System Requirements Specification

Swift Signals

INSIDE INSIGHTS

For Southern Cross Solutions (Pty) Ltd.
insideinsights2025@gmail.com
Capstone Project
University of Pretoria

Swift Signals

Table of Contents

Overview

Swift Signals is a data-driven, simulation-powered traffic light optimization platform designed to address traffic congestion in urban environments. Developed in collaboration with Southern Cross Solutions, the project aims to equip municipal traffic departments with advanced tools to analyse intersection performance and improve traffic signal efficiency using machine learning.

Traffic congestion remains one of South Africa's most costly infrastructure challenges, with the South African Road Federation estimating annual productivity losses of approximately R1 billion. *Swift Signals* tackles this issue by providing a modular, web-based platform capable of simulating real-world intersection behaviour and dynamically optimizing traffic light phase configurations using historical traffic data. The system leverages modern software engineering principles—microservices, containerization, and continuous deployment pipelines—to deliver a scalable and maintainable solution.

Objectives

The *Swift Signals* platform is designed with the following objectives:

- **Simulate traffic flow at intersections** using historical and real-time data to model congestion patterns, vehicle throughput, and signal behaviour across different times of day.
- **Apply Swarm Optimization Algorithms** (e.g., Particle Swarm Optimization) to generate and evaluate multiple traffic light timing configurations, selecting the most efficient ones based on defined performance metrics such as wait time and throughput.
- **Leverage MongoDB** as the primary database to store and manage flexible, time-series traffic datasets, simulation outputs, and optimization results with scalability and efficiency.
- **Implement a microservices architecture** to modularize the system into distinct services such as simulation, optimization, API management, frontend interface, and data storage—each deployable and testable independently.

- **Develop a responsive web portal** using React.js and TailwindCSS, allowing users to:
    - Configure intersections and signal sequences.
    - Launch and monitor simulations.
    - View optimization reports and visual analytics.
    - Interact with traffic flow visualizations and system alerts.
- **Follow an Agile development process**, working in two-week sprints with continuous stakeholder feedback, bi-weekly deliverables, and regular sprint review sessions to ensure alignment with client expectations.
- **Design for future scalability**, including:
    - Support for optimizing multiple intersections simultaneously.
    - Integration of real-time traffic data feeds.
    - Expansion to more complex intersection models (e.g., turn-only lanes, variable lane counts).

Use Cases

User Characteristics

[Insert User Characteristics here]

User Stories

Login Screen

Login Form Functionality

- Given I am on the login page, I should see a form with fields for username and password.
- When I enter text in the username field, the input should update the username state.
- When I enter text in the password field, the input should update the password state.
- When I submit the form with both fields filled, the system should log the input values and redirect me to the dashboard page.
- When I submit the form with either field empty, the system should display a console message indicating that all fields must be filled.

Traffic Light Indicator

- Given I am on the login page, I should see a traffic light component with red, yellow, and green lights.
- When the username field is empty, the red and yellow lights should appear inactive (dimmed with a neutral border).
- When I enter text in the username field, the red and yellow lights should activate (bright with a glowing effect and distinct border).
- When I enter text in the password field, the green light should activate (bright with a glowing effect and distinct border).
- The traffic light should have a modern, visually appealing design with smooth transitions and shadow effects.

Forgot Password Modal

- Given I am on the login page, I should see a "Forgot Password?" link.

- When I click the "Forgot Password?" link, a modal should appear with an email input field and "Cancel" and "Send Reset Link" buttons.
- When I enter an email and click "Send Reset Link," the system should log the email and close the modal.
- When I click "Cancel," the modal should close without logging any data.
- The modal should be styled consistently with the login page and appear centered with a semi-transparent background.

Navigation

- Given I am on the login page, I should see a "Register Here" link.
- When I click the "Register Here" link, I should be redirected to the signup page.
- When I successfully submit the login form, I should be redirected to the dashboard page.

Visual and Accessibility Requirements

- The login page should have a responsive design, adapting to mobile and desktop screens.
- The page should include a logo and a welcome message ("Welcome to Swift Signals") at the top.
- Form inputs should have accessible labels (e.g., using sr-only for screen readers).
- The traffic light and form elements should use Tailwind CSS for consistent styling.
- The page should use a light colour scheme with a gradient background and shadow effects for a modern look.

Sign-Up Screen

Sign-Up Form Functionality

- Given I am on the sign-up page, I should see a form with fields for username, email, and password.
- When I enter text in the username field, the input should update the username state.
- When I enter text in the email field, the input should update the email state.
- When I enter text in the password field, the input should update the password state.

- When I submit the form with all fields filled, the system should log the input values and redirect me to the login page.
- When I submit the form with any field empty, the system should display a console message indicating that all fields must be filled.

Traffic Light Indicator

- Given I am on the sign-up page, I should see a traffic light component with red, yellow, and green lights.
- When the username field is empty, the red light should appear inactive (dimmed with a neutral border).
- When I enter text in the username field, the red light should activate (bright with a glowing effect and distinct border).
- When I enter text in the email field, the yellow light should activate (bright with a glowing effect and distinct border).
- When I enter text in the password field, the green light should activate (bright with a glowing effect and distinct border).
- The traffic light should have a modern, visually appealing design with smooth transitions and shadow effects.

