m.

Resultado esperado: Error en **estatura**.
Valor límite usado: estatura = 2.8f

Pruebas sobre RegistroService

Clase de test: **RegistroServiceTest**
Clase bajo prueba: **RegistroService**

UT-SRV-01 – Usuario no existente

Método: registrarPerfilYPrimerPeso_usuarioNoEncontrado()
Si el usuario no existe, el método debe lanzar excepción y no guardar nada.

Resultado esperado: Se lanza excepción y no se guarda **Perfil** ni **Peso**.

Valores probados:
usuarioId = 1L
DTO válido creado con:
nombre = "Juan"
apellido = "Perez"
edad = 25
sexo = MASCULINO
peso = 70.0f
estatura = 1.75f
objetivo = null
rachaDeseada = 4

UT-SRV-02 – Creación del perfil con IMC y peso inicial

Método: registrarPerfilYPrimerPeso_conPesoYEstatura_creaPerfilYPesoConIMC()
Debe crearse el perfil con los datos del DTO, calcular el IMC y registrar el primer peso.

Resultado esperado: Perfil creado, IMC calculado y peso registrado.
Valores probados:
usuarioId = 1L
Usuario simulado → id = 1
DTO:
nombre = "Juan"
apellido = "Perez"
edad = 25
sexo = MASCULINO
peso = 70.0f
estatura = 1.75f
objetivo = null

rachaDeseada = 4
IMC esperado con assert: 70.0 / (1.75 * 1.75) = 22.8571f

UT-SRV-03 – Sin peso inicial: racha por defecto

Método: registrarPerfilYPrimerPeso_sinPeso_usaRachaPorDefectoYNoCreaPeso()
Si el usuario no ingresa peso, no se registra histórico y se usa racha = 1.

Resultado esperado: No se registra peso y rachaDeseada = 1.
Valores probados:
usuarioId = 2L
Usuario simulado → id = 2
DTO:
nombre = "Juan"
apellido = "Perez"
edad = 25
sexo = MASCULINO
peso = null
estatura = 1.75f
objetivo = null
rachaDeseada = null
Racha establecida por defecto: 1
Peso guardado: No se guarda ninguno (verificado con never()).

Pruebas sobre UsuarioService

Clase de test: **UsuarioServiceTest**
Clase bajo prueba: **UsuarioService**

UT-USR-01 – ValidarRegistro debería retornar lista de errores vacía cuando el usuario es válido

Método: Test0()
Resultado esperado: ValidarRegistro(u) devuelve una lista de errores vacía.
Valores probados:

Usuario con:

- email = "user@gmail.com"
- password = "Pass1234"

Mock: repo.existsByEmailIgnoreCase("user@gmail.com") devuelve false (el email NO está registrado).

UT-USR-02 – ValidarRegistro debería agregar error cuando el dominio del email no es válido

Método: Test3()
Resultado esperado: ValidarRegistro(u) devuelve una lista de errores no vacía. Se agrega al menos un error por dominio inválido.
Valores probados:

Usuario con:

- email = "user@pepe.com"
- password = "Pass1234"

Mock: repo.existsByEmailIgnoreCase("user@pepe.com") devuelve false (el email no existe aún)

UT-USR-03 – ValidarRegistro debería agregar error cuando el email ya esté registrado

Método: Test2()
Resultado esperado: ValidarRegistro(u) devuelve una lista de errores no vacía. La lista contiene el mensaje "El email ya está registrado".
Valores probados:

Usuario con:

- email = "user@gmail.com"
- password = "Pass1234"

Mock: repo.existsByEmailIgnoreCase("user@gmail.com") devuelve true (el email ya está registrado).

UT-USR-04 – ValidarRegistro debería agregar error cuando el email esté vacío

Método: Test3()
Resultado esperado: ValidarRegistro(u) devuelve una lista de errores no vacía. La lista contiene el mensaje "El email no puede estar vacío**".**
Valores probados:

Usuario inicialmente válido y luego:

- email = "" (cadena vacía)
- password = "Pass1234"

Mock: repo.existsByEmailIgnoreCase("") devuelve false.

UT-USR-05 – ValidarRegistro debería agregar errores cuando la contraseña sea débil

Método: Test4()
Resultado esperado: ValidarRegistro(u) devuelve una lista de errores no vacía. "La contraseña debe tener al menos 8 caracteres", "La contraseña debe tener al menos un número"
Valores probados:

Usuario con:

- email = "user@gmail.com"
- password = "user" (corta y sin números)

Mock: repo.existsByEmailIgnoreCase("user@gmail.com") devuelve fal**se.**

UT-USR-06 – crear() debería guardar el usuario cuando no hay errores de validación

Método: Test5()
Resultado esperado:
No se lanza excepción al invocar crear(u).
Al usuario retornado se le asigna id de 1L
El password del usuario se guarda codificado con el PasswordEncoder, valor "encoded".
repo.save() es invocado exactamente una vez.
Valores probados:
Usuario con:

- email = "user@gmail.com"
- password = "Pass1234"

Spy de UsuarioService:

- ValidarRegistro(u) forzado a devolver Collections.emptyList() (sin errores).

Mock de passwordEncoder.encode("Pass1234") → "encoded".

UT-USR-07 – crear() debería lanzar IllegalArgumentException cuando hay errores de validación

Método: Test6()
Resultado esperado:
Al llamar crear(u) se lanza una IllegalArgumentException.
El método repo.save(...) nunca es llamado.
Valores probados:
Usuario:

- email = "user@gmail.com"
- password = "Pass1234”

Spy de UsuarioService:

- ValidarRegistro(u) forzado a devolver List.of("Error") (existe al menos un error de validación).

Prueba del contexto de la aplicación

Clase de test: **RockyApplicationTests**
Clase bajo prueba: **RockyApplication**

UT-APP-01 – contextLoads

Método: contextLoads()
Sirve para verificar que el contexto de Spring Boot cargue correctamente.

Resultado esperado: No debe lanzar errores.
No tiene valores específicos. Simplemente ejecuta la aplicación para comprobar que Spring arranca sin fallar.

Resumen de casos límite

Área	Casos probados
Validación de perfil	nombre inválido, edad mínima, peso mínimo, estatura máxima
Registro de perfil	usuario inexistente, IMC correcto, peso nulo, racha por defecto
Registro de usuario	duplicados, contraseña débil, email vacío
Aplicación	carga del contexto

Cómo ejecutar las pruebas

Esta sección describe brevemente la forma en que las pruebas son ejecutables desde el entorno del proyecto.

Desde la terminal (Maven)

- **Ejecutar todas las pruebas:**

  mvn test

- **Ejecutar una clase específica:**

```
mvn -Dtest=RegistroDTOTest test
mvn -Dtest=RegistroServiceTest test
mvn -Dtest=UsuarioServiceTest test
```

Desde el IDE

1. Abrir la carpeta src/test/java.
2. Clic derecho → **Run 'Tests'**.
3. Para ejecutar solo una clase → clic derecho → **Run**.
4. También se puede ejecutar un solo método.

Qué se ve al ejecutar

- Verde: todos los tests pasaron
- Rojo: algún test falló
- Se muestran mensajes, errores y el test exacto que falló

No se necesita configuración adicional

Spring Boot ya incluye todo lo necesario (JUnit, Mockito, estructura de pruebas), así que las pruebas funcionan sin pasos extra.