CRANCSTORM

Technological Feasibility

Team Members:
Braydon Lamoreaux
Kristopher Thomas
Ethan Ferguson
Noelia Canela

Mentor:
Bailey Hall

Table of Contents

1. Introduction

1.1 Context

Promoting bicycling in urban areas is an important transportation strategy for addressing societal concerns such as physical inactivity, worsening air quality, and rising motor vehicle costs. The adoption of bicycling for commuting and utilitarian travel in the U.S. depends on identifying and mitigating safety barriers in bike networks. The CRANC tool was designed to provide bicyclists with optimized routes based on individual physical abilities, infrastructure availability, and perceived safety.

1.2 Problem

Current bicycling infrastructure analysis is limited by static maps, inconsistent routing recommendations, and delayed user feedback. Sponsors Dr. Steven R. Gehrke and David Wessel require a solution that integrates real-time routing recommendations with data from user observations. Existing systems are insufficient to provide dynamic, evidence-based route guidance and planning insights for transportation agencies.

1.3 Goal

The project aims to improve the CRANC tool's web interface and develop a mobile app with GPS logging, linking person-level attributes to observed bike routing decisions.

Key features include:

1. Updated web interface using General Modeling Network Specification for standardized routing.
2. Expansion of study areas to cover multiple southwestern states.
3. Improved isochrone visualization with accessible point-of-interest details.
4. Mobile apps support iOS and Android with user profiles and GPS logging.
5. Audio and text-based turn-by-turn navigation

1.4 Impact

CRANC will enable transportation agencies to provide personalized route recommendations, collect high-resolution usage data, and support evidence-based planning for bicycle infrastructure investments. Agencies will gain actionable insights to improve safety, accessibility, and network connectivity. This project equips planners with tools to encourage wider bicycling adoption, reduce barriers, and enhance sustainable transportation planning.

Having established the context, goals, and anticipated impact of the CRANC project, we now turn to the key technological challenges that must be addressed to realize these objectives. The following sections identify the major areas where design decisions will significantly influence project success, and set the stage for evaluating potential solutions, exploring alternatives, and selecting the most promising approaches for implementation.

2. Technological Challenges

Through our initial project exploration and requirements gathering, we have identified the following major technological challenges that are critical to project success:

Challenge 1: Real-Time Routing Algorithm Computation – Efficient computation of routes for varying user profiles.

Challenge 2: Web Interface Upgrades and Isochrone Visualization – Improving usability and accessibility of the mapping platform.

Challenge 3: Mobile App GPS Logger and User Profiles – Collecting Spatiotemporal data while maintaining user engagement.

Challenge 4: Cross-Platform Navigation System – Providing accurate audio and text directions on iOS and Android.

Challenge 5: Data Integration and Standardization – Ensuring network data conforms to General Modeling Network Specification.

Challenge 6: Scalability and Performance Optimization – Supporting multiple users and large datasets efficiently.

The following sections provide detailed analysis of each challenge, including evaluation criteria, alternative solutions considered, our investigation methodology, and our chosen approach with justification.

3. Technology Analysis

3.1 Real-Time Routing Algorithm Computation

Issue Introduction

A critical part of our bike navigation system is the ability to create and update routes in real time based on cycling-friendly data. Unlike car navigation, bike routes are harder to find because they require bike lanes and try to avoid high traffic roads.

Desired Characteristics

For a high-quality bike navigation experience, the ideal routing solution must support:

1. **Bike-friendly route computation**: Cyclists require safe and dedicated bike infrastructure to avoid dangerous roads and ensure a comfortable ride. Without bike-friendly routing, users may be directed onto highways or busy streets, making the app unsafe and unreliable.
2. **Custom bike profiles**: Different types of bikes and rider preferences affect feasible routes. For example, e-bike riders may handle hills differently than standard bikes, and our app needs to provide personalized routes that suit each rider's abilities and vehicle type.
3. **Dynamic re-routing**: Cyclists may deviate from the planned route or encounter temporary obstacles such as construction or road closures. Dynamic re-routing ensures that the app can respond in real time and keep the user on an optimal path.
4. **Low-latency performance:** Routes must update quickly, especially when the user changes course or when traffic or weather conditions affect the path. Slow route computation would create a frustrating user experience and reduce trust in the app.
5. **Elevation-aware routing:** Cyclists are sensitive to elevation changes, which can significantly affect effort and ride time. Including elevation in route planning allows the app to recommend routes that are feasible and comfortable for the rider.
6. **Open-source or affordable pricing:** Our project budget and licensing considerations favor solutions that are free or low-cost. Open-source tools also allow for more customization, which is critical for meeting our specific bike-routing requirements.
7. **Ease of integration with our app/backend:** The routing engine must seamlessly integrate with our existing application architecture. If integration is difficult or requires extensive workarounds, it could delay development and reduce overall project efficiency.