Navigation

- Given I am on the sign-up page, I should see a "Login here" link.
- When I click the "Login here" link, I should be redirected to the login page.
- When I successfully submit the sign-up form, I should be redirected to the login page.

Visual and Accessibility Requirements

- The sign-up page should have a responsive design, adapting to mobile and desktop screens.
- The page should include a logo and a welcome message ("Welcome to Swift Signals") at the top.
- Form inputs should have accessible labels (e.g., using sr-only for screen readers).
- The traffic light and form elements should use Tailwind CSS for consistent styling.
- The page should use a light colour scheme with a gradient background and shadow effects for a modern look.

Dashboard Screen

Summary Cards

- Given I am on the dashboard page, I should see three summary cards displaying:
  - Total Intersections (e.g., "24") with a road icon.
  - Active Simulations (e.g., "8") with a play icon.
  - Optimization Runs (e.g., "156") with a chart line icon.
- Each card should have a consistent design with Tailwind CSS styling, including icons in distinct colours (blue, green, purple) and a clean layout.

Quick Actions

- Given I am on the dashboard page, I should see three quick action buttons:
  - "New Intersection" with a plus icon.
  - "Run Simulation" with a play icon.
  - "View Map" with a map icon.
- Each button should have a distinct background colour (indigo, green, purple) and hover effects.
- Clicking the buttons should be clickable but currently log no action (placeholder functionality).

Recent Simulations Table

- Given I am on the dashboard page, I should see a table listing recent simulations with columns for:
  - ID (e.g., "#1234").
  - Intersection (e.g., "Main St & 5th Ave").
  - Status (e.g., "Complete", "Running", "Failed").
  - Actions (e.g., "View Details" button).
- The status should display with color-coded badges (green for Complete, yellow for Running, red for Failed).
- The "View Details" button should be styled as a clickable link but currently log no action.

Traffic Volume Chart

- Given I am on the dashboard page, I should see a line chart displaying traffic volume over time (e.g., 6 AM to 10 AM).
- The chart should use Chart.js with a gradient fill (red-to-transparent), smooth line tension, and customized tooltips styled with Tailwind CSS colours.

- The chart should be responsive, with no grid lines on the x-axis, light grid lines on the y-axis, and formatted tick labels.
- The chart should clean up properly when the component unmounts to prevent memory leaks.

Top Intersections Section

- Given I am on the dashboard page, I should see a section listing the top intersections by vehicle volume (e.g., "Main St & 5th Ave: 15,000 vehicles").
- The section should include an "Avg Daily Volume" summary (e.g., "12,000 vehicles").
- The data should be displayed in a clean, bordered list with Tailwind CSS styling.

Visual and Accessibility Requirements

- The dashboard should include a Navbar at the top and a Footer at the bottom, consistent with the application's design.
- The page should have a responsive layout, adapting to mobile and desktop screens using a grid system for larger screens.
- The page should use a light background (gray-100) with dark mode support (gray-900).
- All text, including headings, should be styled with Tailwind CSS for consistency and readability.
- The chart and table should be accessible, with clear labels and colour contrasts suitable for screen readers and visual clarity.

Intersections Page

Search Functionality

- Given I am on the intersections page, I should see a search bar with a placeholder text "Search by Name or ID...".
- When I type in the search bar, the list of intersections should dynamically filter to show only those matching the search query by name (case-insensitive) or ID.
- The search bar should include a search icon (from Lucide React) positioned on the right side.
- The search bar should be styled with Tailwind CSS, including a border, rounded edges, and a focus ring in red.

Intersection List

- Given I am on the intersections page, I should see a list of intersection cards, each displaying:
  - ID (e.g., "1").
  - Name (e.g., "Main St & 1st Ave").
  - Location (e.g., "Pretoria CBD").
  - Lanes (e.g., "4-way, 2 lanes each").
- Each card should be rendered using the IntersectionCard component and support three actions: Simulate, Edit, and Delete.
- Clicking the Simulate, Edit, or Delete buttons should log a corresponding message to the console (e.g., "Simulate 1", "Edit 1", "Delete 1").
- If no intersections match the search query, the list should display no cards.

Add Intersection Button

- Given I am on the intersections page, I should see an "Add Intersection" button styled in red (bg-red-700, hover:bg-red-800).
- Clicking the button should log "Add Intersection" to the console (placeholder functionality).
- The button should be positioned to the right of the search bar in the top bar.

Visual and Accessibility Requirements

- The intersections page should include a Navbar at the top, consistent with the application's design.
- The page should have a responsive layout, adapting to mobile and desktop screens, with a maximum width of 6xl (Tailwind CSS) and padding.
- The page should use a light background (gray-100) with a scrollable container for the intersection list.
- The search bar should have an accessible placeholder and be usable with screen readers.
- Intersection cards should be spaced vertically and styled with Tailwind CSS for consistency and readability.
- The search icon should be clearly visible and properly aligned within the search bar.

