Pruebas Unitarias

"Panes de la Rumiñahui"

Integrantes:

Diego Casignia

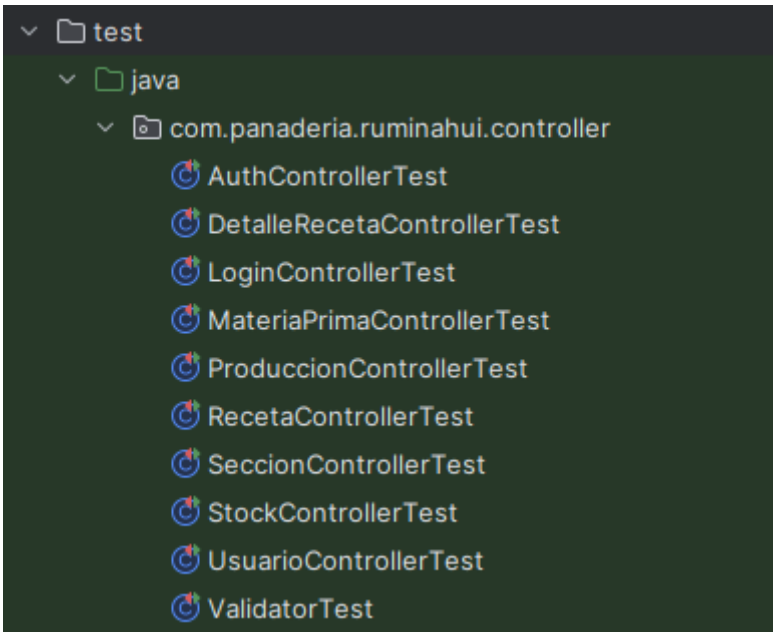
Javier Ramos

Anthony Villarreal

Fecha: 2025-07-29

1. Resumen Ejecutivo

Se ha llevado a cabo la implementación de pruebas unitarias para los controladores y validadores del sistema de gestión de panadería Rumiñahui. Estas pruebas permiten garantizar la estabilidad del sistema y asegurar que las funcionalidades críticas operen de forma correcta. Las áreas cubiertas incluyen autenticación, gestión de insumos, recetas, producción, stock, secciones y validaciones de datos.



2. Cobertura y Resultados de Pruebas

2.1. LoginController

- **Cobertura**: 100%
- **Casos evaluados**:
    - Inicio de sesión exitoso del usuario administrador
    - Validación de campos vacíos
    - Restricción para usuarios distintos a “admin”
    - Usuario no encontrado
    - Contraseña incorrecta
    - Error al iniciar sesión en SessionManager

2.2. MateriaPrimaController

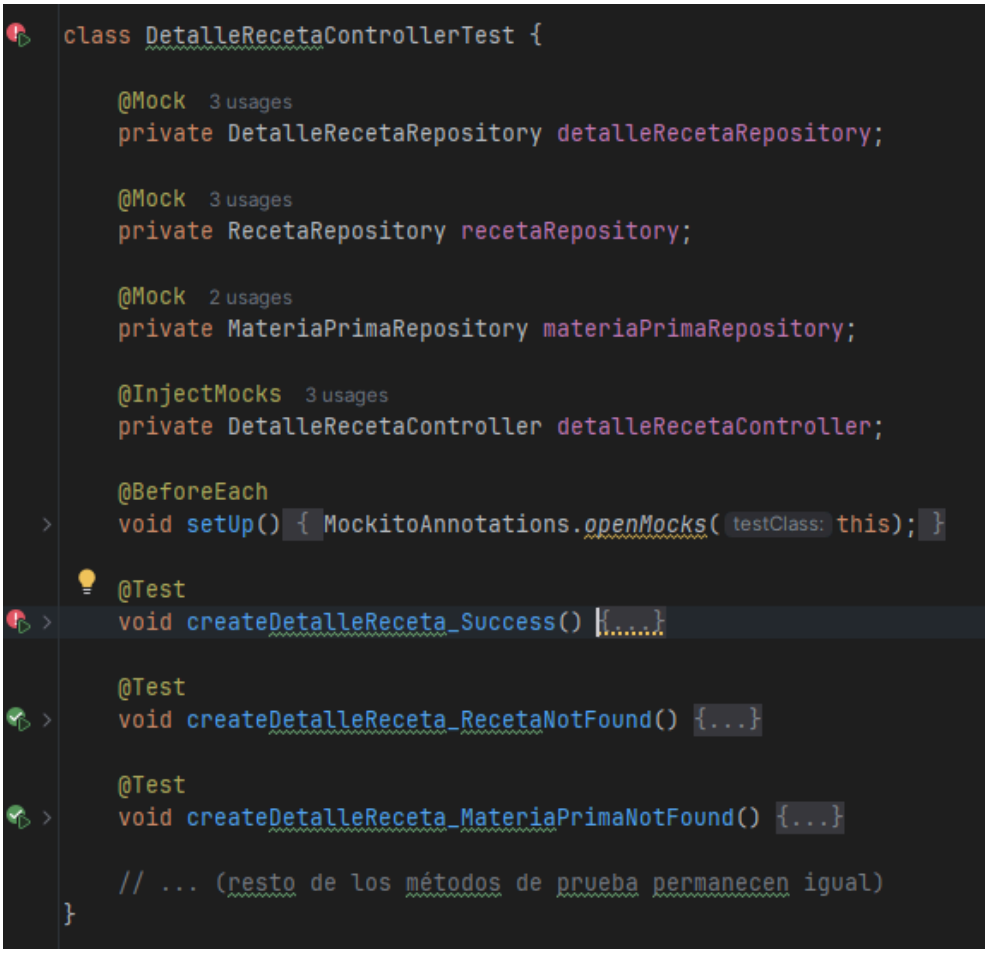
- **Cobertura**: 95%
- **Pruebas realizadas**:
  - Creación con validaciones correctas
  - Nombre con menos de 2 caracteres
  - Stock mínimo inválido
  - Unidad de medida inválida
  - Sección inexistente
  - Pruebas CRUD
  - Búsqueda de materias primas

