Manual Técnico y Especificación de requisitos de software

Proyecto: Sistema de Optimización de Rutas de Transporte

Integrantes : Andres Felipe Carrasquilla Gutierrez
Yohan Stevens Piñarte Diaz
Jhon Franklin Sandoval Segura

1. Introducción

La presente Especificación de Requisitos de Software (SRS) proporciona una visión general completa del sistema de optimización de rutas de transporte desarrollado para la Universidad de los Llanos. Este documento describe de manera estructurada el propósito del sistema, su alcance funcional, las definiciones y acrónimos relevantes, las referencias utilizadas y una vista general de los contenidos incluidos en esta SRS.

1.1. Propósito

- El objetivo de este documento es establecer de manera formal y detallada los requisitos del **Sistema de Optimización de Rutas de Transporte**, diseñado para mejorar la eficiencia del desplazamiento institucional mediante el uso de **algoritmos genéticos y metaheurísticas**.

  Este manual técnico también documenta el proceso completo de desarrollo del sistema, garantizando la **trazabilidad desde los requisitos iniciales hasta la implementación final**, con el fin de facilitar su mantenimiento, evolución y evaluación posterior

1.2. Alcance

El sistema cubre las funcionalidades necesarias para la planificación y optimización de rutas de transporte dentro del contexto operativo de la Universidad de los Llanos. Incluye:

- Gestión y registro de puntos de recogida y destinos.
- Procesamiento de datos mediante algoritmos genéticos y metaheurísticos.
- Generación de rutas óptimas basadas en criterios como distancia, tiempo y capacidad.
- Interfaz para visualización y selección de rutas recomendadas.
- Módulos para administración, consulta y configuración del sistema.

1.3. Definiciones, acrónimos y abreviaturas

GA: Algoritmo Genético (Genetic Algorithm)

SA: Recocido Simulado (Simulated Annealing)

TSP: Problema del Viajante (Traveling Salesman Problem)

VRP: Problema de Enrutamiento de Vehículos (Vehicle Routing Problem)

API: Interfaz de Programación de Aplicaciones

HTTP: Protocolo de Transferencia de Hipertexto

JSON: Notación de Objetos de JavaScript

OSRM: Open Source Routing Machine

1.4. Referencias

Referencia	Título	Ruta	Fecha consulta	Autor
1	Documentación FastAPI 0.104.1	https://fastapi.tiangolo.com	20-11-2025	Tiangolo
2	Fundamentos de algoritmos genéticos	https://www.escom.ipn.mx/docs/oferta/matDidacticoISC2009/AGntcs/apuntesAlgsGeneticos.pdf	20-11-2025	Escuela Superior de Cómputo
3	Dataset Geográfico Rutas	https://www.unillanos.edu.co/index.php/noticias-imagenes/6577-las-rutas-unillanos-estan-a-tu-disposicion-este-es-el-horario	20-11-2025	Universidad de los Llanos

1.5. Resumen

- Este documento presenta la Especificación de Requisitos de Software (SRS) del Sistema de Optimización de Rutas de Transporte desarrollado para la Universidad de los Llanos. En él se describen de manera estructurada los objetivos generales del sistema, su alcance funcional, las definiciones técnicas relevantes y las referencias utilizadas durante el desarrollo. Asimismo, se incluye una visión completa del producto, su contexto operativo, los tipos de usuarios, las restricciones técnicas y los supuestos bajo los cuales el sistema ha sido diseñado.

  El cuerpo central del documento detalla los requisitos específicos que definen el comportamiento esperado del sistema, incluyendo los requisitos funcionales relacionados con la gestión de paraderos, selección de origen/destino, ejecución de algoritmos genéticos, visualización geográfica y configuración de parámetros de optimización. Del mismo modo, se describen los requisitos no funcionales que garantizan un rendimiento adecuado, fiabilidad operativa, disponibilidad del servicio y mantenibilidad del software.

  Finalmente, el documento proporciona criterios de aceptación para cada funcionalidad, lineamientos técnicos para las interfaces de usuario y comunicación, así como requisitos legales y operativos necesarios para asegurar la correcta implementación, evolución y uso del sistema. La estructura presentada permite una comprensión completa del proyecto, favorece la trazabilidad entre los

requerimientos y los componentes desarrollados, y sirve como base formal para futuras mejoras e integraciones.

Este documento se organiza en secciones que describen de manera progresiva los elementos fundamentales del sistema. Tras esta introducción, se presentan:

- **Descripción general:** contexto, usuarios, restricciones, supuestos.
- **Requisitos específicos:** funcionales, no funcionales y de interfaz.
- **Modelos y diagramas:** casos de uso, arquitectura, flujo de datos.
- **Detalles técnicos de implementación:** algoritmos, módulos y componentes.
- **Trazabilidad:** relación entre requisitos, diseño y componentes desarrollados.

Esta estructura permite una comprensión integral del sistema y su proceso de desarrollo.

2. Descripción general

2.1. Perspectiva del producto

El Sistema de Optimización de Rutas es un producto independiente que funciona como herramienta de apoyo a la decisión para la planificación de transporte universitario. No forma parte de un sistema mayor actualmente, pero está diseñado para posibles integraciones futuras con sistemas de gestión universitaria.