Simulations Page

Simulation and Optimization Tables

- Given I am on the simulations page, I should see two tables: one for "Recent Simulations" and one for "Recent Optimizations".
- Each table should display columns for:
  - Simulation ID (e.g., "SIM001").
  - Intersection (e.g., "Main St & 1st Ave").
  - Avg Wait Time (e.g., "45.2").
  - Throughput (e.g., "1200").
  - Graph (bar chart comparing wait time and throughput).
  - Status (e.g., "Complete", "Running", "Failed").
  - Actions (View Results and Delete buttons).
- The status should display with color-coded badges (green for Complete, yellow for Running, red for Failed).
- The bar chart should use Chart.js with gradient fills (green for wait time, blue for throughput), no axis labels, and modern styling (rounded bars, tooltips).
- Clicking "View Results" should trigger an alert with the simulation ID (e.g., "Viewing results for simulation SIM001").
- Clicking "Delete" should trigger an alert with the simulation ID (e.g., "Deleting simulation SIM001").
- Each table should support pagination with 4 rows per page, displaying "Prev", page numbers, and "Next" buttons styled with Tailwind CSS gradients.

Filtering by Intersection

- Given I am on the simulations page, each table should have a dropdown to filter by intersection or "All Intersections".
- Selecting an intersection should filter the table to show only simulations/optimizations for that intersection and reset the page to 0.
- The dropdown should list all unique intersections from the respective table’s data, plus "All Intersections".

New Simulation/Optimization Modal

- Given I am on the simulations page, I should see a "New Simulation" button for the simulations table.
- Clicking "New Simulation" should open a modal titled "New Simulation" (or "New Optimization" for future functionality).
- The modal should include:
  - A text input for "Simulation Name".
  - A text area for "Simulation Description".
  - An intersection selection section with three tabs: List, Search, and Map.
- List Tab: Displays a dropdown of available intersections (from both tables' data). Selecting an intersection adds it to the selected intersections list.
- Search Tab: Allows typing a location and clicking "Add" to include it in the selected intersections list.
- Map Tab: Displays an interactive Leaflet map centered at [-26.2041, 28.0473] (South Africa) with zoom level 6. Clicking the map places a marker, logs coordinates, and adds them as an intersection to the selected list.
- Selected intersections should appear as removable pills (with a cross button) styled with Tailwind CSS.
- Clicking "Create" should validate that a name and at least one intersection are provided, log the data, clear the form, and navigate to /simulation-results with the data.
- Clicking "Cancel" or the close button (cross) should close the modal without saving.
- The modal should have a semi-transparent background, dark mode support, and Tailwind CSS styling.

Visual and Accessibility Requirements

- The page should include a Navbar at the top and a Footer at the bottom, consistent with the application's design.
- The page should have a responsive layout, using a two-column grid for medium and larger screens, and a single-column layout for mobile.
- The page should use a light background (gray-100) with dark mode support (gray-900).

- Tables, inputs, and buttons should be styled with Tailwind CSS for consistency and readability.
- The map should use OpenStreetMap tiles with proper attribution.
- All inputs and interactive elements (dropdowns, buttons, map) should be accessible, with clear labels and sufficient contrast for screen readers and visual clarity.
- Font Awesome icons (eye for View, trash for Delete) should be used in the table action buttons.

Simulation Results Page

Simulation Data Display

- Given I am on the simulation results page, I should see the simulation's name, description, and intersections passed from the previous page via react-router-dom state.
- The simulation name should be displayed as a large, bold heading with a teal-to-emerald gradient.
- The description should be a paragraph in gray text.
- Intersections should appear as clickable pills with a teal border and hover effects, styled using Tailwind CSS.
- If no simulation data is available (e.g., no state passed), a "Loading..." message should display in the center.

Simulation Visualization Section

- Given I am on the simulation results page, I should see a "Simulation Visualization" section with:
    - A bar chart displaying speed and density parameters (default: speed=50, density=30) using Chart.js.
    - The chart should have green bars (#34D399, #10B981), no legend, and customized tooltips with a dark background.
    - Axes should display "Parameter" (x) and "Value" (y, 0-100 range) with light grid lines and bold labels.
    - The section should include input fields for adjusting speed and density (number inputs, 0-100 range).

- Input fields should be styled with Tailwind CSS, using a gray background and teal focus ring.
- A "Run" button should toggle a "Running..." state for 2 seconds when clicked, disabling itself during this period.
- An "Optimize" button should adjust optimized parameters (increase speed by 20 up to 100, decrease density by 10 down to 0).
- Clicking the chart should open a full-screen modal displaying the same chart with a title ("Simulation Visualization").

Optimized Visualization Section

- Given I am on the simulation results page, I should see an "Optimized Visualization" section with:
  - A bar chart displaying optimized speed and density parameters (default: speed=70, density=20) using Chart.js.
  - The chart should have blue bars (#3B82F6, #2563EB), no legend, and customized tooltips with a dark background.
  - Axes should match the simulation chart's configuration.
  - The section should include input fields for adjusting optimized speed and density (number inputs, 0-100 range).
  - Input fields should match the simulation section’s styling.
  - A "Run" button should toggle a "Running..." state for 2 seconds when clicked, disabling itself during this period.
  - Clicking the chart should open a full-screen modal displaying the same chart with a title ("Optimized Visualization").

Full-Screen Chart Modal

- Given I click a chart in either visualization section, a full-screen modal should appear with:
  - The selected chart (simulation or optimized) rendered at full size using Chart.js.
  - A title matching the chart type ("Simulation Visualization" or "Optimized Visualization").
  - A close button (cross) in the top-right corner to exit the modal.
  - The modal should have a semi-transparent black background and be centered with a maximum width of 7xl.

- Charts should clean up properly when the modal closes to prevent memory leaks.

