Swift Signals

Technology Requirements

Swift Signals Technology Requirements & Justification

This document provides a detailed overview of the technologies, tools, and environments used to develop and deploy the Swift Signals traffic optimization platform, alongside clear justifications based on system quality requirements.

System Overview

Swift Signals is a web-based, microservices-driven traffic simulation and optimization platform for traffic planners and municipalities. It enables intersection configuration, simulation, and AI-driven signal optimization, designed to meet strict performance, scalability, and maintainability goals.

Technology Stack and Architecture

Frontend:

- **Framework:** React
- **Build Tool:** Vite
- **Language:** TypeScript
- **Styling:** Tailwind CSS
- **Justification:**
    - Provides fast, responsive, maintainable UI; ensures consistent styling and seamless user experience.

API Gateway:

- **Language:** Go
- **Functions:**
    - Routes requests, handles load balancing, validates authentication, aggregates services.
- **Communication:** HTTP/gRPC
- **Justification:**
    - Go's performance and concurrency handle high traffic loads; gRPC enables efficient secure inter-service communication.

User Authentication Service:

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

- JWT ensures secure, scalable user sessions while PostgreSQL provides scalable storage for structured data.

Intersection Service:

- **Language:** Go
- **Function:** Stores intersection information
- **Database:** MongoDB
- **Justification:**
    - MongoDB enables the storage of a large volume of intersection information, and gives flexibility in the way information is stored by the use of JSON documents.

Simulation Service:

- **Language:** Python (integrating SUMO)
- **Function:** Executes realistic, configurable traffic simulations
- **Justification:**
    - SUMO offers detailed, accurate modelling of urban traffic.

Optimization Service:

- **Language:** Go
- **Techniques:** Automatic parameter tuning and Genetic Programming algorithms
- **Function:** Optimizes traffic signal timings based on simulation data
- **Justification:**
    - Automatic parameter tuning and Genetic Programming enable efficient search for optimal signal patterns under dynamic traffic conditions.

Metrics Service:

- **Language:** Go
- **Function:** Collects system and simulation metrics
- **Database:** PostgreSQL with Prometheus local time-series data
- **Justification:**
    - Prometheus enables scalable real-time metrics collection crucial for observability.

Communication Protocols

- **Inter-service Communication:** gRPC
- **External API Interfaces:** REST with JSON

- **API Specifications:** Protobuf definitions maintained in Service Contracts.

Development and Deployment

- **Local Deployment:** Minikube, Docker, Kubernetes, Tilt/Skaffold for hot-reload
- **Production Environments:**
  - On-premises servers
  - Cloud platforms (GCP, AWS, DigitalOcean)
- **Deployment Features:** Kubernetes orchestration, Docker Registry, load balancing ingress controllers

---

Observability and Monitoring

Logging

- **In-Service:** Zap (GO), Logrus (Go), Python Logging
- **Aggregation:** Grafana Loki
- **Collection:** Promtail or Fluent Bit

Monitoring

- **Metrics:** Prometheus
- **Dashboards:** Grafana
- **Exporters:** prometheus/client.golang for Go services

Design Principles

- **Open Source:** Built on open-source tools (SUMO, PostgreSQL, MongoDB)
- **Portability:** Compatible with any Kubernetes-compliant environment
- **Extensibility:** Supports new intersection models, AI techniques, metrics
- **Security:** JWT authentication, RBAC, secure gRPC with optional TLS

Quality Requirements & Technology Justification

Performance

- **Requirement:** Complex simulations; UI response < 2s; Optimization < 30s
- **Technology Justification:** Go's efficiency; SUMO's realism; React + Vite's fast frontend

Scalability

- **Requirement:** Scale from single intersections to city-wide; handle large datasets
- **Technology Justification:** Microservices, Kubernetes, MongoDB, Prometheus enable elastic scaling

Maintainability

- **Requirement:** Independent service updates; Iterative improvements
- **Technology Justification:** Isolated microservices; Strong typing with Go/TypeScript; Tailwind for UI consistency

Security

- **Requirement:** Robust authentication; Secure service communication
- **Technology Justification:** JWT for secure sessions; gRPC for efficient, encrypted inter-service calls

Responsiveness

- **Requirement:** Fast, seamless user interaction
- **Technology Justification:** React, Vite, and Tailwind ensure lightweight, high-speed UI

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

Swift Signals leverages modern, proven technologies to meet stringent quality demands for performance, scalability, maintainability, and security in traffic optimization.