Alternatives

Alternative 1: Valhalla is an open-source routing engine developed by Mapzen in 2015 and maintained by the Linux Foundation. It supports multimodal routing with detailed bicycle infrastructure. We discovered it through research into open-source routing engines for cycling applications and GitHub documentation. Valhalla offers customizable bike profiles, elevation-aware routing, and low-cost deployment.

Alternative 2: GraphHopper, developed in 2012, is a Java-based open-source routing engine. It supports cycling routes, cloud-hosted or self-hosted deployment, and customizable profiles. We found GraphHopper via searches for Java-friendly routing engines and references in cycling and logistics forums. It balances performance and flexibility for moderate-scale projects.

Alternative 3: OSRM (Open-Source Routing Machine) was developed in 2010 at Karlsruhe Institute of Technology. Designed for high-performance routing with bike, car, and pedestrian profiles, it is widely used in speed-critical applications. We discovered OSRM through OpenStreetMap community recommendations. While very fast, it is less flexible for complex bike profiles and elevation-aware routing.

Alternative 4: Google Maps Directions API is a commercial routing solution launched in 2005. It provides highly accurate routes and live traffic data. We evaluated it as an industry benchmark and found documentation and examples on Google Developers. Google Maps has limited customization for specialized cycling needs and usage-based pricing, but it offers reliable dynamic routing and low latency.

Analysis

Valhalla scored highest overall because it best matched the weighted desired characteristics: bike-friendly routing, elevation awareness, customizable profiles, and open-source licensing. GraphHopper performed well in ease of integration and cost but slightly lagged in bike-specific routing features. OSRM offered the fastest routing but lacked advanced bike customization. Google Maps provided excellent dynamic re-routing and low latency but had limited customization for cycling and higher costs.

Based on these comparisons and the weighted scoring table below, Valhalla was selected as the most suitable solution for our bike navigation system, balancing performance, flexibility, and project feasibility.

Chosen Approach

<u>Table Explanation:</u>

Each characteristic in the table below was assigned a weight based on its importance to CRANC's performance objectives, with the sum of all weights equaling 1.0. Alternatives were rated on a scale from **1 (poor)** to **5 (excellent)** for each characteristic, based on their documented performance, architecture design, and implementation feasibility.

Characteristic	Weight	Valhalla	GraphHopper	Google Maps	OSRM
Bike Friendly Route Computing	.25	5	4	3	3
Custom Bike Profiles	.20	5	4	2	1
Dynamic Rerouting	.10	4	4	5	4
Low-Latency	.15	4	4	5	4
Elevation-aware routing	.15	5	5	2	3
Open Source or Affordable Pricing	.10	5	5	5	2
Ease of integration	.05	4	5	4	5
Weighted Score	---------	4.70	4.35	3.40	2.80

Decision: Valhalla Routing Engine

Valhalla achieved the highest overall weighted score (4.70), providing the most balanced combination of accuracy, flexibility, and affordability. Its open-source nature, elevation awareness, and customizable profiles make it the most suitable routing framework for CRANC's real-time cycling navigation. While GraphHopper and OSRM deliver strong performance, Valhalla's bike-specific optimization and seamless backend integration justify its selection.

Proving Feasibility

To validate Valhalla as our real-time routing solution, we conducted initial tests using small OpenStreetMap datasets to generate bike-friendly routes with custom profiles. These tests confirmed that Valhalla can compute routes dynamically, consider elevation data, and respect bike-specific lanes. Moving forward, we plan to expand these tests using larger city-scale maps and real-world cycling scenarios. As part of the upcoming Technology Demo assignment, we will demonstrate live route generation, dynamic re-routing when a user deviates, and integration with our app backend, ensuring that the chosen solution performs effectively under the conditions expected in our project.