Visual and Accessibility Requirements

- The page should include a Navbar at the top and a dynamic Footer at the bottom, consistent with the application's design.
- The page should have a responsive layout, using a two-column grid for medium and larger screens and a single-column layout for mobile.
- The page should use a dark gradient background (gray-900 to black) with light text for readability.
- Charts, inputs, and buttons should be styled with Tailwind CSS, including gradients, shadows, and hover effects.
- Input fields should have accessible labels and be constrained to valid ranges (0-100).
- All interactive elements (buttons, charts, inputs) should have sufficient contrast and be usable with screen readers.

Users Page

Users Table Display

- Given I am on the users page, I should see a table displaying user data with columns for:
  - ID (e.g., "1").
  - Name (e.g., "John Doe").
  - Email (e.g., "email@email.com").
  - Role (e.g., "Admin", "Engineer", "Viewer").
  - Last Login (e.g., "2025-05-13 09:00").
- The table should be rendered using the UsersTable component, passing the current page's user data and handlers for Edit and Delete actions.
- Clicking the Edit button for a user should log "Edit user [ID]" to the console.
- Clicking the Delete button for a user should log "Delete user [ID]" to the console.

Pagination

- Given I am on the users page, I should see pagination controls below the table, including:
    - A "Previous" button with a left arrow icon, disabled on the first page.
    - Numbered page buttons, showing up to 5 pages (first, last, and up to 3 around the current page).
    - An ellipsis ("...") for skipped page numbers when there are more than 5 pages.
    - A "Next" button with a right arrow icon, disabled on the last page.
    - Clicking a page number should display the corresponding user data slice (9 or 7 rows per page, depending on screen size).
    - Clicking "Previous" or "Next" should navigate to the adjacent page, updating the table data.
    - Pagination buttons should be styled with Tailwind CSS, with the active page highlighted in blue and hover effects on others.

Responsive Row Adjustments

- Given I am on the users page, the table should display 9 rows per page by default.
- When the screen size is 1400px or smaller and 800px or shorter, the table should display 7 rows per page, resetting to the first page.
- The row adjustment should occur dynamically when the window size changes, using a media query listener.
- The total number of pages should update automatically based on the number of users and rows per page.

Visual and Accessibility Requirements

- The page should include a Navbar at the top, consistent with the application’s design.
- The page should have a responsive layout, with a maximum width of 6xl (Tailwind CSS) and padding for content.
- The page should use a light background (gray-100) for a clean appearance.
- The table and pagination controls should be styled with Tailwind CSS for consistency and readability.

- Pagination buttons should have accessible labels (e.g., "Previous page", "Next page") and use SVG icons for arrows.
- The table should be accessible, with proper semantic structure and support for screen readers (handled by the UsersTable component).

Authentication

- As a **user**,
  I want to **register, log in, and manage my account**,
  so that I can **use the system securely and privately**.
- As a **user**,
  I want to **reset my password** if I forget it,
  so that I can **regain access** to my account.

Simulation Creation

- As a **user**,
  I want to **create a new simulation** with basic intersection details,
  so that I can **model traffic behaviour**.
- As a **user**,
  I want to **manually configure** traffic flow probabilities and light timings,
  so that I can **experiment with different traffic patterns**.
- As a **traffic department user**,
  I want to **upload real-world traffic data**,
  so that the **system can create simulations based on actual conditions**.

Simulation Insights

- As a **user**,
  I want to **view performance metrics** for each simulation,
  so that I can **understand and evaluate the traffic behaviour**.

Optimization

- As a **user**,
  I want to **optimize my simulation's timing**,
  so that I can **observe what timing's best suit my simulation**
- As a **user**,
  I want to **compare the original and optimized configurations**,
  so that I can **see how the optimization has improved** traffic flow.

Functional Requirements

Requirements

R1: Login/Signup
R1.1: Must allow users to login their existing account details
R1.2: If user does not have an account, allow user to create account
R1.3: Once registered, allow users to login
R1.4: Allow users to switch between light/dark mode.

R2: Dashboard page
R2.1: Include navbar in all pages that allows users to access various pages such as the simulations page.
R2.2: Show statistics such as total intersections, active simulations, traffic volume and top intersections.
R2.3: Show recent simulations as well as their status and allow users to view details about them.
R2.4: Allow users to insert a new intersection, run simulation and view the map.
R2.5: Allow users to logout by clicking the logout button.

R3: Intersections page
R3.1: Allow users to view all listed intersections
R3.2: Allow users to search a specific intersection
R3.3: Allow users to add a new intersection
R3.4: Allow users to be able to simulate, edit and delete a specific intersection

R4: Simulation page
R4.1: Users should be able to view recent simulations and optimizations with information about them such as average wait time and throughput.
R4.2: Allow users to insert a new simulation.
R4.3: Allow users to select a specific intersection for filtering purposes.
R4.4: Allow users to navigate through simulation pages by clicking on 'next', 'prev' or a page number button.

R5: Users Page

R5.1: Users must be able to view all listed users, with their information included.

R5.2: Allow users to be able to navigate through table pages.

R5.3: Users must be able to edit and delete information.