2.2. Funcionalidad del producto

- Gestión de datos geográficos: Almacenamiento y gestión de coordenadas de paraderos
- Configuración de demanda: Especificación de número de estudiantes por paradero
- Optimización de rutas: Aplicación de tres algoritmos (Greedy, GA, SA) para encontrar rutas óptimas
- Visualización geográfica: Representación gráfica de rutas en mapas interactivos
- Cálculo de métricas: Evaluación de distancia, tiempo, capacidad y fitness de soluciones
- Gestión de restricciones: Aplicación de penalizaciones por sobrecapacidad

2.3. Características de los usuarios

Tipo de usuario	Docentes, Administrativos, Estudiantes
Formación	Profesional, En formación profesional
Habilidades	- Manejo básico de computadores - Uso de aplicaciones web - Capacidad para interpretar rutas y horarios - Conocimiento de procesos institucionales básicos
Actividades	- Consultar rutas optimizadas de transporte - Seleccionar rutas destino - Seleccionar método Metaheurístico - Capacidad de análisis de oferta y demanda de las rutas de transporte

2.4. Restricciones

- Tecnológicas: Python, FastAPI, Leaflet.js, navegadores modernos
- Temporales: Desarrollo en 4 semanas con equipo de 3 desarrolladores
- Presupuesto: Uso exclusivo de tecnologías open-source
- Rendimiento: Tiempo máximo de cálculo de 30 segundos por optimización
- Datos: Disponibilidad limitada a paraderos predefinidos

2.5. Suposiciones y dependencias

- Coordenadas precisas: Las ubicaciones de paraderos pueden ser inexactas (no se detallan las direcciones específicas en el data set de la universidad)
- Demanda estable: Número de estudiantes por paradero no varía durante la optimización
- Velocidad constante: 35 km/h promedio en todas las rutas
- Capacidad uniforme: Todos los vehículos tienen misma capacidad
- Conectividad: Servicio OSRM disponible para cálculo de rutas realistas

2.6. Evolución previsible del sistema

- Integración con sistemas de GPS en tiempo real

- Optimización multi-objetivo (costo, tiempo, emisiones)
- Gestión de flotas múltiples de vehículos
- Planificación horaria y por turnos
- Aplicación móvil para conductores

3. Fundamento Matemático

El sistema implementa un problema de optimización combinatoria basado en un Problema de Ruta o Problema del Viajante (TSP) con variaciones, utilizando técnicas heurísticas y metaheurísticas. Matemáticamente, el fundamento se divide en:

- Geometría para cálculo de distancias
- Modelado del problema (TSP-Capacitado)
- Función objetivo con restricciones
- Metaheurísticas (Greedy, GA, SA)
- Medición del rendimiento (fitness y penalización)

Cálculo de Distancias: Fórmula de Haversine

El sistema trabaja con coordenadas reales (latitud/longitud), por lo que se usa la distancia sobre la superficie de la Tierra.

La fórmula Haversine se basa en trigonometría esférica:

Distancia entre dos puntos geográficos

$$d = 2R \cdot arcsin(\sqrt{sin^2(\frac{\Delta\varphi}{2}) + cos(\varphi 1)cos(\varphi 2)sin^2(\frac{\Delta\lambda}{2})})$$

Donde:

- $\varphi 1, \varphi 2$: latitudes
- λ1,λ2 : longitudes
- R=6371 km : radio terrestre
- Δφ=φ2−φ1
- Δλ=λ2−λ1

Esto permite construir una matriz de distancias:

$D = [d_{ij}]$, d_{ij}=distancia entre paraderos i y j

```python
def haversine(lat1, lon1, lat2, lon2):
    R = 6371.0  # km
    phi1 = math.radians(lat1)
    phi2 = math.radians(lat2)
    dphi = math.radians(lat2 - lat1)
    dlambda = math.radians(lon2 - lon1)

    a = math.sin(dphi / 2) ** 2 + math.cos(phi1) * math.cos(phi2) * math.sin(dlambda / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    return R * c
```

Modelado del Problema: TSP-Capacitado

Tu sistema modela una variación del TSP:

Dado:

- Un conjunto de paraderos

$$S = \{1, 2, ..., n\}$$

- Una demanda asociada a cada paradero:

$$dem(i)$$

- La capacidad del bus:

$$C$$

- La matriz de distancias

$$D_{ij}$$

Se busca:

Un orden de visita (ruta):

$$R = (r1, r2, ..., rk)$$

que:

- Minimice la distancia total.
- No exceda la capacidad del bus.

- Cumpla el orden requerido (inicio y fin: UNILLANOS)

Función Objetivo y Penalización

La distancia total de una ruta está dada por:

$$Dist(R) = \sum_{i=1}^{k-1} D_{ri,ri+1}$$

La demanda total :

$$Dem(R) = \sum_{i=1}^{k} dem(r_i)$$

Restricción de capacidad

$$Dem(R) \leq C$$

Como es una metaheurística, las violaciones a la restricción se penalizan:

$$Penalidad(R) = \{0 \quad si\ Dem(R) \leq C\}$$

$$\{\alpha\ (Dem(R) - C)\ si\ Dem(R) > C\}$$

En la implementación adaptamos el valor a :

$$\alpha = 2$$

Función objetivo (fitness)

$$Fitness(R) = Dist(R) + Penalidad(R)$$

El sistema selecciona la ruta con **menor fitness**

```python
def evaluate_route(route: List[str], demands: Dict[str, int], capacity: int) -> Dict:
    total_distance = 0.0
    for i in range(len(route) - 1):
        a = IDX[route[i]]
        b = IDX[route[i + 1]]
        total_distance += DIST_MATRIX[a][b]

    # Demanda total (no contamos UNILLANOS)
    total_demand = sum(
        demands.get(stop_id, 0)
        for stop_id in route
        if stop_id != "UNILLANOS"
    )

    over_capacity = max(0, total_demand - capacity)
    penalty_km = over_capacity * PENALTY_PER_STUDENT_KM
    objective_km = total_distance + penalty_km
    fitness = 1.0 / (1.0 + objective_km)
    time_min = (total_distance / AVG_SPEED_KMH) * 60.0 if total_distance > 0 else 0.0

    return {
        "route": route,
        "total_distance_km": total_distance,
        "total_demand": total_demand,
        "over_capacity": over_capacity,
        "penalty_km": penalty_km,
        "objective_km": objective_km,
        "fitness": fitness,
        "time_min": time_min,
    }
```

$$Dist(R) + penalidad(R)$$

$$Penalidad = 2 \cdot max(0, Demanda - Capacidad)$$

Metaheurísticas Implementadas

Greedy (Algoritmo Voraz)

Se basa en seleccionar siempre:

$$Siguiente\ nodo = arg\ min_{j \notin visitados} D_{actual,j}$$

Es decir, siempre el paradero más cercano disponible.

