Tabla de Contenido

1 Generalidades del documento

1.1 Objetivo del documento

El siguiente documento tiene como fin definir los distintos estándares que se llevarán a cabo para la fase de planeación siguiendo la metodología TSP. Estos estándares están definidos en el alcance del documento junto con las revisiones e inspecciones de código, todo con el fin de desarrollar un producto que se asegure que los principales atributos de calidad discutidos en documentos previos (seguridad, mantenimiento, mantenibilidad, etc) se cumplan, manteniendo la trazabilidad con los requerimientos elaborados y de la misma forma, habilitando inspecciones en el marco TSP para prevenir defectos de manera temprana.

1.2 Acrónimos

TIC: Tecnologías de la Información y la Comunicación
API: Interfaz de programación de Aplicaciones
HTML: Lenguaje de Marcado de Hipertexto
SQL: Lenguaje de Consulta Estructurada
DNS: Sistema de Nombres de Dominio
SSL/TLS: Capa de Sockets Seguros / Transport Layer Security
JSON: JavaScript Object Notation
JWT: JSON Web Token
HTTP: Hypertext Transfer Protocol
CRUD: Create, Read, Update and Delete.
ID: Identificador

2 Introducción

El presente Plan de Implementación describe las actividades, directrices y mecanismos necesarios para llevar a cabo la fase de implementación del proyecto bajo el marco de la metodología TSP. En esta fase el propósito es asegurar que los componentes desarrollados sean integrados de manera controlada, siguiendo estándares previamente definidos en documentos del proyecto para así garantizar la alineación con los requerimientos funcionales del sistema.

El plan establece las pautas para la codificación, las pruebas unitarias, revisiones técnicas e inspecciones de código, con el

objetivo de garantizar y asegurar la calidad del producto, prevenir defectos desde etapas tempranas y mantener la trazabilidad frente a los requerimientos y artefactos elaborados en fases previas del proyecto. De la misma manera, este documento va a servir como guía de referencia para todos los miembros del equipo, definiendo recursos y mecanismos de control para que la implementación sea exitosa.

2.1 Alcance

El alcance del presente documento abarca:

- Definir los estándares de codificación, nombramiento y documentación a aplicar durante la fase de implementación.
- Establecer los procedimientos de integración de los módulos y componentes desarrollados, asegurando su compatibilidad y correcto funcionamiento en la arquitectura definida para el proyecto
- Definir las actividades tanto de inspección como de revisión de código en el marco TSP, para así prevenir y corregir defectos antes de su propagación
- Establecer los lineamientos para la gestión de la configuración y el control de versiones

Se aclara que en este documento no se incluye la definición de requerimientos, el diseño arquitectónico ni la gestión de riesgos, debido a que estas actividades corresponden a documentos que fueron entregados en fases anteriores del proyecto.

2.2 Restricciones

El presente documento se encuentra sujeto a las siguientes restricciones:

- **Tiempo:** Las actividades deben de realizarse en el cronograma aprobado para no afectar las entregas comprometidas. Sucesos inesperados que intervengan en este aspecto son contemplados
- **Recursos:** El equipo debe de ajustarse a las herramientas, entornos de desarrollo y recursos tecnológicos definidos previamente, para así evitar incorporar nuevas tecnologías no aprobadas por la gerencia del proyecto

- **Alcance funcional:** Los cambios que serán implementados serán en base a requerimientos aprobados y priorizados en la fase, cualquier cambio deberá seguir la correcta trazabilidad para su elaboración e implementación
- **Metodología:** La elaboración de la fase y sus correspondientes actividades deben estar alineadas con el marco TSP, cumpliendo con sus roles, inspecciones y demás actividades que demande esta tecnología

3 Estándares de nombramiento y documentación

Estándares de nombramiento

Archivos y carpetas

- Nombres cortos y descriptivos.
- Uso de camelCase
- Ejemplo: controllerTest

Versionado

- Se incluye el número de la versión del documento y la fecha en la que fue creado el documento.
- Ejemplo: 1.0Nombre_Fecha

Identificadores de Componentes

- Seguir un patrón homogéneo según el archivo y la carpeta a la que corresponda(uso de sufijos por grupo de módulos agrupados en una carpeta)
- Ejemplo: serviceController

Roles y responsables

- Usar convenciones para identificar actores responsables.
- Manejar las mismas variables a nivel de software como a nivel de documentación.