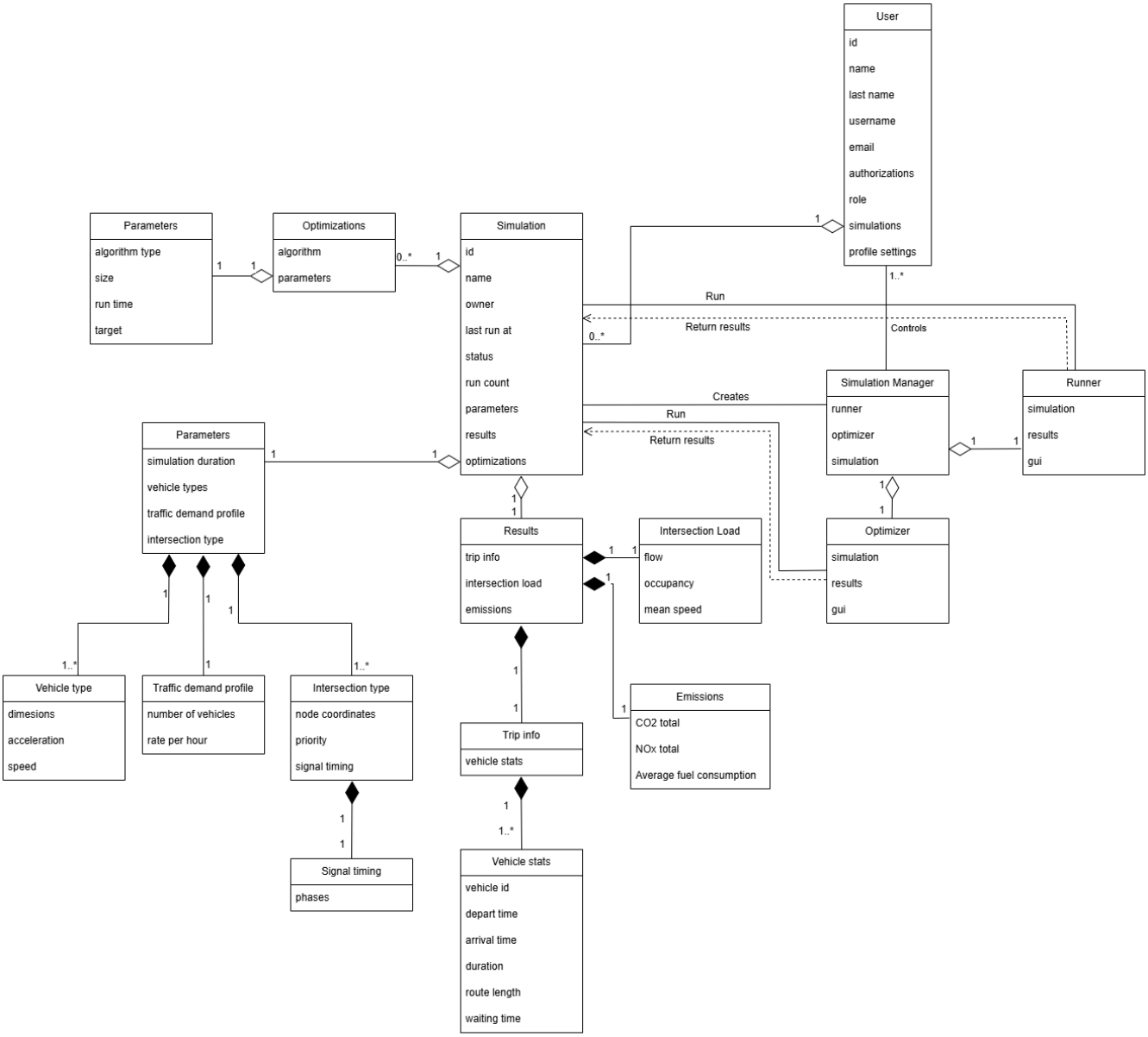
Subsystems

[Insert Subsystems here]

Use Case Diagrams

[Insert Use Case Diagrams corresponding to subsystems above here]

Domain Model



This domain model represents our vision for a comprehensive and optimized traffic simulation system, centring around the Simulation entity which integrates traffic parameters, optimization settings, and results data.

System Architecture

Users can configure and execute simulations through a Simulation Manager System that orchestrates two distinct components:

- Runner: Responsible purely for running and visualizing the simulation, constructed using user input as parameters.
- Optimizer: Applies optimization algorithms (e.g., swarm optimization techniques) to improve simulation outcomes.

This modular separation enables dynamic control and real-time feedback during simulations.

Performance Metrics and Results

The model captures detailed performance metrics for all simulations, including:

- Trip-Level Data: Such as Vehicle Stats (departure time, arrival time, route length, waiting time).
- Intersection-Level Data: Including Intersection Load (flow, occupancy, mean speed).
- Environmental Impact: Captured through Emissions (CO_2 total, NOx total, fuel consumption).

Key Features

- Flexible optimization strategies through parameterized Optimizer settings.
- Comprehensive user interaction and result tracking via the User, Simulation Manager, and Results entities.
- Clear and modular structure, supporting scalability and future enhancements.

Design Principles

The overall structure emphasizes:

- Modularity – distinct responsibilities for each component.
- Scalability – allowing for growth in simulation complexity or optimization scope.
- Clear data flow – ensuring easy traceability of inputs, parameters, and results.

Architectural Requirements Specification

1.1 System Overview

Swift Signals is a comprehensive traffic optimization system designed to analyze traffic patterns, simulate intersection performance, and optimize traffic light timing using machine learning algorithms. The system employs a microservices architecture to ensure scalability, maintainability, and independent deployment of core components including user management, traffic simulation, AI optimization, real-time control, metrics collection, and a web-based frontend interface.