3.2 Isochrone Visualization Upgrade

Issue Introduction:

The web interface and isochrone visualization need to be more usable for individuals with limited technical experience and easier to interpret visually. The previous implementation already stores data for stores and other points of interest (POIs) within the isochrone range. Therefore, the primary improvement required is to display these POIs as interactive blips, making the visualization more informative and actionable for users.

Desired Characteristics

For this project, we have identified the following key characteristics that an ideal solution must provide:

1. **Ease of use:** The interface must be intuitive and accessible to users with limited technical experience, ensuring that all users can quickly interpret travel information.
2. **Clear visualization:** Isochrones or alternative visualizations should clearly communicate travel time or distance, allowing users to make informed decisions at a glance.
3. **Filterable POIs:** Users should be able to toggle different types of POIs, enabling them to focus on relevant destinations and reduce visual clutter.
4. **Responsive design:** The visualization should function smoothly across desktop and mobile browsers, maintaining usability regardless of device.

5. **Integration compatibility:** The solution must integrate seamlessly with the CRANC routing system and backend architecture to ensure real-time interactivity.

Alternatives

Alternative 1: A *heatmap visualization* uses color gradients to represent data density or intensity over a geographic area. This technique is popular in urban mobility analytics and stress mapping. We discovered its use in systems such as Strava's Global Heatmap and Google's Traffic layers. A heatmap can visually communicate areas of high or low activity through color, which can make data interpretation intuitive. However, while effective for showing relative density or traffic stress, it can be less precise when users want to see exact POIs or travel boundaries.

Alternative 2: A Clustered Points of Interest (POI) Map groups from visible destinations into clusters to emphasize density and simplify visualization. This approach is used in platforms such as Map boxes and OpenStreetMap to reduce clutter when multiple markers overlap. It provides a balance between visual simplicity and information density, allowing users to zoom in for more details.

Alternative 3: The isochrone visualization approach highlights areas reachable within a certain distance or time from a central point. This method, used by services like Mapbox Isochrone API and OpenRoute Service, provides a clear polygon overlay showing accessible regions. The main limitation of the existing implementation was the lack of clear POI markers and filtering. By enhancing the base isochrone with filterable POIs, legend controls, and hover tooltips, we can maintain precision while improving usability.

Analysis

The heatmap visualization offered an intuitive sense of density but lacked precision for travel-time interpretation, limiting its usefulness for decision-making. The clustered POI map improved readability by reducing visual clutter, but it did not clearly communicate travel ranges or accessibility. In contrast, the enhanced isochrone visualization displayed reachable areas accurately and allowed users to filter POIs dynamically. It is also integrated well with the CRANC routing system.

Based on these comparisons and the weighted scoring table below, the **enhanced isochrone visualization** achieved the highest overall rating, particularly excelling in clarity, filterability, and integration compatibility. Although more complex to implement, its advantages in usability and accuracy made it the most suitable option for CRANC.

Chosen Approach

Table Explanation:

Each characteristic in the table below was assigned a weight based on its importance to CRANC's performance objectives, with the sum of all weights equaling 1.0. Alternatives were rated on a scale from **1 (poor)** to **5 (excellent)** for each characteristic, based on their documented performance, architecture design, and implementation feasibility.

Characteristic	Weight	Heatmap	Clustered POI Map	Enhanced Isochrone Visualization
Ease of use	.20	4	5	4
Clear visualization	.25	3	4	5
Filterable POIs	.20	2	4	5
Responsive design	.15	5	5	4
Integration Compatibility	.20	3	4	5
Weighted Score	-------	3.45	4.45	4.75

Decision: Enhanced Isochrone Visualization

The **enhanced isochrone visualization** received the highest overall score (4.75). It provides the clearest representation of reachable areas while supporting POI filtering and integration with CRANC's routing system. The clustered POI map scored well for usability but lacked range representation. The heatmap, though visually engaging, was less precise for user decision-making.

Proving Feasibility

To validate the enhanced isochrone visualization, we performed initial testing using existing CRANC datasets with POIs such as stores and public facilities. These tests confirmed that POIs can be displayed accurately within isochrone polygons and filtered by type. Moving forward, we plan to conduct larger-scale tests with real-world scenarios, ensuring responsiveness on desktop and mobile devices. As part of the upcoming Technology Demo assignment, we will demonstrate interactive POI filtering, hover tooltips, and real-time integration with the CRANC routing system to confirm usability and accuracy.