No garantiza óptimos globales.

```python
while remaining:
        best_next = None
        best_dist = float("inf")
        for cand in remaining:
            d = DIST_MATRIX[IDX[current]][IDX[cand]]
            if d < best_dist:
                best_dist = d
                best_next = cand
        route.append(best_next)
        remaining.remove(best_next)
        current = best_next
```

Algoritmo Genético (GA)

Inspirado en evolución biológica

Población

$$P = \{R_1, R_2, ..., R_m\}$$

Cada ruta es un individuo

```python
population = [make_random_route(stops_to_visit) for _ in range(population_size)]
```

Evaluación

A cada individuo se le aplica

$$Fitness(R_i)$$

Selección

Se elige una proporción de los mejores:

$$P_{elite} = top(P, 30\%)$$

```python
def select():
            r = random.random() * total_fit
            acc = 0.0
            for f, rt in scored:
                acc += f
                if acc >= r:
                    return rt
            return scored[-1][1]
```

Crossover(Cruce)

En el código se realiza la implementación de:

$$Hijo = CrossoverOrdenado(Padre_1, Padre_2)$$

```python
middle = parent1[i:j]
    rest = [x for x in parent2 if x not in middle or x == "UNILLANOS"
```

El cruce por orden garantiza que:

- No se repitan nodos
- Se respeten las posiciones heredadas parcialmente

Mutación

$$Swap(R[i], R[j])$$

```python
def mutate(route: List[str], mutation_rate: float = 0.15) -> List[str]:
    route = route[:]
    inner_indices = list(range(1, len(route) - 1))
    for i in inner_indices:
        if random.random() < mutation_rate:
            j = random.choice(inner_indices)
            route[i], route[j] = route[j], route[i]
    return route
```

Probabilidad de Mutación = 15%

Generaciones

$$Iterar\ por\ 80\ generaciones$$

El GA progresa hacia soluciones de error fitness

Simulated Annealing (SA)

Basado en la mecánica estadística

Estado Inicial

Ruta Greddy:

$$R_0$$

Temperatura

$$T_0 = 100$$

$$T_{k+1} = T_k * \alpha \quad \text{sabiendo que } \alpha = 0.99$$

Movimiento(Vecino)

$$R' = Swap(R)$$

```
candidate_route = neighbor_swap(current_route)
            cand_info = evaluate_route(candidate_route, demands, capacity)
```

Probabilidad de Aceptación

Si R' es peor

$$P = exp(-\frac{Fitness(R') - Fitness(R)}{T})$$

El sistema acepta soluciones malas mientras T es alta, permitiendo escapar de mínimos locales

```python
if delta < 0:
                current_route = candidate_route
                current_info = cand_info
                current_score = cand_score
            else:
                p = math.exp(-delta / T)
                if random.random() < p:
                    current_route = candidate_route
                    current_info = cand_info
                    current_score = cand_score
```

4. Herramientas y Librerías utilizadas

Para el desarrollo del sistema de optimización de rutas institucionales se empleó una arquitectura basada en tecnologías web y librerías especializadas tanto para el procesamiento en el backend como para la interfaz de usuario. A continuación, se describen las herramientas utilizadas, su función dentro del proyecto y la justificación de su uso.

4.1. Lenguajes y Tecnologías Principales

Python

El backend del sistema fue desarrollado en Python, debido a su versatilidad, amplia disponibilidad de librerías científicas y facilidad para implementar algoritmos metaheurísticos. Python permitió integrar cálculos geoespaciales, heurísticas, modelos de datos y un servidor web de manera eficiente y estructurada.

HTML y CSS

La interfaz gráfica del sistema fue construida utilizando HTML y CSS, tecnologías esenciales para el desarrollo web.

- HTML se empleó para estructurar el contenido visual del sistema, creando la página principal donde el usuario selecciona rutas activas, ingresa demandas y visualiza resultados.
- CSS se utilizó para estilizar y dar diseño a la interfaz, garantizando una presentación limpia, responsiva y fácil de usar.

Estas tecnologías fueron necesarias para permitir la interacción directa del usuario con el sistema desde cualquier navegador sin instalar software adicional.

4.2. Frameworks y Librerías de Python

FastAPI

FastAPI fue el framework principal para construir la API del sistema.
Su utilización se justifica por:

- Alto rendimiento gracias a su estructura asíncrona.
- Validación automática de datos mediante Pydantic.
- Generación automática de documentación en Swagger.
- Sencillez para implementar endpoints ligeros y eficientes.

En el proyecto, FastAPI gestiona dos funciones principales:

1. Servir el archivo index.html (interfaz del usuario).
2. Ejecutar el endpoint /solve encargado de correr las metaheurísticas de optimización.

Pydantic

La librería Pydantic se utilizó para definir y validar los modelos de datos que recibe y retorna la API, como SolveRequest y SolveResponse.
Fue necesaria porque:

- Garantiza que los datos enviados por el usuario cumplan con el formato esperado.
- Previene errores por tipos incorrectos.
- Facilita la conversión ordenada de los datos hacia estructuras Python.

FastAPI CORS Middleware (CORSMiddleware)

Esta herramienta fue integrada para permitir que el frontend (HTML + JavaScript) pueda comunicarse con el servidor sin restricciones de seguridad del navegador (CORS).
Se volvió necesaria porque:

- La página HTML puede ejecutarse desde rutas locales o servidores distintos.
- Sin CORS habilitado, el navegador bloquearía las solicitudes a /solve.

Pathlib

La librería pathlib se utilizó para manejar rutas de archivos de forma segura y compatible con distintos sistemas operativos.

El proyecto permite referenciar y leer correctamente el archivo index.html sin errores de direccionamiento.