---

2. Quality Requirements

2.1 Performance

The system must deliver high-performance capabilities to handle real-time traffic data processing and simulation workloads. Response times for web interface interactions should not exceed 2 seconds under normal load conditions. The simulation service must be capable of processing complex intersection scenarios with hundreds of vehicles within acceptable timeframes. Machine learning inference for traffic optimization should complete within 30 seconds for single intersection analysis. The system must support concurrent users and simultaneous simulation runs without degradation in performance.

2.2 Scalability

Swift Signals is designed to scale horizontally across multiple dimensions. The microservices architecture enables individual services to scale independently based on demand. The system must support scaling from single intersection analysis to city-wide traffic network optimization. Database systems should handle growing volumes of time-series traffic data and simulation results. The AI service must accommodate training and inference workloads that increase with system adoption. Container

orchestration through Kubernetes ensures efficient resource utilization and automatic scaling capabilities.

2.3 Reliability and Availability

The system requires high availability with a target uptime of 99.5% to support continuous traffic monitoring and optimization. Critical services like the control service and metrics collection must implement redundancy and failover mechanisms. Data persistence layers must include backup and recovery procedures to prevent loss of historical traffic data and trained models. The API gateway should implement circuit breaker patterns to handle service failures gracefully. Health monitoring and alerting systems must provide proactive notification of system issues.

2.4 Security

Security is paramount given the system's potential integration with municipal traffic infrastructure. Authentication and authorization mechanisms must protect against unauthorized access to traffic control functions. API endpoints require rate limiting and input validation to prevent abuse. Sensitive configuration data and model parameters must be encrypted at rest and in transit. The system should implement audit logging for all configuration changes and control actions. Network security policies must isolate internal services from external access points.

2.5 Maintainability

The codebase must support long-term maintenance and evolution through clear separation of concerns and comprehensive documentation. Each microservice should follow established coding standards and include unit and integration tests. Configuration management through YAML files enables environment-specific deployments without code changes. Service interfaces should be well-defined with comprehensive API documentation to facilitate integration and troubleshooting. API versioning strategies ensure backward compatibility during system updates.

2.6 Extensibility

The architecture must accommodate future enhancements and integration with external traffic management systems. New intersection types and traffic patterns should be addable without requiring changes to core simulation logic. The AI service should support multiple machine learning algorithms and model types. Plugin architectures enable third-party integrations for traffic data sources and optimization algorithms. The metrics service should accommodate new performance indicators and reporting requirements.

2.7 Usability

The web interface must provide an intuitive experience for traffic engineers and system administrators. Visualization components should clearly present complex traffic data and simulation results. Configuration workflows should guide users through intersection setup and optimization processes. Real-time dashboards must display system status and traffic conditions effectively. Help documentation and user guides should support users of varying technical expertise levels.

2.8 Interoperability

The system must integrate with existing traffic management infrastructure and data sources. Standard protocols and data formats ensure compatibility with third-party traffic monitoring systems. REST APIs enable integration with municipal traffic control centers. Export capabilities allow sharing of optimization results with external planning tools. The system should support common traffic data formats and simulation standards.

---

3. Architectural Patterns

3.1 Microservices Architecture

Swift Signals employs a microservices architecture to achieve modularity, scalability, and independent deployment capabilities. Each core function is implemented as a separate service with its own data store and deployment lifecycle.

3.1.1 Services:

- API Gateway: Central entry point providing request routing, authentication, and rate limiting

- User Service: Handles authentication and user management, with JWT-based login/signup functionality

- Simulation Service: Traffic simulation engine with SUMO integration for intersection modeling

- Optimization Service: Applies machine learning or heuristic algorithms to generate optimized traffic signal strategies

- Control Service: Acts as a coordinator between the Simulation Service and the AI Service, ensuring smooth data flow and trigger sequencing

- Metrics Service: Collects and serves statistics about simulation runs and AI performance

- Frontend Service: React-based and Vite web interface for system interaction and visualization

3.2 API Gateway Pattern

The API Gateway serves as the single entry point for all client requests, providing essential cross-cutting concerns including request routing, authentication, rate limiting, and HTTP-to-gRPC translation. This pattern simplifies client interactions by presenting a unified REST interface while allowing internal services to communicate efficiently via gRPC. The gateway implements load balancing for downstream services and provides centralized logging and monitoring capabilities.

3.3 Database Per Service Pattern

Each microservice maintains its own database to ensure loose coupling and independent scaling. The User Service utilizes MongoDB for flexible user profile storage, while the Metrics Service employs PostgreSQL for time-series data. This pattern prevents database-level coupling between services and allows for technology choices optimized for specific data access patterns.

3.4 Event-Driven Architecture

Services communicate through asynchronous events using gRPC streaming capabilities to reduce coupling and improve system resilience. The simulation service publishes events when simulations complete, triggering downstream processing in the AI and metrics services via efficient binary Protocol Buffer messages. This pattern enables loose coupling between services while maintaining high-performance communication channels and supports eventual consistency across the system.

3.5 CQRS (Command Query Responsibility Segregation)

The metrics service implements CQRS to separate read and write operations for optimal performance. Historical traffic data writes are optimized for ingestion speed, while query operations are optimized for complex analytics and reporting. This separation allows for independent scaling of read and write workloads.