- Ejemplo: ROL_ADMIN,ROL_DEV

Estándares de documentación

- **Estructura Uniforme:** Secciones mínimas que deben incluir los documentos (objetivo, alcance, responsables, control de cambios)

- **Formato y estilo:** Párrafos con estilo Arial 12 y títulos con negrilla

- **Lenguaje:** Evitar tecnicismos innecesarios o en caso de ser necesarios, realizar un apartado nuevo y explicarlos en un glosario.

Control de cambios

- Cada documento debe tener una versión, fecha , autor y descripción del cambio
- Ejemplo:

CONTROL DE CAMBIOS		
Fecha	**Descripción**	**Autor(es)**
30/09/2025	Corrección en Sección de estándares de codificación	Felipe Triviño
11/11/2025	Revisión y corrección en Sección de estándares de nombramiento	Julián Nova

4 Estándares de codificación

4.1 Guías de estilo

CSS/Javascript/React:
Nombres para los componentes en camelCase y en css snake_case evitar el uso de !important.

Java / Spring :
Nombres de clases en PascalCase, métodos con camelCase, constantes UPPER SNAKE.

SQL:

mayúsculas para palabras clave (SELECT,FROM,WHERE,ORDER BY), snake_case en tablas y columnas

4.2 Estructura y Convenciones

Front: estructura de proyecto src/app con carpetas (roles,components,styles), layout por arquitectura es la raíz del proyecto de la cual hacen parte las pages, usar favico.ico para el uso de iconos, globals.css para estilos generales

Back: paquetes por cada capa por ejemplo controller,service,repository,config,model (entre otros), cada clase debe tener en su nombre el servicio que realiza dentro de la aplicación (PaymentsService, StatisticsController).

4.3 Manejo de Errores y Excepciones

Front: React Error Boundary y toasts que contribuyan al flujo del proceso.

Back: excepciones mapeadas a códigos HTTP con mensajes sin datos sensibles para el usuario. Se recomienda usar ResponseEntity<?> para retornar cualquier tipo de objeto.

4.4 Seguridad por defecto

Front: uso de middleware con autenticación JWT.

Back: uso de JSON Web Token,BCrypt,Spring Security, validación usando Validation Beans (@Notnull,@Email), proteger de SQL injection y protección de endpoints.

4.6 Pruebas Mínimas obligatorias

- JUnit/Mockito con cobertura a la capa de servicios

4.7 Commits,Ramas y PRs

Palabras clave para commits : add:, fix: ,refactor: ,test: ,docs:, entre otros.
Ramas :
Frontend: Main,Develop
Backend: Main,Backup,Julian,Felipe,Nicolas.

Checklist para Pull Request:

Pasa Pruebas
Docs Actualizados
Sin datos sensibles expuestos
Integrado sin inyectar defectos
Revisión de Seguridad (opcional)

5 Revisión de diseño

La revisión del diseño tiene como objetivo asegurar que la arquitectura y los componentes definidos cumplen con los requisitos funcionales y no funcionales del sistema SmartTraffic, así como con los estándares de calidad establecidos en el proyecto. Esta actividad busca identificar inconsistencias, ambigüedades o posibles riesgos técnicos antes de la implementación.

1. Objetivos de la revisión

- Verificar la consistencia entre el diseño y los requerimientos del sistema.
- Garantizar que la arquitectura seleccionada soporte las restricciones de seguridad, escalabilidad y desempeño del proyecto.
- Detectar y corregir defectos de diseño de forma temprana para minimizar costos de corrección en fases posteriores.
- Validar que el diseño cumple con los estándares de codificación, nomenclatura y documentación definidos en el plan de calidad.
- Asegurar la trazabilidad entre requisitos, componentes, módulos, clases.

2. Alcance

La revisión se aplica a todos los artefactos de diseño producidos durante la fase de diseño detallado:

- Diagramas de clases y paquetes.
- Modelo de datos (entidades, relaciones, restricciones).
- Diagramas de despliegue y componentes.
- Interfaces de usuario y flujos de navegación.
- Documentación de las decisiones de arquitectura.

3. Criterios de revisión

Para evaluar cada artefacto se utilizarán los siguientes criterios:

- Corrección: el diseño debe reflejar fielmente los requisitos del usuario.
- Completitud: todos los requisitos deben estar cubiertos en el diseño.
- Consistencia: no deben existir contradicciones entre artefactos ni duplicidades en los modelos.
- Simplicidad: el diseño debe ser entendible, evitando complejidad innecesaria.
- Modularidad y cohesión: cada módulo debe tener responsabilidades bien definidas.
- Acoplamiento mínimo: las dependencias entre módulos deben ser claras y reducidas.
- Cumplimiento de estándares: debe seguir las guías de documentación, nomenclatura y codificación del proyecto.