Math y Random

Estas librerías estándar se emplearon para cálculos matemáticos y generación de valores aleatorios:

- math permitió implementar la fórmula Haversine para calcular distancias geográficas entre paraderos.
- random fue esencial para los algoritmos metaheurísticos (AG y SA), ya que generan soluciones aleatorias, mutaciones y selecciones estocásticas.

Su uso fue indispensable para garantizar el correcto funcionamiento de los algoritmos de optimización.

Time

La librería time se utilizó para medir el rendimiento computacional (CPU Time) de cada algoritmo de optimización.
Esta métrica se incluye en la respuesta del backend para evaluar la eficiencia de cada método.

4.3 Estructuras de Datos y Componentes Computacionales

El sistema emplea:

- Diccionarios para almacenar coordenadas de paraderos.
- Tablas de rutas oficiales del boletín.
- Matriz de distancias basada en Haversine.
- Modelos de ruta y penalización por sobrecapacidad.
- Metaheurísticas como Greedy, Algoritmo Genético y Simulated Annealing.

El uso de estas estructuras permitió un cálculo eficiente y escalable para el problema del TSP adaptado con restricciones de demanda y capacidad

5. Historias de Usuario (HU)

Código: **HU-01**	Usuario: **administrador del sistema**
Título: **Gestionar Rutas Oficiales**	
Prioridad: **Alta**	Tipo: **Historia**
Descripción: Como administrador del sistema Quiero seleccionar y configurar rutas oficiales del boletín universitario Para permitir combinaciones flexibles según necesidades operativas	
Criterios de aceptación ● Establecer "UNILLANOS" como origen principal predeterminado ● Permitir selección múltiple de destinos mediante checkboxes ● Proporcionar interfaz visual con mapa interactivo ● Validar que al menos un destino sea seleccionado ● Configurar rutas predefinidas del boletín como conjuntos de destinos	

Código: **HU-02**	Usuario: **administrador del sistema**
Título: **Optimización con Algoritmos Genéticos**	
Prioridad: **Alta**	Tipo: **Historia**
Descripción: Como administrador del sistema Quiero ejecutar algoritmos genéticos para generar rutas óptimas Para minimizar la distancia total considerando restricciones de capacidad	
Criterios de aceptación ● Implementar operadores de selección, cruce y mutación configurables ● Evaluar rutas basándose en distancia, tiempo y penalizaciones ● Garantizar convergencia en número razonable de generaciones ● Aplicar penalizaciones por sobrecapacidad del vehículo ● Generar soluciones reproducibles con semilla fija ● Completar optimización en menos de 30 segundos para 20 paraderos	

Código: **HU-03**	Usuario: **administrador del sistema**

Título: **Visualización de Rutas Optimizadas**	
Prioridad: **Alta**	Tipo: **Historia**
Descripción: Como administrador del sistema Quiero visualizar gráficamente las rutas optimizadas en el mapa Para analizar y validar las soluciones generadas por el sistema	
Criterios de aceptación ● Integrar Leaflet.js con OpenStreetMap para visualización ● Diferenciar rutas oficiales (gris) vs optimizadas (rojo con patrón dashed) ● Incluir leyenda explicativa de estilos y colores ● Mostrar información detallada en popups al hacer clic ● Ajustar automáticamente zoom y centro del mapa ● Utilizar OSRM para calcular rutas realistas por calles	

Código: **HU-04**	Usuario: **administrador del sistema**
Título: **Monitoreo de Métricas de Rendimiento**	
Prioridad: **Media**	Tipo: **Historia**
Descripción: Como administrador del sistema Quiero monitorear las métricas de rendimiento de las optimizaciones Para evaluar la eficiencia y calidad de las soluciones generadas	
Criterios de aceptación ● Mostrar distancia total en kilómetros ● Mostrar tiempo estimado en minutos ● Mostrar demanda vs capacidad del vehículo ● Calcular y mostrar fitness de la solución ● Medir y mostrar tiempo de ejecución en CPU ● Indicar estado de validación de restricciones	

Código: **HU-05**	Usuario: **administrador del sistema**
Título: **Configuración de Demanda y Capacidad con Monitoreo de Optimización**	
Prioridad: **Alta**	Tipo: **Historia**
Descripción: Como administrador del sistema Quiero poder configurar la capacidad del bus y la demanda de estudiantes por paradero, y visualizar el paso a paso del algoritmo de optimización Para entender cómo se genera la ruta óptima y ajustar los parámetros según la demanda real	
Criterios de aceptación ● Configurar capacidad del bus mediante campo numérico con validación de valores positivos ● Establecer demanda individual por cada paradero mediante inputs numéricos ● Visualizar la demanda asignada en los popups informativos del mapa ● Acceder a un overlay detallado que muestre cada iteración del algoritmo ● Ver las fórmulas matemáticas aplicadas en cada paso del cálculo ● Observar gráficas en tiempo real de la evolución del algoritmo ● Validar automáticamente que la demanda total no exceda significativamente la capacidad	

6. Requisitos específicos

Número de requisito	RF-01
Nombre de requisito	Selección de rutas a destinos
Tipo	☑ ~~Requisito~~ ☐ Restricción
Fuente del requisito	Requerimiento de los usuarios (docentes, administrativos, estudiantes)
Prioridad del requisito	☑ ~~Alta~~ ☐ Media ☐ Baja

Número de requisito	RF-02
Nombre de requisito	Generacion de rutas optimas mediante algoritmos geneticos
Tipo	☑ ~~Requisito~~ ☐ Restricción
Fuente del requisito	Modelado matemático y necesidad de optimización

Prioridad del requisito	☑ ~~Alta~~	☐ Media	☐ Baja

Número de requisito	RF-03		
Nombre de requisito	Visualización gráfica de la ruta optimizada		
Tipo	☑ ~~Requisito~~	☐ Restricción	
Fuente del requisito	Experiencia de usuario / Interfaz		
Prioridad del requisito	☑ ~~Alta~~	☐ Media	☐ Baja

