Tabla de Contenido

1 Introducción

1.1 Objetivo del documento

Validar que cada unidad (función, clase o método) del sistema SmarTraffic cumpla su especificación antes de su integración, detectando defectos lo más temprano posible y cumpliendo con los criterios de salida TSP.

1.2 Alcance

Las pruebas unitarias cubrirán las unidades de lógica de negocio de los servicios, verificando que apliquen correctamente las reglas, realicen transformaciones de datos y manejen errores y condiciones límite. Para aislar la lógica, las dependencias externas se sustituirán por dobles de prueba en las respectivas clases de prueba del backend a través de los mocks, objetos simulados que imitan y clonan el comportamiento de un objeto real.

En los repositorios, se validará la lógica de acceso a datos definida en los métodos, comprobando que los filtros, rangos, ordenamientos y agregaciones produzcan los resultados esperados. Estas pruebas se ejecutarán con contexto en memoria o mediante mocks, sin manejar persistencia evitando conectarse a una base de datos real.

También se incluirán utilidades como validadores, conversores y funciones puras, verificando su comportamiento con entradas válidas e inválidas.

Quedan fuera de alcance de las unitarias la lógica propia de controladores web, la integración HTTP, la seguridad aplicada de extremo a extremo y el acceso a infraestructura real; estos aspectos se cubrirán en pruebas de integración y funcionales. Para unitarias se emplearán datos en memoria o simulados, evitando I/O real.

Entorno y herramientas: JUnit 5, Mockito y Spring Boot Test para inicializar solo lo mínimo.

Criterios de entrada: código que compila, métodos implementados y mocks configurados.

Criterios de salida: al menos 90% de cobertura de los métodos en las clases de prueba que están testeando las clases críticas y todos los casos aprobados o con plan de corrección registrado.

2 Referencias

2.1 Requerimientos funcionales aplicables

En esta sección enlaza cada caso de prueba unitario con los requerimientos funcionales del sistema. El objetivo es tener un cobertura y trazabilidad de que el caso de uso en realidad fue aprobado.

Listado de RF:

- **RF-01:** Registrar usuario
- **RF-03:** Recomendar oficinas de trámite cercana
- **RF-05:** Actualizar datos personales
- **RF-09:** Estadística para la interfaz del rol administrador
- **RF-10:** Inscripción de usuario a un curso
- **RF-11:** Visualizar historial de pagos
- **RF-13:** Notificar al usuario cuando sus documentos de sus vehículos estén vencidos

Criterios de trazabilidad:

- **Unidades bajo prueba:**

Clases, métodos y repositorios que implementan el requerimiento.

- **Casos unitarios asociados:** Identificadores de las pruebas unitarias relacionadas (UT_###).
- **Criterios de aceptación unitarios:** Resultado esperado, efectos sobre el sistema y validación de interacciones con dependencias.
- **Evidencias:** Resultado obtenido en la ejecución y estado de la prueba.

2.2 Normas, guías y checklist de calidad del equipo

- **Normas de codificación:**

- Usar nombres claros y consistentes.
- Cada prueba debe tener un nombre descriptivo que indique qué se está probando y cuál es el resultado esperado.
- Mantener el código de las pruebas limpio y entendible.

- **Guías para las pruebas unitarias:**

- Todas las dependencias externas (como repositorios, correos o fechas) deben simularse con mocks para no depender de recursos reales.
- Las pruebas deben dar siempre el mismo resultado (deterministas), evitando datos aleatorios.
- Asegurar una cobertura mínima de 80% en general y al menos 90% en las partes críticas del sistema.

- **Checklist de calidad antes de aprobar una prueba**
- ¿La prueba tiene un nombre claro y fácil de entender?
- ¿La prueba no depende de conexiones reales (BD, red, etc.)?
- ¿Se probaron entradas válidas, inválidas y casos límite?
- ¿Se verificó el valor devuelto y las acciones realizadas por el método?
- ¿El resultado de la prueba es siempre el mismo al repetirla?
- ¿La prueba está vinculada a un requerimiento (trazabilidad)?

3 Estrategia de prueba

3.1 Nivel de automatización

Se realizan pruebas con el uso del framework JUnit/Mockito.

3.2 Técnica de diseño de pruebas (caja negra, caja blanca, valores límite, tablas de decisión, etc.).

La técnica de diseño de pruebas a usar será caja blanca con aislamiento mediante dobles de prueba (Mocks) y verificación de comportamiento con assets y verify.

3.3 Herramientas a utilizar (IDE, cobertura de código, análisis estático, CI/CD).

- Spring-Security-Test
- Spring-Boot-Starter-Test

- Mockmvc
- Mockito
- JUnit5
- Visual Studio Code

4 Casos de prueba unitarios

4.1 Id del caso de prueba : Identificador de la prueba Use Case Test (ej. UCT-01)

4.2 Nombre/Descripción: Nombre del caso de uso junto con la descripción del resultado, separada por un guión bajo (ej. PaymentsHistory_ok)

4.3 Precondiciones: *Información necesaria preexistente para llevar a cabo la ejecución del método (ej. usuario registrado y JWT autenticado).*

4.4 Datos de entrada

Parámetros concretos que se pasan al método (valores literales), o el cuerpo/DTO si aplica.
Ej: email="no@exa.co", currentPassword="X", newPassword="Y".