4. Proceso de revisión

La revisión del diseño seguirá la siguiente metodología:

1. **Planificación de la revisión:** se agenda la reunión con el equipo de desarrollo.
2. **Preparación individual:** cada miembro del equipo analiza los artefactos asignados usando listas de chequeo.
3. **Revisión grupal:** se exponen los hallazgos en sesión conjunta, se discuten inconsistencias y se documentan defectos.
4. **Registro de defectos:** los hallazgos se documentan en el portal TSP en el formato de Log de Defectos.
5. **Corrección:** los responsables de cada módulo ajustan el diseño y suben la versión actualizada al repositorio.
6. **Verificación de correcciones:** el Líder de Calidad valida que los defectos hayan sido resueltos.

5. Roles y responsabilidades

- Líder de Calidad: Coordina la revisión y consolida el registro de defectos.
- Líder de Desarrollo: Asegura la coherencia técnica de las decisiones.
- Equipo de Desarrollo: Revisa diagramas y modelos de manera individual y colectiva.
- Líder de Planeación: Valida que el diseño soporte las restricciones de tiempo y recursos del proyecto.
- Líder de Planeación: Coordina, segura y procura el buen funcionamiento de cada uno de los roles, asegurando que cada integrante cumpla con sus responsabilidades como rol
- Líder de Soporte: Actualiza los documentos dentro del portal TSP.

6. Productos esperados

- Lista consolidada de defectos de diseño detectados.
- Registro de correcciones realizadas.
- Versión actualizada y validada de los artefactos de diseño en el repositorio del proyecto.

6 Estándares de implementación

Con el fin de garantizar uniformidad, trazabilidad y calidad en la fase de implementación, se establecen los siguientes estándares:

1. **Gestión de dependencias**
    - Para el desarrollo en Java se utilizará Maven como gestor de dependencias.
    - Para el desarrollo en JavaScript se emplea npm como gestor de paquetes.
    - Toda dependencia deberá estar documentada con el estándar de control de versiones, justificando su uso y registrando la versión aprobada.
2. **Entorno de ejecución**
    - Versión de Java: 21.0.0.
    - Framework backend: Spring Boot.
    - Base de datos: PostgreSQL 9.0.

- Entorno frontend: Node.js v22.16.0 con npm 10.9.2.
- IDE oficial de desarrollo: Visual Studio Code, con configuración estandarizada para todos los miembros del equipo.

3. **Control de versiones**
   - Se trabajará con dos repositorios independientes: backend y frontend.
   - Cada repositorio contará con dos ramas principales:
     - main: rama estable, destinada únicamente a versiones en producción.
     - develop: rama de integración de funcionalidades en desarrollo.
   - Los mensajes de *commit* deberán seguir la convención definida:
     - add: incorporación de archivos, recursos nuevos y/o dependencias.
     - commit: confirmación de cambios generales que no encajan en otra categoría.
     - feat: implementación de una nueva funcionalidad.
     - fix: corrección de errores o defectos.
     - docs: adición o modificación de documentación.
   - Todo *pull request* deberá ser revisado y aprobado por al menos un miembro antes de su integración.
4. **Pruebas e integración continua**
   - Todo módulo deberá incluir pruebas unitarias y de integración con una cobertura mínima del 80%.
   - Las pruebas deberán ejecutarse en localhost y una vez incorporado con la rama develop teniendo un índice de errores menor al 5%, se podrá hacer la fusión de código en la rama main.
5. **Configuración y gestión de secretos**
   - La configuración sensible (credenciales, llaves, tokens, etc.) se gestionará mediante archivos .env con encriptación o utilizando gestores de secretos.
   - Queda estrictamente prohibido exponer información sensible en mensajes de error o registros de log.
6. **Despliegue**
   - El despliegue en ambiente de desarrollo se realizará en localhost para el frontend y backend.
   - Se aplicará versionado semántico del producto (ejemplo: v1.0.0) para identificar de forma clara las actualizaciones y

compatibilidad de versiones en la documentación al inicio de los archivos de implementación (.java, .tsx, .css y .env).

7. **Seguridad en la implementación**
    - Todo flujo de autenticación y autorización se realizará mediante JWT o un estándar equivalente definido por el equipo.

7 Revisiones e inspecciones de código

Frente al desarrollo de las revisiones e inspecciones de código, hay que dividir sus distintos componentes de la siguiente manera para una óptima elaboración.