Número de requisito	RF-04		
Nombre de requisito	Configuración de parámetros del algoritmo		
Tipo	☑ ~~Requisito~~	☐ Restricción	
Fuente del requisito	Equipo de desarrollo / Administrativos		
Prioridad del requisito	☑ ~~Alta~~	☐ Media	☐ Baja

Número de requisito	RF-05		
Nombre de requisito	Gestión de Demanda y Capacidad con Análisis de Algoritmo		
Tipo	☑ ~~Requisito~~	☐ Restricción	
Fuente del requisito	Equipo de desarrollo / Administrativos		
Prioridad del requisito	☐ Alta	☑ ~~Media~~	☐ Baja

6.1. Requisitos comunes de los interfaces.

6.1.1. Interfaces de usuario

- Panel de control lateral: Interfaz flotante con controles de configuración
- Mapa interactivo: Visualización central con capas de rutas diferenciadas
- Controles de selección: Checkboxes para rutas oficiales, dropdown para algoritmos
- Formularios dinámicos: Inputs numéricos para configuración de demanda
- Panel de métricas: Visualización en tiempo real de indicadores de desempeño
- Leyenda gráfica: Diferenciación visual entre rutas oficiales y optimizadas

6.1.2. Interfaces de hardware

- Cliente: Navegador web con soporte HTML5, mínimo 2GB RAM
- Servidor: 2GB RAM, 1 CPU core, 10GB almacenamiento, conexión internet

- Display: Resolución mínima 1024x768 para correcta visualización

6.1.3. Interfaces de software

- Sistema Operativo: Compatible con Windows 10+
- Backend: FastAPI 0.104+, Python 3.8+, Uvicorn, Pydantic
- Frontend: Leaflet 1.9.4, JavaScript ES6+, CSS3
- Servicios Externos: OSRM para cálculo de rutas realistas

6.1.4. Interfaces de comunicación

- Protocolo: HTTP/HTTPS para comunicación cliente-servidor
- Formato Datos: JSON para intercambio de información
- API REST: Endpoints estándar RESTful para operaciones de optimización

6.2. Requisitos funcionales

6.2.1. RF-01: Selección de rutas

Descripción:
El sistema debe permitir a los usuarios seleccionar y configurar los puntos de las rutas para las rutas de transporte, con interfaces intuitivas que faciliten esta selección y permitan combinaciones flexibles según las necesidades operativas.

Criterios de aceptación

- Origen fijo predeterminado: Establecer "UNILLANOS" como origen principal por defecto en todas las optimizaciones
- Selección múltiple de destinos: Permitir seleccionar múltiples puntos de destino mediante checkboxes o selección múltiple
- Interfaz visual: Proporcionar un mapa interactivo donde los usuarios puedan visualizar y seleccionar los puntos de origen/destino
- Validación de selección: Asegurar que al menos un destino sea seleccionado antes de proceder con la optimización
- Configuración de rutas oficiales: Permitir la selección de rutas predefinidas del boletín universitario como conjuntos de destinos

6.2.2. RF-02: Generación de rutas óptimas mediante algoritmos genéticos

Descripción:
El sistema debe implementar y ejecutar algoritmos genéticos para generar rutas óptimas que minimicen la distancia total de recorrido, considerando restricciones de capacidad y optimizando la secuencia de visitas a los paraderos seleccionados.

Criterios de aceptación

- Implementación GA completa: Incluir operadores de selección, cruce y mutación con parámetros configurables
- Función de fitness robusta: Evaluar rutas basándose en distancia total, tiempo estimado y penalizaciones por sobrecapacidad
- Convergencia garantizada: El algoritmo debe converger hacia soluciones mejoradas en un número razonable de generaciones
- Manejo de restricciones: Aplicar penalizaciones cuando la demanda total exceda la capacidad del vehículo
- Resultados reproducibles: Generar soluciones consistentes mediante semilla fija (random.seed(42))
- Tiempos de ejecución: Completar la optimización en menos de 30 segundos para casos con hasta 20 paraderos

6.2.3. RF-03: Visualización gráfica de la ruta optimizada

Descripción:
El sistema debe proporcionar una visualización gráfica clara e interactiva de las rutas optimizadas sobre un mapa geográfico, diferenciando visualmente entre rutas oficiales y rutas optimizadas, con capacidades de zoom, pan y consulta de información específica por paradero.

Criterios de aceptación

- Mapa interactivo: Integrar Leaflet.js con OpenStreetMap para visualización base
- Diferenciación visual: Rutas oficiales en gris (color: #9ca3af, opacidad: 0.5) y rutas optimizadas en rojo con patrón dashed (color: #ef4444, dashArray: "6 4")
- Leyenda explicativa: Incluir leyenda que explique los diferentes estilos y colores de rutas
- Popup informativo: Al hacer clic en cualquier paradero, mostrar información detallada (nombre, demanda actual, coordenadas)
- Ajuste de vista: Ajustar automáticamente el zoom y centro del mapa para mostrar la ruta completa optimizada

- Rutas realistas: Utilizar OSRM para calcular y mostrar la ruta real por calles y vías, no solo línea recta

6.2.4. RF-04: Configuración de parámetros del algoritmo

Descripción:
El sistema debe permitir la configuración de los parámetros clave del algoritmo genético para ajustar su comportamiento según las necesidades específicas de optimización, balanceando entre velocidad de convergencia y calidad de la solución.

Criterios de aceptación

- Tamaño de población: Configurable entre 20 y 100 individuos (valor por defecto: 40)
- Número de generaciones: Configurable entre 50 y 200 iteraciones (valor por defecto: 80)
- Tasa de mutación: Configurable entre 0.05 y 0.3 (valor por defecto: 0.15)
- Tamaño de élite: Configurable entre 1 y 10 individuos (valor por defecto: 5)
- Capacidad del vehículo: Configurable según el tipo de vehículo (valor por defecto: 40)
- Interfaz intuitiva: Proporcionar controles deslizantes o inputs numéricos con validación
- Valores por defecto optimizados: Establecer valores predefinidos que funcionen bien para la mayoría de casos
- Reset a valores por defecto: Permitir restaurar rápidamente la configuración inicial

6.2.5. .RF-05: Configuración de parámetros del algoritmo

Descripción:
El sistema debe permitir al administrador definir la capacidad del bus y la demanda de estudiantes por paradero, y proporcionar un análisis detallado paso a paso del algoritmo de optimización para facilitar la comprensión y validación de los resultados.