3.3 Mobile App GPS Logger and User Profiles

Issue Introduction

Our client has requested the implementation of a **mobile GPS logging system** and **user profile database** for CRANC to enable user authentication, data tracking, and behavioral analysis. This functionality will allow the system to collect and analyze valuable cycling data—such as routes taken, ride duration, speed, and preferred paths—to improve model accuracy and personalize route recommendations. The mobile app must securely log user data, function across multiple platforms, and integrate seamlessly with CRANC's routing and modeling framework.

Desired Characteristics

To ensure the mobile GPS and profile system meets project goals, the ideal solution should provide:

1. **Secure authentication:** Protects user privacy through encrypted credentials and secure login mechanisms, ensuring compliance with modern data security standards.
2. **Cross-platform compatibility:** Must function reliably on both Android and iOS devices, allowing broad adoption among cyclists.
3. Accurate GPS tracking: Captures high-resolution spatiotemporal data while minimizing battery consumption to ensure reliable, continuous location capture.
4. **Data synchronization:** Efficiently manages high volumes of GPS data and user profiles, maintaining consistency between devices and backend systems.
5. **Integration with routing system:** The GPS data must seamlessly feed into CRANC's routing algorithms and General Modeling Network Specification (GMNS) pipeline to enhance route prediction and personalization over time.

Alternatives

Alternative 1: Firebase, developed in 2011 and acquired by Google in 2014, is a leading Backend-as-a-Service (BaaS) platform offering real-time databases, authentication, and hosting. We identified Firebase through developer documentation and GitHub discussions. It provides rapid prototyping and strong real-time synchronization features. However, reviews from developer forums note that costs can rise significantly at scale and that vendor lock-in can be a long-term concern.

Alternative 2: AWS Amplify, launched in 2017, is Amazon's integrated framework for building cloud-connected applications. We discovered Amplify through AWS documentation, Stack Overflow discussions, and GitHub samples. It simplifies access to AWS services like Cognito for authentication, AppSync for APIs, and Amazon Location Service for accurate geospatial tracking. Although it has a steeper learning curve than Firebase, developers report superior scalability, cost predictability, and robust GPS accuracy—especially in urban environments.

Alternative 3: Supabase, introduced in 2020, is an open-source alternative to Firebase built atop PostgreSQL. Paired with PostGIS (a spatial database extension from 2001), it enables advanced geographic data handling. We found Supabase through open-source developer blogs and documentation. While it excels at precise geospatial analysis, reviews highlight that its mobile SDKs are still maturing, and setup requires more SQL expertise, potentially slowing development for smaller teams.

Alternative 4: A fully custom backend using Node.js, MongoDB, and Socket.io provides complete flexibility and control over system design. We explored this option based on prior development experience and open-source examples. It offers powerful real-time communication for GPS tracking and avoids vendor lock-in. However, it requires extensive DevOps effort and ongoing maintenance compared to managed services like AWS or Firebase.

Analysis

The team analyzed each alternative through documentation, community discussions, user reviews, and technical blog evaluations. AWS Amplify achieved the highest overall weighted score due to its exceptional GPS accuracy, seamless integration capabilities, and robust authentication via AWS Cognito. Firebase performed well in data synchronization and ease of implementation but fell short in integration with CRANC's routing system. Supabase offered excellent spatial data precision with PostGIS but demanded greater SQL expertise and additional SDK maturity before production deployment. The custom backend, while powerful, required excessive setup time and maintenance, reducing development efficiency.

Based on these findings and the weighted table below, **AWS Amplify with Amazon Location Service** was selected as the best fit for CRANC's mobile GPS and user profile subsystem, balancing scalability, accuracy, and integration compatibility.

Chosen Approach

Table Explanation:

Each characteristic in the table below was assigned a weight based on its importance to CRANC's performance objectives, with the sum of all weights equaling 1.0. Alternatives were rated on a scale from **1 (poor)** to **5 (excellent)** for each characteristic, based on their documented performance, architecture design, and implementation feasibility.

Characteristic	Weight	Firebase	AWS Amplify	Supabase	Custom Backend
Secure Authentication	.15	5	5	4	3
Cross-Platform Compatibility	.20	5	4	3	3
Accurate GPS Tracking	.20	4	5	4	4
Data Synchronization	.15	5	4	3	3
Integration with Routing System	.30	3	5	4	4
Weighted Score	-------	4.15	4.55	3.65	3.50

Decision: AWS Amplify with Amazon Location Service

AWS Amplify earned the highest weighted score (4.55), largely due to its superior integration capabilities, GPS precision, and scalability. Its use of **Amazon Location Service** offers strong map-matching accuracy, which is particularly beneficial for urban cycling environments. By standardizing on AWS infrastructure, CRANC can unify its backend and mobile components under one ecosystem, improving maintainability and reducing integration overhead.