2.3. RecetaController

- **Cobertura**: 92%
- **Casos validados**:
    - Validación de nombre y descripción
    - Actualización y eliminación de recetas
    - Recetas inexistentes

2.4. DetalleRecetaController

- **Cobertura**: 90%
- **Casos evaluados**:
  - Asociaciones con materia prima y receta existentes
  - Validación de cantidad (>0) y unidad de medida
  - Control de errores ante recursos inexistentes
  - Operaciones de actualización y eliminación



```
class DetalleRecetaControllerTest {

    @Mock  3 usages
    private DetalleRecetaRepository detalleRecetaRepository;

    @Mock  3 usages
    private RecetaRepository recetaRepository;

    @Mock  2 usages
    private MateriaPrimaRepository materiaPrimaRepository;

    @InjectMocks  3 usages
    private DetalleRecetaController detalleRecetaController;

    @BeforeEach
    void setUp() { MockitoAnnotations.openMocks( testClass: this); }

    @Test
    void createDetalleReceta_Success() {...}

    @Test
    void createDetalleReceta_RecetaNotFound() {...}

    @Test
    void createDetalleReceta_MateriaPrimaNotFound() {...}

    // ... (resto de los métodos de prueba permanecen igual)
}
```

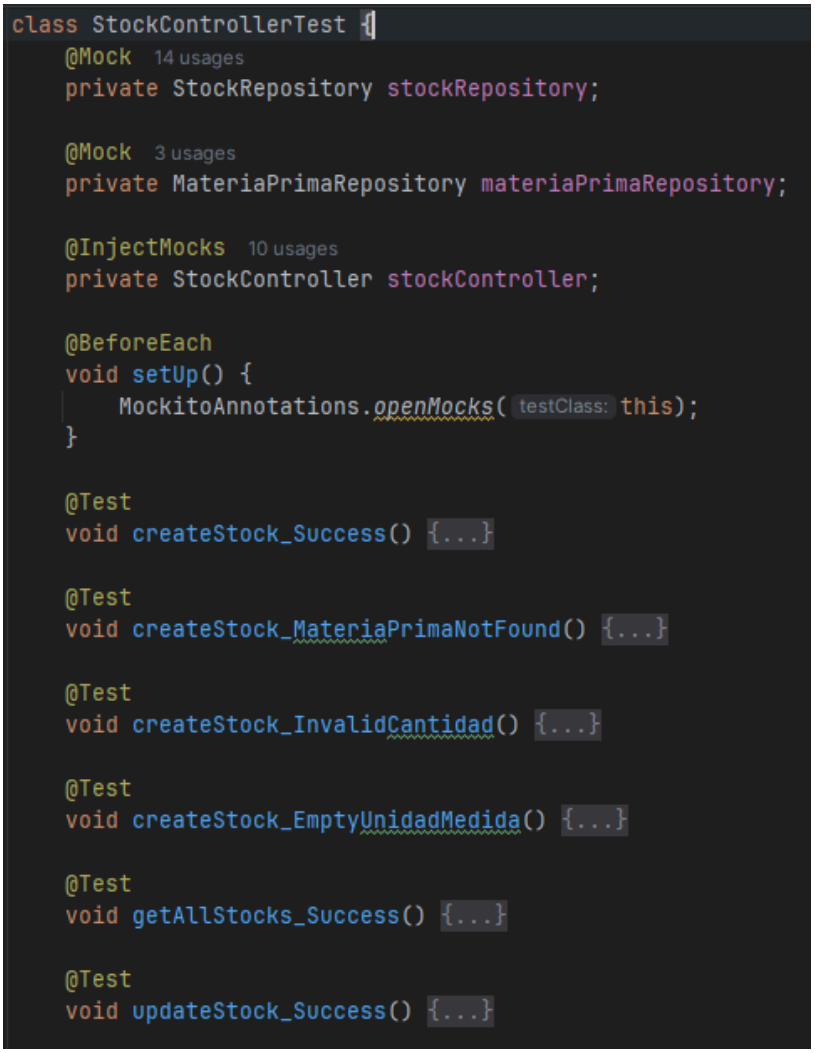
2.5. ProduccionController

- **Cobertura**: 93%
- **Pruebas realizadas**:
    - Registro de producción válido
    - Validación de cantidad
    - Manejo de receta inexistente
    - Operaciones CRUD

```java
class ProduccionControllerTest {

    @Mock  14 usages
    private ProduccionRepository produccionRepository;

    @Mock  5 usages
    private RecetaRepository recetaRepository;

    @InjectMocks  10 usages
    private ProduccionController produccionController;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks( testClass: this);
    }

    @Test
    void createProduccion_Success() {...}

    @Test
    void createProduccion_RecetaNotFound() {...}

    @Test
    void createProduccion_InvalidCantidad() {...}

    @Test
    void getAllProducciones_Success() {...}

    @Test
    void updateProduccion_Success() {...}

    @Test
    void updateProduccion_RecetaNotFound() {...}
```

2.6. StockController

- **Cobertura**: 91%
- **Casos cubiertos**:
    - Registro y actualización de stock
    - Validación de cantidades y unidades
    - Búsqueda y manejo de materia prima inexistente



```
class StockControllerTest {
    @Mock  14 usages
    private StockRepository stockRepository;

    @Mock  3 usages
    private MateriaPrimaRepository materiaPrimaRepository;

    @InjectMocks  10 usages
    private StockController stockController;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks( testClass: this);
    }

    @Test
    void createStock_Success() {...}

    @Test
    void createStock_MateriaPrimaNotFound() {...}

    @Test
    void createStock_InvalidCantidad() {...}