Criterios de aceptación
Configuración de capacidad del bus:

- Campo de entrada numérico para capacidad con valor por defecto de 40 estudiantes
- Validación para aceptar sólo valores enteros positivos mayores a 0
- Actualización inmediata de las restricciones en los cálculos de optimización

Establecimiento de demanda por paradero:

- Interfaz con lista de todos los paraderos (excluyendo UNILLANOS) con campos de entrada numérica
- Valores por defecto en 0 para todos los paraderos
- Validación para valores enteros no negativos
- Actualización en tiempo real de la información en los popups del mapa

Visualización de información contextual:

- Mostrar la demanda actualizada en los popups de cada marcador del mapa
- Indicar capacidad utilizada vs capacidad total en las métricas principales
- Resaltar paraderos con demanda significativa en la interfaz

Análisis paso a paso del algoritmo:
- Overlay modal que muestre iteración por iteración del algoritmo seleccionado
- Detalle de la ruta considerada en cada paso con su secuencia completa
- Métricas numéricas de distancia, objetivo y fitness para cada iteración
- Fórmulas matemáticas desglosadas mostrando los cálculos intermedios
- Indicación visual cuando una iteración establece un nuevo mejor resultado

Monitoreo gráfico del progreso:
- Gráfica de evolución de la distancia total a lo largo de las iteraciones
- Gráfica de la función objetivo (distancia + penalizaciones) durante la optimización
- Gráfica del fitness de la solución en cada paso del algoritmo
- Gráfica de temperatura (para Simulated Annealing) mostrando el enfriamiento

Validación de restricciones:
- Cálculo automático de demanda total por ruta
- Comparación visual entre demanda total y capacidad del bus
- Aplicación de penalizaciones en la función objetivo por sobrecapacidad
- Indicadores de estado que muestren si la solución está dentro de capacidad

Interfaz de usuario:
- Botón dedicado para abrir el overlay de análisis paso a paso
- Diseño responsive que permita visualizar tanto lista de pasos como gráficas simultáneamente
- Navegación fluida entre las diferentes iteraciones del algoritmo

6.3. REQUISITOS NO FUNCIONALES

6.3.1. Requisitos de rendimiento

RNF-01: Rendimiento - Tiempo de respuesta de interfaz

Descripción

El sistema debe garantizar tiempos de respuesta rápidos en todas las operaciones de la interfaz de usuario, manteniendo una experiencia fluida e interactiva para los usuarios durante la configuración y visualización de rutas.

Criterios de aceptación

- Operaciones de UI: Respuesta en menos de 2 segundos para todas las interacciones del usuario (clic, selección, actualización)
- Carga inicial: Tiempo de carga inicial de la aplicación menor a 3 segundos
- Actualizaciones mapa: Renderizado de rutas en el mapa en menos de 1.5 segundos tras la optimización
- Responsividad: Feedback visual inmediato (< 100ms) para acciones del usuario

RNF-02: Rendimiento - Tiempo de cálculo de optimización

Descripción
El sistema debe completar los cálculos de optimización dentro de límites de tiempo específicos según el algoritmo utilizado, balanceando calidad de solución y tiempo de procesamiento.

Criterios de aceptación

- Algoritmo Greedy: Completar optimización en menos de 5 segundos para hasta 30 paraderos
- Recocido Simulado: Completar optimización en menos de 10 segundos para hasta 30 paraderos
- Algoritmo Genético: Completar optimización en menos de 30 segundos para hasta 30 paraderos

RNF-03: Rendimiento - Carga de datos

Descripción
El sistema debe manejar eficientemente grandes volúmenes de datos geográficos y de configuración sin impactar el rendimiento general.

Criterios de aceptación

- Paraderos simultáneos: Manejo eficiente de hasta 30 paraderos en una sola optimización
- Matriz de distancias: Cálculo y almacenamiento eficiente de matriz de distancias para 30+ paraderos
- Memoria cliente: Consumo máximo de 512MB RAM en el navegador del cliente
- Memoria servidor: Consumo máximo de 1GB RAM en el servidor durante operaciones
- Transferencia datos: Tamaño máximo de respuesta API de 1MB

6.3.2 Mantenibilidad

RNF-04: Mantenibilidad - Modularidad del código

Descripción
El código del sistema debe estar organizado en módulos claramente separados con responsabilidades bien definidas para facilitar el mantenimiento y la evolución.

Criterios de aceptación

- Separación backend/frontend: Arquitectura clara cliente-servidor
- Módulos algorítmicos: Separación de algoritmos en funciones independientes
- APIs bien definidas: Interfaces claras entre componentes
- Configuración externalizada: Parámetros en ubicaciones centralizadas
- Dependencias gestionadas: Uso de requirements.txt y package.json