Proving Feasibility

To confirm the feasibility of AWS Amplify with Amazon Location Service, initial proof-of-concept testing was performed using sample GPS datasets to validate real-time logging accuracy and location updates. The integration with AWS Cognito verified secure user authentication and synchronization between multiple test devices. Future development will expand testing to full route logging and live tracking scenarios integrated with CRANC's routing engine. These demos will form the foundation for the **Technology Demo** assignment, demonstrating authenticated user sessions, dynamic GPS data capture, and synchronization with the backend routing model.

3.4 Cross-Platform Navigation System

Issue Introduction

The CRANC software requires a cross-platform navigation system capable of providing real-time, turn-by-turn guidance across mobile and web environments. Currently, navigation features are limited to desktop use, which restricts usability for riders testing routes in the field. The new system must allow seamless operation on Android, iOS, and desktop browsers, maintaining consistent performance, interface design, and integration with CRANC's routing backend. This enhancement will enable on-the-go navigation, improve accessibility, and support the collection of live feedback for model improvement.

Desired Characteristics

For the navigation system to meet project goals, the ideal solution must support:

1. **Cross-Platform Compatibility:** Must operate seamlessly on Android, iOS, and desktop environments
2. **Offline Functionality:** Should allow navigation when users are offline or in areas with poor connectivity
3. **Accurate Turn-by-Turn Guidance:** Must support real-time directions, rerouting, and location tracking
4. **User Interface Consistency:** Deliver a uniform, intuitive experience across devices to simplify usability and reduce learning curves.
5. **Ease of Integration:** The navigation experience should look and function similarly across all devices

Alternatives

Alternative 1: Mapbox provides full-featured navigation SDKs for Android and iOS that include offline capabilities, real-time routing, and customizable UI components. It is well-documented, widely adopted in mobile navigation apps, and integrates effectively with OpenStreetMap data. However, Mapbox's subscription-based pricing model may present scalability challenges as usage increases.

Alternative 2: Google Maps offers mature SDKs and exceptionally accurate routing backed by real-world traffic data. While its turn-by-turn navigation is highly refined, it lacks true offline support and does not allow for deep backend integration due to its closed-source environment. Its API usage costs also scale with request frequency, making it less cost-effective for continuous routing or data collection.

Alternative 3: Flutter, an open-source framework by Google, enables development for Android, iOS, web, and desktop using a single codebase. When paired with Mapbox or OpenStreetMap plugins, it provides flexible navigation, offline map support, and UI consistency across all platforms. Although it introduces slight performance trade-offs compared to native SDKs, Flutter's unified development environment and integration flexibility make it ideal for CRANC's long-term deployment goals.

Analysis

The Flutter + Mapbox/OSM solution scored the highest overall because it provides the best balance between cross-platform coverage, ease of integration, and user interface consistency. Mapbox's native SDKs performed well in navigation accuracy and offline reliability but lack desktop support, making them less suitable for a multi-environment solution. Google Maps excelled in precision and latency but failed to meet offline and open-integration requirements.

Based on these comparisons and the weighted scoring table below, the Flutter + Mapbox/OSM hybrid approach was selected as the most appropriate choice, aligning well with CRANC's development goals for unified mobile and web navigation.

Chosen Approach

Table Explanation:

Each characteristic in the table below was assigned a weight based on its importance to CRANC's performance objectives, with the sum of all weights equaling 1.0. Alternatives were rated on a scale from **1 (poor)** to **5 (excellent)** for each characteristic, based on their documented performance, architecture design, and implementation feasibility.

Characteristic	Weight	Mapbox Navigation SDK	Google Maps Platform	Flutter + Mapbox/OSM
Cross-Platform Compatibility	0.25	4	3	5
Offline Functionality	0.20	5	2	4
Accurate Turn-by-Turn Guidance	0.20	5	5	4
User Interface Consistency	0.15	4	4	5
Ease of Integration	0.20	4	3	5
Weighted Score	-----	4.45	3.55	4.70

Decision: Flutter + Mapbox/OSM Integration

The Flutter + Mapbox/OSM solution achieved the highest overall score (4.70). It offers robust cross-platform compatibility, reliable offline functionality, and straightforward integration with CRANC's backend. While Mapbox's native SDKs deliver slightly higher navigation precision, their mobile-only scope limits expansion to desktop or web environments. Google Maps excels in accuracy but lacks offline and open-source flexibility. Consequently, Flutter + Mapbox/OSM provides the most balanced and sustainable architecture for CRANC's navigation ecosystem.