---

4. Design Patterns

4.1 Factory Pattern

The AI service employs the Factory pattern to create appropriate machine learning models based on intersection types and optimization requirements. This pattern enables runtime selection of algorithms and supports extensibility for new optimization approaches.

4.2 Observer Pattern

The control service implements the Observer pattern to notify interested parties when traffic light configurations change. This enables real-time updates to the frontend interface and logging of configuration changes for audit purposes.

4.3 Strategy Pattern

Traffic simulation scenarios use the Strategy pattern to support different intersection types and traffic patterns. This pattern allows the simulation service to handle various intersection configurations through pluggable strategy implementations.

4.4 Command Pattern

Configuration changes to traffic light timing are implemented using the Command pattern, providing undo capabilities and audit trails. This pattern enables queuing of configuration changes and rollback functionality for testing scenarios.

4.5 Singleton Pattern

Shared resources like database connection pools and configuration managers are implemented as singletons to ensure efficient resource utilization and consistent configuration across service instances.

4.6 Decorator Pattern

The API gateway uses the Decorator pattern to add cross-cutting concerns like authentication, logging, and rate limiting to request processing pipelines. This pattern enables flexible composition of middleware components.

4.7 Circuit Breaker Pattern

Inter-service gRPC communication implements the Circuit Breaker pattern to handle failures gracefully and prevent cascade failures. This pattern improves system resilience by failing fast when downstream services are unavailable and provides automatic retry mechanisms with exponential backoff for transient failures.

---

5. Constraints

5.1 Technology Constraints

- Open Source Software Only: All system components must use open-source technologies and libraries to comply with project constraints

- Containerization: All services must be containerized using Docker for consistent deployment across environments

- Kubernetes Orchestration: Production deployments must use Kubernetes for container orchestration and service management

- gRPC Communication: Inter-service communication must use gRPC protocols with Protocol Buffer serialization for high-performance, type-safe communication between microservices

5.2 Infrastructure Constraints

- Cloud-Native Design: The system must be designed for cloud deployment with Southern Cross Solutions providing infrastructure resources

- Container Resource Limits: Each service must operate within defined CPU and memory constraints to ensure efficient resource utilization

- Network Security: Internal service communication must be secured through service mesh or VPN technologies

- Storage Limitations: Database storage must be planned for cost-effectiveness while meeting performance requirements

5.3 Development Constraints

- Independent Deployment: Each microservice must be independently deployable without affecting other services

- Automated Testing: All services must include comprehensive unit and integration tests with minimum 80% code coverage

- CI/CD Pipeline: Automated build, test, and deployment pipelines must be implemented using GitHub Actions

- Configuration Management: Environment-specific configurations must be externalized and managed through YAML files

5.4 **Operational Constraints**

- Monitoring and Logging: All services must implement structured logging and expose health check endpoints

- Data Retention: Historical traffic data must be retained according to specified policies while managing storage costs

- **Backup and Recovery:** Critical data must be backed up with defined recovery time objectives

- **Security Compliance:** System must implement security best practices including secret management and access controls

5.5 Performance Constraints

- **Response Time:** Web interface interactions must complete within 2 seconds under normal load

- **Simulation Performance:** Traffic simulations must complete within acceptable timeframes based on intersection complexity

- **Concurrent Users:** System must support multiple concurrent users without performance degradation

- **Data Processing:** Time-series data ingestion must keep pace with real-time traffic monitoring requirements

5.6 Integration Constraints

- **Third-Party APIs:** Integration with external traffic data sources must handle rate limits and API versioning

- **SUMO Compatibility:** Simulation service must maintain compatibility with SUMO traffic simulation software

- **Standard Protocols:** External integrations must use industry-standard protocols and data formats

- **Backward Compatibility:** API changes must maintain backward compatibility for existing integrations

Technology Requirements

1. System Overview

The system is built with a microservices architecture using modern technologies to ensure scalability, maintainability, and performance.

---

2. Technology Stack

2.1 Frontend

- **Framework**: React
- **Build Tool**: Vite
- **Language**: TypeScript
- **Styling**: Tailwind CSS

2.2 API Gateway

- **Language**: Go
- **Function**: Handles request routing, load balancing, authentication validation, and service aggregation
- **Communication**: HTTP/gRPC

2.3 User Authentication Service

- **Language**: Go
- **Authentication**: JWT-based login and signup
- **Database**: MongoDB

2.4 Simulation Service

- **Language**: Go and Python (integrating SUMO)
- **Simulation Engine**: SUMO (Simulation of Urban Mobility)
- **Function**: Executes traffic simulations with configurable intersection parameters

2.5 Optimization Service

- **Language**: Go
- **Techniques**: Swarm optimization algorithms
- **Function**: Optimizes traffic signal timings based on simulation data

2.6 Metrics Service

- **Language**: Go
- **Function**: Collects and serves system and simulation metrics (e.g., duration, success rate, resource usage)
- **Database**: Prometheus with local time-series storage

2.7 Controller Service

- **Language**: Go
- **Function**: Coordinates data flow between the Simulation and Optimization Services

---

3. Communication

- **Protocol**: gRPC is used for internal communication between microservices
- **API Gateway Interface**: REST over HTTP, serving JSON for client communication
- **API Documentation**: Defined via .proto files (Protocol Buffers) and documented in the Service Contracts wiki page

---

4. Development and Deployment

4.1 Local Development

- **Platform**: Minikube
- **Containerization**: Docker
- **Orchestration**: Kubernetes