7. Apéndices

7.1. Diagrama de Clases del Sistema

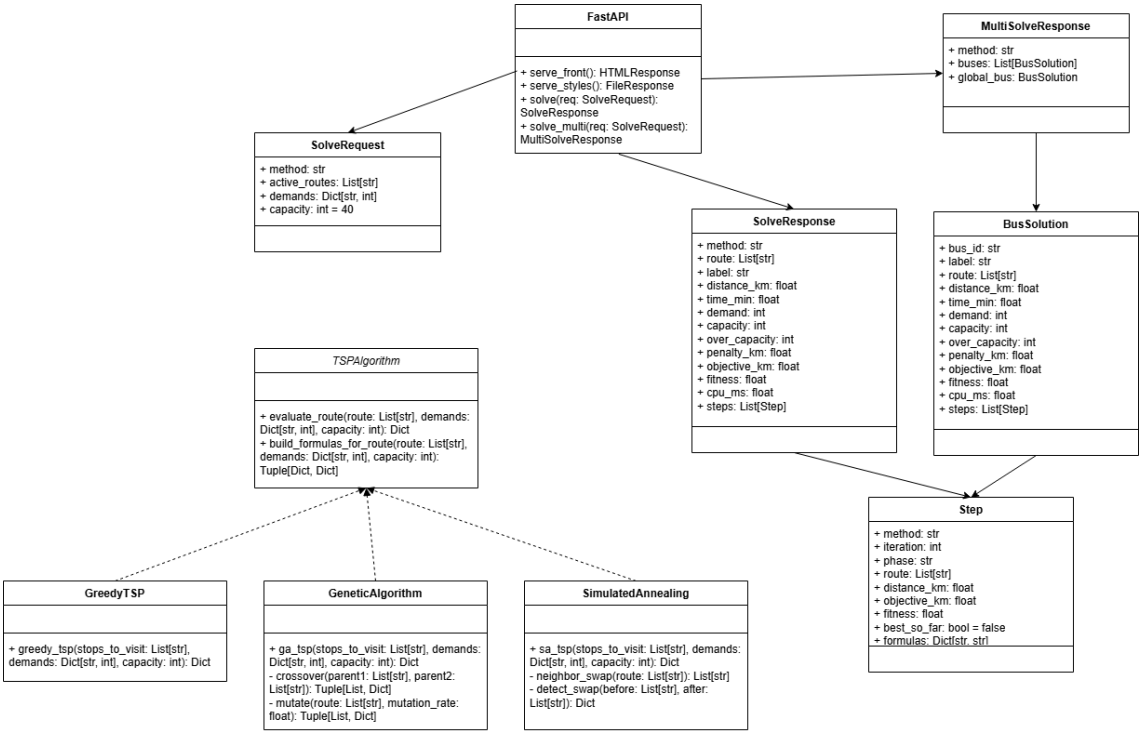

Figura 1 : Diagrama de clases
link: [Diagrama De Clases]