Proving Feasibility

To validate our decision, the team tested the Flutter + Mapbox/OSM framework across Android, iOS, and web environments. The prototype demonstrated reliable real-time route rendering, GPS tracking, and consistent interface behavior across all platforms. Offline testing confirmed that cached map tiles allowed navigation to continue in low-connectivity areas. Moving forward, additional tests will focus on improving GPS precision, optimizing performance, and integrating routing updates directly from the CRANC backend. These steps will be showcased during the upcoming Technology Demonstration phase to confirm end-to-end navigation feasibility.

3.5 Scalability and Performance Optimization

Issue Introduction

As the cyclist routing system scales to support higher volumes of real-time spatial data and simultaneous user requests, maintaining system responsiveness becomes a primary engineering concern. The system must efficiently compute optimal routes, process large OpenStreetMap datasets, and return results with minimal delay, even as demand and data complexity grow. Without robust scalability and optimization strategies, performance degradation could result in delayed route generation, excessive memory use, or communication bottlenecks across distributed modules.

Desired Characteristics

To ensure sustainable performance, the ideal solution should provide the following characteristics:

1. **Low Latency:** Route computations and updates should complete within acceptable response times (e.g., under 500 ms per query).
2. **High Throughput:** The system should support multiple concurrent route requests without significant degradation in performance.
3. **Efficient Resource Utilization:** CPU, memory, and network bandwidth should remain within thresholds suitable for commodity hardware.
4. **Horizontal Scalability:** The architecture must support adding additional nodes or servers to handle increased demand.
5. **Fault Tolerance:** The system should continue performing reliably even if one or more nodes fail.
6. **Maintainability:** Optimization techniques must not compromise readability, modularity, or future extensibility of the system.

Alternatives:

Alternative 1: Monolithic Architecture with Code-Level Optimization focuses on improving efficiency within a single-instance architecture by implementing optimized algorithms, caching, and pruning unnecessary computations. It delivers strong short-term performance improvements but is inherently limited in scalability since all processes depend on a single server.

Alternative 2: Distributed Architecture with Load Balancing distributes user requests across multiple computation nodes through a load balancer. It supports horizontal scaling and improves reliability by eliminating single points of failure. However, it introduces moderate network overhead and requires orchestration management to ensure node synchronization.

Alternative 3: Microservices with Parallel Processing routing, data storage, and analytics are separated into distinct services that scale independently. This provides maximum flexibility and resilience but also increases deployment complexity and inter-service communication overhead.

Alternative 4: Edge Caching and Preprocessing caches frequently accessed routes on edge servers or client devices, reducing central server load and delivering very fast response times for repeated queries. However, it is less effective for new or infrequent route requests and adds caching consistency challenges.

Analysis

The monolithic optimization approach offered the fastest short-term performance but failed to scale effectively. The distributed load-balancing model provided strong scalability and fault tolerance but required moderate orchestration effort. Microservices achieved high scalability and fault tolerance but added significant management complexity. Edge caching performed well for repeat queries but was limited for dynamic routing scenarios.

Based on these comparisons and the weighted scoring table below, the Distributed Load Balancing with Code-Level Optimization approach achieved the highest overall performance balance. It combines the scalability benefits of distribution with the efficiency of targeted algorithmic improvements, providing a sustainable and maintainable architecture for CRANC's long-term needs.

Chosen Approach:

Table Explanation:

Each characteristic in the table below was assigned a weight based on its importance to CRANC's performance objectives, with the sum of all weights equaling 1.0. Alternatives were rated on a scale from **1 (poor)** to **5 (excellent)** for each characteristic, based on their documented performance, architecture design, and implementation of feasibility.