4.5 Acción a ejecutar

La llamada exacta a la unidad bajo prueba (UUT): método y cómo se invoca.
Ej: updatePassword(email, currentPassword, newPassword).

4.6 Resultado esperado

Qué debe ocurrir si la unidad funciona bien, en términos observables y medibles:

- Valor de retorno (exacto o patrón).
- Efectos colaterales (por ejemplo, “se invoca save() una vez”, “no envía correo”).
- Cambios de estado (propiedades del objeto).

- Llamadas a dependencias (interacciones con mocks, conteo de invocaciones).

4.7 Resultado obtenido

Qué debe ocurrir si la unidad funciona bien, en términos observables y medibles, lo cual incluye

- El valor de retorno real o la excepción con su mensaje
- Métodos y efectos colaterales ejecutados, por ejemplo, evidenciar como se llama el método createPayment() a la hora de crear un pago
- Evidenciar en los artefactos la ejecución del intento, como logs de consola y registros en la base de datos

Adicionalmente, debe de retornar la respuesta de la request, por ejemplo, un retorno de 200 OK, un body con información relevante, por ejemplo “count” : 5. También deben de tenerse en cuenta las excepciones previstas en los métodos y, opcionalmente, detallar el tiempo de ejecución del intento.

4.8 Estado

Refiere al resultado obtenido del caso con respecto al resultado esperado, que puede tomar alguno de los siguientes valores:

- PASSED: El resultado obtenido coincide con el valor esperado
- FAILED: El resultado obtenido difiere con el valor esperado, incluye escenarios con interacciones con dependencia
- BLOCKED: No logró ejecutarse debido a falla de una dependencia externa, por ejemplo, se cae la base de datos
- SKIPPED: Omitido por no cumplir una determinada condición

Por ende, aquellas pruebas que no retornen el valor esperado, en caso de diferir, serán categorizadas como FAILED y si no son ejecutadas por causas de terceros, serán categorizadas como BLOCKED o SKIPPED según el caso

4.9 Trazabilidad al requerimiento

Hace referencia a la vinculación de la prueba con artefactos diseñados para la gestión del código, lo cual incluye:

- Id del caso de uso, por ejemplo, UC-001
- Módulo / clase / método que está bajo prueba
- Defecto relacionado en caso de fallar
- Riesgo TSP relacionado con el defecto

Dentro de los siguientes documentos está la información, parámetros y métricas asociadas con la trazabilidad al requerimiento:

https://docs.google.com/document/d/1IR4QDJqRQpQk9fMgjNCGO7NejML4QqsU/edit

https://docs.google.com/document/d/1D5v8n8zMt3f7pHdFDjL5llSOJ0gJJj7Ovnx7mcvNoYE/edit?tab=t.0

5 Gestión de defectos

5.1 Procedimiento de registro (Descripción del defecto, link y número de registro en el log de defectos)

Frente al registro de un determinado defecto, debe tenerse en cuenta las siguientes consideraciones para diligenciar de manera correcta el formato establecido:

- Fecha del defecto
- Número ID del defecto
- Tipo de defecto obtenido
- Fase de inyección
- Fase de remoción (campo que queda vacío hasta que se soluciona el defecto)
- Tiempo de resolución
- Fecha de resolución del defecto
- Descripción que resuma brevemente el error y el motivo de su resultado

Dentro del marco TSP, se decidió manejar 2 archivos de logs de defectos, uno dedicado netamente para la codificación y el otro que incluya todo lo relativo a la documentación y estructuración de planes

para las fases del proyecto. A continuación, se dejan los links de dichos archivos de defectos:
https://docs.google.com/document/d/1eVk9znKXPidGaisoFvA3CPEFOlnDj-Z8/edit
https://docs.google.com/document/d/1dI-QJdcO3SsoNgTyXmMH5Yyi42_jgk8m/edit

6 Métricas y seguimiento

Fase de Inyección	% de casos ejecutados vs planificados	% de éxito	Cobertura de código alcanzada	Defectos encontrados por fase
Iniciación	100%	90%	N/A	20
Estrategia	95%	88%	N/A	9
Requerimientos	95%	85%	N/A	11
Planeación	90%	85%	N/A	4
Diseño	90%	80%	N/A	6
Implementación (Backend)	100%	100%	10764	4
Implementación (Backend)	100%	100%	6359	3

7 Control de Cambios

CONTROL DE CAMBIOS		
Fecha	**Descripción**	**Autor(es)**
17/09/2025	Versión inicial del plan	Felipe Triviño