7.2. Diagrama de Secuencia del Sistema

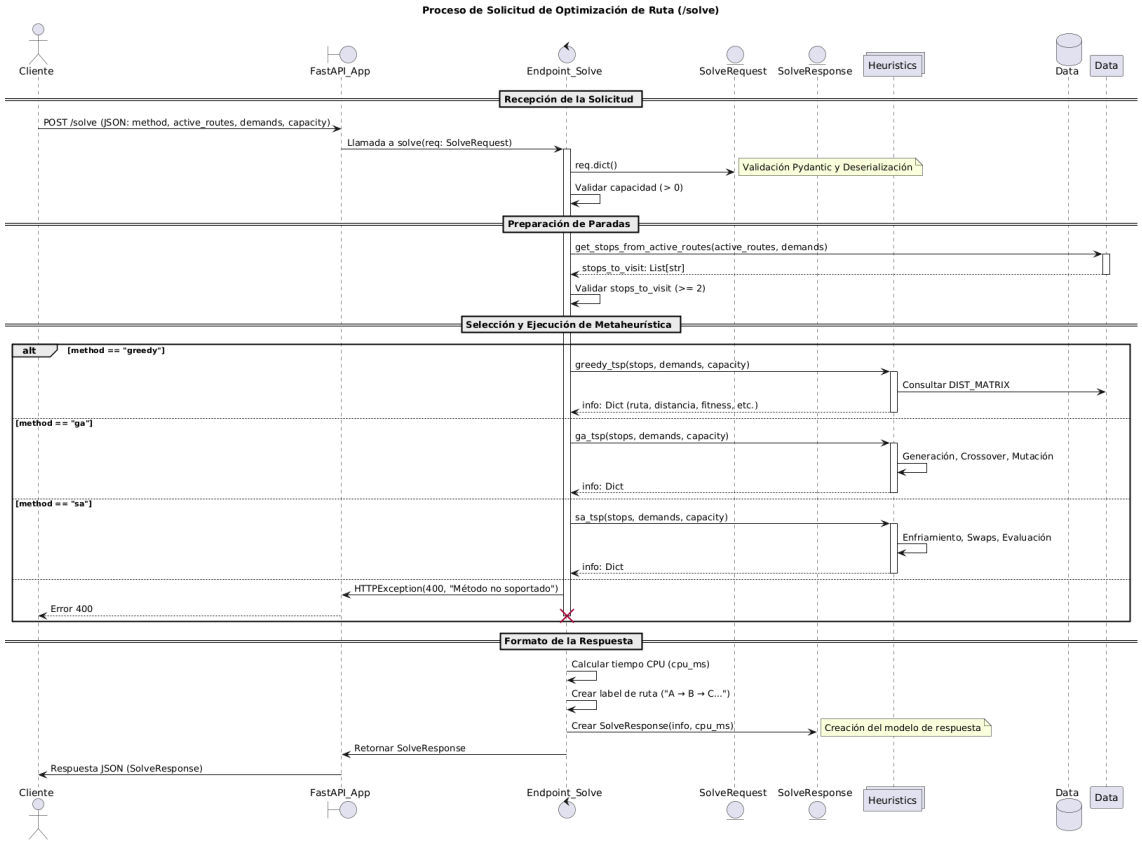

Figura 2: Diagrama de secuencia

7.3. Diagrama de Arquitectura del Sistema

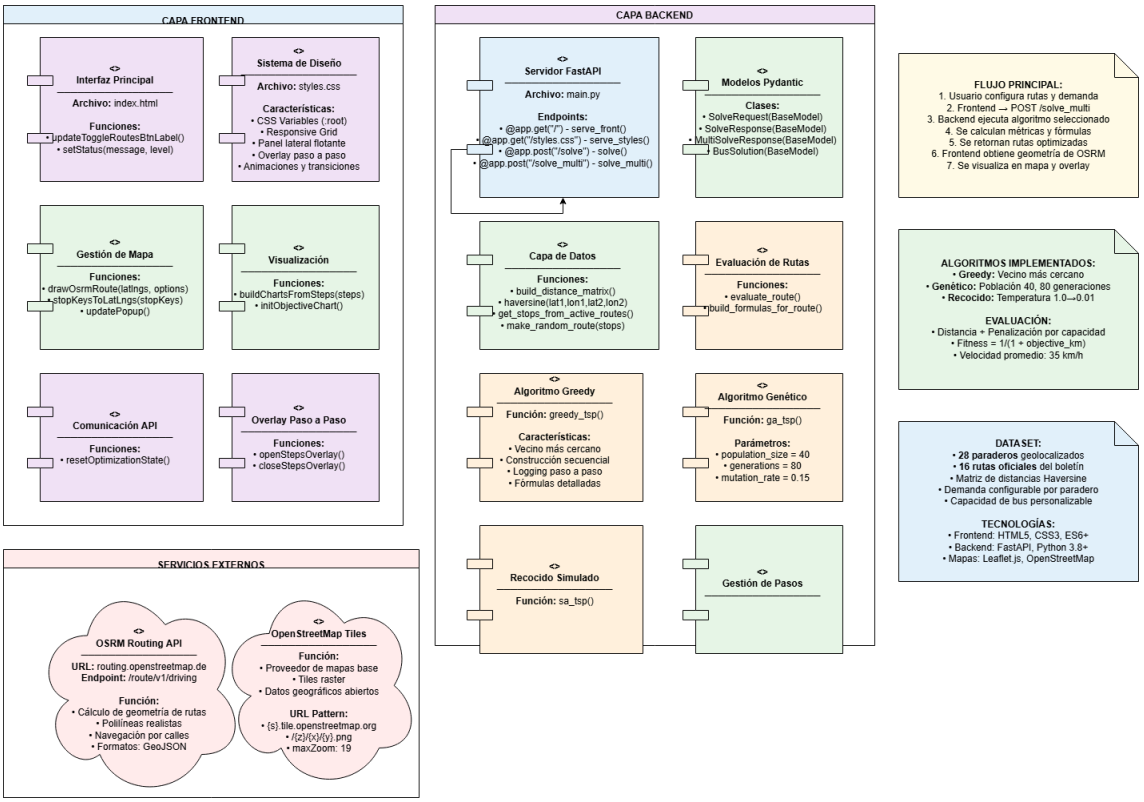

Figura 3: Diagrama De arquitectura
Link:[Diagrama De arquitectura]

7.4. Diagrama de casos de uso

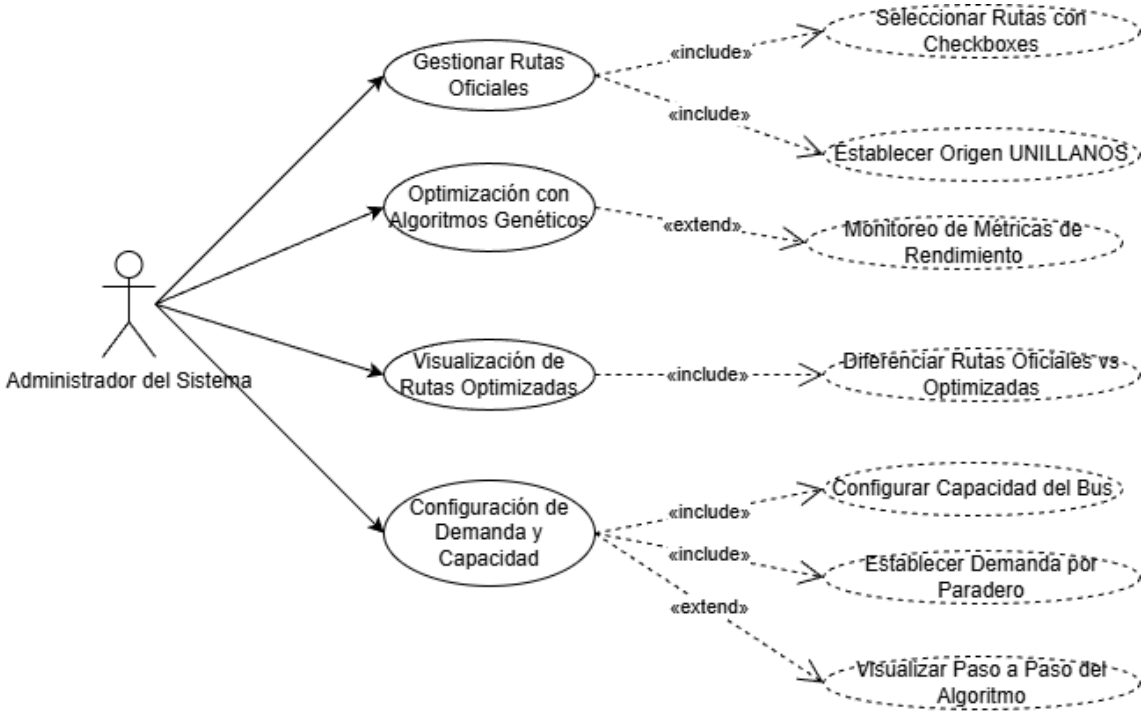

Figura 4: Diagrama De Casos De Uso

7.5. Glosario de Términos Técnicos

- **Cromosoma:** Representación de una solución (ruta) como lista de paraderos
- **Fitness**: Valor que representa la calidad de una solución (mayor = mejor)
- **GA:** Algoritmo Genético - técnica de optimización basada en evolución natural
- **SA:** Recocido Simulado - técnica de optimización basada en proceso térmico
- **OSRM:** Open Source Routing Machine - servicio para cálculo de rutas realistas
- **Paradero:** Punto de recogida o dejada de estudiantes en la ruta