Characteristic	**Weight**	**Monolithic Optimization**	**Distributed Load Balancing**	**Microservices Architecture**	**Edge Caching**
Low Latency	0.25	3	4	4	5
High Throughput	0.20	2	5	4	5
Resource Efficiency	0.15	4	3	3	4
Horizontal Scalability	0.20	2	5	5	3
Fault Tolerance	0.10	2	4	5	4
Maintainability	0.10	4	3	3	3
Weighted Score	-------------	2.90	4.4	4.1	4.2

Decision: Distributed Load Balancing with Code-Level Optimization

The distributed load balancing with code-level optimization approach achieved the highest overall score (4.40). It offers a balanced combination of scalability, performance, and maintainability. Distributed architecture ensures the system can handle increasing user demand through horizontal scaling, while targeted optimizations reduce latency and computation costs within each node. Compared to microservices, this hybrid approach provides comparable scalability with less operational complexity and greater maintainability.

Proving Feasibility
To validate this solution, the team conducted multiple performance tests:

- **Load Testing:** Simulated concurrent route requests using Apache JMeter to measure throughput and latency under realistic conditions.
- **Stress Testing:** Evaluated system stability at maximum CPU and memory loads to identify bottlenecks and verify fault recovery.
- **Scalability Benchmarking:** Deployed the system across multiple Docker containers to test scaling efficiency as nodes were added.
- **Technology Demonstration:** Conducted a live test where real-time routing requests were distributed across several nodes. Performance metrics—including response time, CPU usage, and network throughput—confirmed that the system met target performance goals and scaled effectively.

4. Technology Integration

This section presents our envisioned system design and demonstrates how the selected technologies work together to deliver our complete solution.

System Architecture

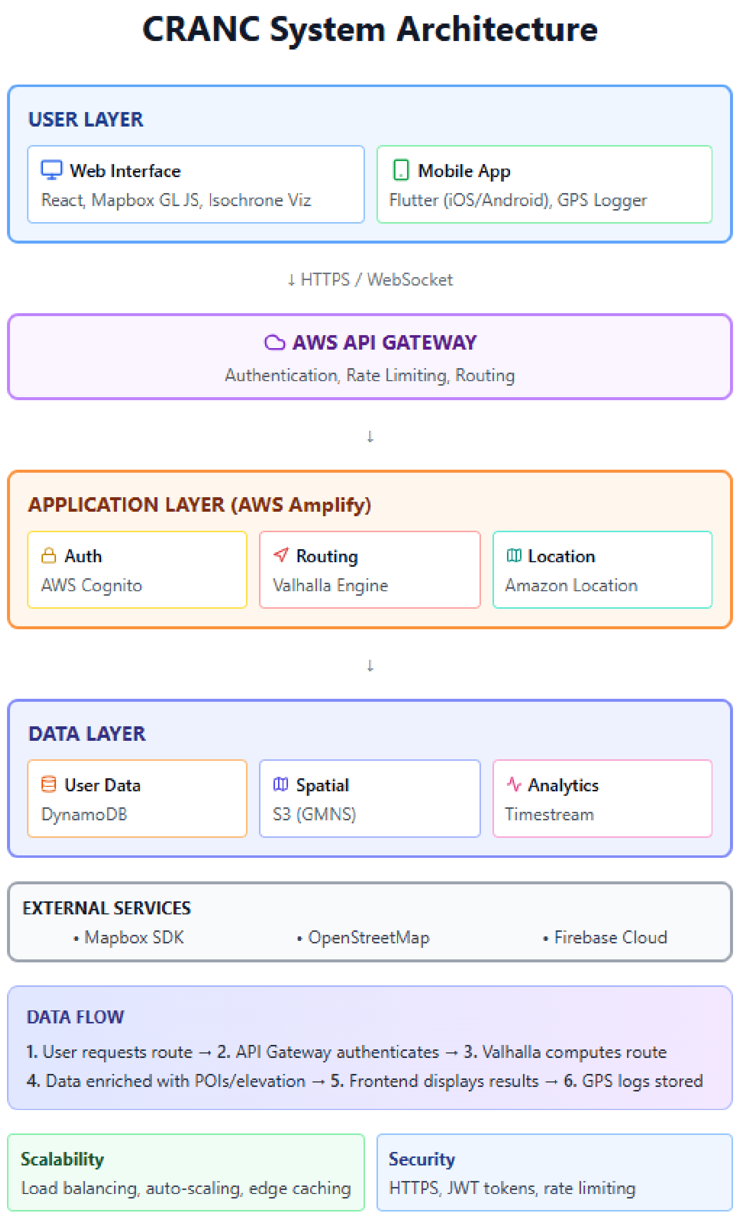
CRANC System Architecture
USER LAYER
Web Interface
React, Mapbox GL JS, Isochrone Viz
Mobile App
Flutter (iOS/Android), GPS Logger
↓ HTTPS / WebSocket
AWS API GATEWAY
Authentication, Rate Limiting, Routing
APPLICATION LAYER (AWS Amplify)
Auth
AWS Cognito
Routing
Valhalla Engine
Location
Amazon Location
DATA LAYER
User Data
DynamoDB
Spatial
S3 (GMNS)
Analytics
Timestream
EXTERNAL SERVICES
• Mapbox SDK
• OpenStreetMap
• Firebase Cloud
DATA FLOW
1. User requests route → 2. API Gateway authenticates → 3. Valhalla computes route
4. Data enriched with POIs/elevation → 5. Frontend displays results → 6. GPS logs stored
Scalability
Load balancing, auto-scaling, edge caching
Security
HTTPS, JWT tokens, rate limiting