4.2 Target Deployment Environments

- **Option 1**: On-premises deployment on the client's local infrastructure
- **Option 2**: Cloud-hosted deployment on platforms such as GCP, AWS, or DigitalOcean

Deployment Requirements:

- Kubernetes support
- Docker Registry integration
- Load balancing and ingress controller support

---

5. Observability

5.1 Logging

- **In-Service Logging**: Zap or Logrus (for Go), and standard Python logging
- **Log Aggregation**: Grafana Loki
- **Log Collection**: *Promtail* or *Fluent Bit*

5.2 Monitoring and Dashboards

- **Monitoring Tool**: Prometheus
- **Dashboards**: Grafana

- **Metrics Exporting**: *prometheus/client_golang* library for exposing Go service metrics

---

6. Design Principles

- **Open-Source Compliance**: All components rely on open-source technologies (e.g., SUMO, Prometheus, Grafana)
- **Portability**: The system is easily deployable on any Kubernetes-compatible environment
- **Extensibility**: The architecture supports easy extension with new AI modules, intersection types, or services
- **Security**: Uses JWT authentication, role-based access control, and secure gRPC communication (with optional TLS support)

Service Contracts

1. Overview

Each service in the Swift Signals architecture owns its own logic and data. Communication between services is done using gRPC, with well-defined contracts to allow independent development and deployment.

---

2. Microservice Input/Output Overview

2.1 User Authentication Service

Description: Handles user registration, login, session validation, and admin controls.

Inputs:

- SignupRequest:
  - email (string): User's email address
  - password (string): Plain-text password
- LoginRequest:
  - email (string): User's email address
  - password (string): Plain-text password
- WhoAmIRequest: Authenticated request using JWT token
- BanUserRequest:
  - user_id (string): Unique ID of the user to be banned

Outputs:

- SignupResponse:
  - user_id (string): Unique ID of the newly registered user
  - jwt_token (string): JWT authentication token
- LoginResponse:
  - jwt_token (string): JWT authentication token
- UserInfoResponse:
  - user_id (string): ID of the currently logged-in user
  - role (string): Role assigned to the user (e.g., "user", "admin")
- BanUserResponse:
  - success (boolean): Whether the user was successfully banned

---

2.2 Simulation Service

Description: Manages traffic simulation execution using SUMO and stores simulation parameters and results.

Inputs:

- CreateSimulationRequest:
  - user_id (string): ID of the user
  - intersection_type (enum): 3-way, 4-way, etc.
  - coordinates (object): Intersection layout
  - light_timing (object): Initial signal timing
  - traffic_flow (object): Number of vehicles per time unit
  - time_series_data (optional, file): Uploaded dataset
- GetSimulationsRequest:
  - user_id (string): User ID to fetch simulations
- GetSimulationByIdRequest:
  - simulation_id (string): Unique simulation ID

Outputs:

- SimulationResponse:
  - simulation_id (string): ID of the created simulation
  - status (string): Simulation status ("running", "complete", etc.)
- GetSimulationsResponse:
  - simulations (list): List of simulations with metadata
- GetSimulationByIdResponse:
  - simulation_id (string): Simulation ID
  - status (string): Status of the simulation
  - parameters (object): Input parameters used for the simulation

---

2.3 Optimization Service

Description: Optimizes traffic signal configurations using AI or heuristic algorithms.

Inputs:

- OptimizeRequest:
  - simulation_id (string): ID of the simulation to optimize
- StatusRequest:
  - simulation_id (string): ID to check optimization progress

Outputs:

- OptimizeResponse:
  - optimized_simulation_id (string): ID of the new optimized simulation
  - status (string): Queue status ("queued", "running", "done")
- StatusResponse:
  - status (string): Optimization progress
  - message (string): Optional error or status message

2.4 Metrics Service

Description: Collects simulation and optimization performance data and makes metrics available for visualization.

Inputs:

- Automatic ingestion via Prometheus scraping exposed endpoints
- No explicit RPC inputs (uses standard exporters)

Outputs:

- Prometheus-compatible metrics exposed via /metrics endpoint, including:
  - Simulation duration
  - Resource usage
  - Optimization success rate

2.5 Controller Service

Description: Orchestrates the optimization process by coordinating between the Simulation and Optimization services.

Inputs:

- simulation_id (string): ID used to fetch simulation data and start optimization

Outputs:

- Notifies Simulation Service when optimization results are ready
- Returns optimization completion status to API Gateway

2.6 API Gateway

Description: Acts as the system's external entry point. Handles HTTP requests from the frontend and converts them into gRPC calls.

Endpoints:

- POST /auth/signup

    - **Input**: email, password
    - **Output**: userId, token
- POST /auth/login
    - **Input**: email, password
    - **Output**: token
- GET /simulations
    - **Input**: JWT token in header
    - **Output**: List of user simulations
- POST /simulations
    - **Input**: intersectionType, coordinates, lightTiming, trafficFlow, timeSeriesData
    - **Output**: Simulation ID and status
- POST /simulations/{id}/optimize
    - **Input**: Simulation ID in path
    - **Output**: Optimized simulation ID and queue status

---

3. Contract Format

All internal service interfaces are defined using .proto files, which describe the services, RPC methods, and message structures. These are compiled to generate client/server code in each language used (Go, Python, etc.).