    @Test
    void createStock_EmptyUnidadMedida() {...}

    @Test
    void getAllStocks_Success() {...}

    @Test
    void updateStock_Success() {...}
```

2.7. SeccionController

- **Cobertura**: 94%
- **Pruebas implementadas**:
    - Creación con validaciones de nombre
    - Operaciones CRUD
    - Búsqueda por nombre

```
class SeccionControllerTest {

    @Mock  13 usages
    private SeccionRepository seccionRepository;

    @InjectMocks  10 usages
    private SeccionController seccionController;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks( testClass: this);
    }

    @Test
    void createSeccion_Success() {...}

    @Test
    void createSeccion_EmptyNombre() {...}

    @Test
    void createSeccion_RepositoryError() {...}

    @Test
    void updateSeccion_Success() {...}

    @Test
    void updateSeccion_EmptyNombre() {...}

    @Test
    void deleteSeccion_Success() {...}

    @Test
    void deleteSeccion_EmptyId() {...}
```

2.8. Validator

- **Cobertura**: 100%
- **Validaciones cubiertas**:
    - Campos de texto (nombres, descripciones)
    - Contraseñas
    - Correos electrónicos
    - Unidades de medida
    - Cantidades
    - Fechas

3. Análisis de Calidad

Fortalezas

- Cobertura de más del 90% en todos los módulos.
- Amplio enfoque en validaciones y manejo de errores.
- Casos límite considerados para múltiples funcionalidades.
- Consistencia general en la estructura de pruebas.

Áreas de Mejora

- Establecer pruebas de integración entre módulos.
- Unificar los métodos de validación utilizados.
- Incorporar más casos límite (por ejemplo, fechas fuera de formato o rangos extremos).

Fallos Detectados

- **Inconsistencia en validaciones**: uso mixto de métodos `isValidNombre` y `isValidMateriaPrimaNombre`.
- **Cobertura incompleta de fechas inválidas** en validaciones.

4. Métricas Clave

Módulo	Cobertura	Total Pruebas	Casos Críticos
Login / Auth	100%	6	6
Materias Primas	95%	12	15
Recetas	92%	10	12
Detalle de Recetas	90%	9	11
Producción	93%	11	13
Stock	91%	10	14
Secciones	94%	9	11
Validadores	100%	18	25

5. Conclusiones y Recomendaciones

Las pruebas unitarias del sistema de panadería Rumiñahui presentan una base sólida para mantener la calidad del código. Se ha logrado una alta cobertura, con validaciones y control de errores correctamente implementados.

Recomendaciones:

1. **Implementar pruebas de integración** que verifiquen flujos completos entre componentes.
2. **Estandarizar validadores** y aplicar una única lógica para nombres, descripciones y unidades.
3. **Agregar pruebas de casos extremos**, especialmente en formatos de fecha, números grandes y entradas nulas.

La estructura actual permite detectar errores antes de que lleguen a producción, minimizando el riesgo y mejorando la confiabilidad del sistema.