- Roles y responsabilidades:

  Divididos de la siguiente manera:

  a) Autor del código, quien se encarga de la codificación de la/s clase/s y asegura que cumpla con sus estándares de funcionalidad.
  b) Moderador de la revisión, quien se encarga de dirigir la inspección, mantener el enfoque en la calidad del producto y garantizar que se cumpla el proceso
  c) Revisores, quienes analizan el código en busca de defectos, malas prácticas y errores en la funcionalidad
  d) Registrador, quien se encarga de documentar los hallazgos y documentarlos en el log de defectos

  Cabe aclarar que en estas divisiones, los miembros que participen pueden elaborar 1 o más de estos roles, esto frente a la consideración del número de miembros del equipo.

- Procedimiento

  Inicialmente, en la preparación del código, el autor sube su código a la rama que le corresponda y hace la respectiva documentación. Esto se hace después de comprobar que el código compile sin errores y tenga una clase con pruebas unitarias que respalde su veracidad.

Luego, en la revisión, quienes ejecuten el rol de revisor se encargan de analizar el código de manera independiente y evaluando su calidad verificando la funcionalidad operacional y las clases de prueba.

En caso de encontrar errores y bugs, se discuten las razones de los mismos y se registran y clasifican según su severidad, posteriormente se asignan los responsables y los tiempos de corrección estimados. Todas estas consideraciones deben ser registradas en el log de defectos del proyecto y para finalizar, se valida que los cambios implementados cumplan los estándares antes de la integración.

Dentro de las herramientas de control y de apoyo, está GitHub para el control de versiones, se implementarán Pull Request para unificar y subir versiones funcionales del producto y el log de defectos para registrar todos los errores que puedan surgir en el proceso.

- Métricas de Calidad

Las inspecciones del código generarán datos que se van a integrar con respecto a las directrices plasmadas en el documento del plan de calidad del proyecto, donde se encuentran consideraciones como:

a) Número de defectos detectados por fase
b) Densidad de defectos por KLOC
c) Porcentaje de defectos removidos frente a pruebas

El siguiente link redirige al plan previamente mencionado, elaborado en la fase de planeación:

https://docs.google.com/document/d/1D5v8n8zMt3f7pHdFDjL5llSOJ0gJJj7Ovnx7mcvNoYE/edit?usp=sharing

La aprobación de un componente se hará efectiva cuando:

- Todos los defectos hayan sido corregidos
- El código cumpla con los estándares de codificación del proyecto
- Se haya registrado y cerrado los registros del log de defectos, asegurando la trazabilidad entre los cambios llevados a cabo para sus respectivas correcciones

Frente al cumplimiento de la revisión e inspección de código se elaboró el siguiente checklist:

- **Roles y responsabilidades**
  - ☐ ¿Se designó un Moderador para dirigir la revisión y mantener el enfoque en calidad?
  - ☐ ¿Se nombraron Revisores responsables de analizar defectos, malas prácticas y errores de funcionalidad?
  - ☐ ¿El autor/autores del código documentaron correctamente hallazgos de errores y bugs en el log de defectos?
  - ☐ ¿En el equipo de trabajo, quedó claro que un miembro puede asumir más de un rol dependiendo del compromiso y el tamaño del equipo?
- **Procedimiento**
  - ☐ El autor del código subió su versión del código en su rama correspondiente en el repositorio de GitHub
  - ☐ La versión del código subido compila sin errores antes de su revisión
  - ☐ Las responsabilidades repartidas de codificación cuentan con sus clases y pruebas unitarias asociadas
  - ☐ Se hizo la verificación de la funcionalidad operacional y de las clases de prueba
  - ☐ Los defectos encontrados fueron registrados en el log de defectos
  - ☐ Se asignaron responsables y tiempos estimados para la elaboración de las labores de codificación
- **Herramientas de control y apoyo**
  - Se utilizó GitHub para llevar el control de las versiones
  - Para unificar las versiones y subir la versión funcional se creó un Pull Request para unificar las ramas
  - Los conflictos resultantes del Merge de distintas ramas fueron resueltos
- **Métricas de calidad**

- ☐ Se tiene el dato del número de defectos detectados en la fase
- ☐ Se midió el porcentaje de defectos removidos antes de implementar las pruebas

- **Seguimiento posterior a la revisión**
  - ☐ El estado de los defectos del log quedaron registrados de manera correcta
  - ☐ El responsable del defecto confirmó su resolución y pusheo su versión del código corregida
  - ☐ Se realizó una reunión para la próxima sesión de seguimiento

8 Control de Cambios

CONTROL DE CAMBIOS		
Fecha	**Descripción**	**Autor(es)**