Figure 1: System Architecture Overview

Architecture Walkthrough

Frontend Layer

The frontend enables users to access CRANC through both a web browser and a mobile app, ensuring a unified experience across all devices

- Web Interface: Built using React with Mapbox GL JS, it allows users to view maps, compute routes, and visualize isochrones dynamically. Accessibility and responsive design ensure functionality for planners and casual users alike.
- Mobile App: Developed in Flutter, supporting both iOS and Android, it provides turn by turn navigation, GPS logging, and route recording. Offline mode and lightweight caching allow continued operation even without network connectivity.

Both interfaces communicate securely with backend through RESTful APIs and WebSocket Channels managed by AWS Amplify

Application Layer

The application layer manages backend logic, coordinating frontend requests, data storage, and business rules to deliver real-time routing and analytics.

- Routing Engine: Computes personalized routes using GMNS network data and Valhalla algorithms.
- API Layer: Exposes RESTful and WebSocket endpoints for route requests, visualizations, and user profiles.
- Business Logic: Orchestrates route computation, map rendering, POI enrichment, and asynchronous GPS data processing.
- Analytics Module: Summarizes spatiotemporal data into safety metrics and planning insights for agencies.

This layer acts as the backbone of CRANC, connecting frontend interfaces with data storage and external services to provide a unified, responsive system.

Data Layer

The data layer ensures efficient storage, access, and synchronization of user and spatial data.

- User Profiles & GPS Data: Stored in AWS DynamoDB, with access managed by Cognito authentication tokens.
- Spatial and Network Data: Stored in AWS S3 in General Modeling Network Specification (GMNS) format for consistency across study areas.
- Historical and Analytical Data: Logged in AWS Timestream for performance analysis and to support future machine learning enhancements to routing recommendations.

This tiered data design enables high read and writes throughput and horizontal scalability while maintaining strong data integrity.

Integration Points

All Communication between components is secured via HTTPS and managed by AWS API Gateway. Data flows follow this general pattern:

1. User Interaction: Users request a route or visualization through the web or mobile client.
2. Routing Request: The front-end calls the backend API, which queries Valhalla and GMNS data.
3. Data Processing: The backend processes route computation and returns results with metadata.
4. Visualization: The front end displays enhanced isochrones, POIs, and navigation guidance.
5. Data Logging: The mobile app logs user routes and GPS data, which are asynchronously uploaded to DynamoDB for analysis.

External Services

The Cross-Platform Navigation System integrates several third-party services to enable reliable map rendering, route generation, and backend communication. The primary external services include Mapbox, OpenStreetMap (OSM), and Firebase Cloud Messaging.

- Mapbox provides a navigation engine, routing APIs, and map visualization layers used within the Flutter framework. It handles real-time route generation, rerouting logic, and offline map tile management.
- OpenStreetMap (OSM) serves as the underlying geospatial data source, ensuring open-access map coverage and allowing the system to function without dependency on proprietary datasets.
- Firebase Cloud Messaging is integrated for push notifications, enabling route status updates and synchronization alerts between users and the CRANC backend.

This architecture satisfies our core requirements by:

- Ensuring cross-platform functionality through Flutter's single codebase, enabling simultaneous support for Android, iOS, and web builds.
- Providing offline navigation via Mapbox's local tile caching and OSM's open map datasets, which maintain usability in low-connectivity environments.
- Delivering accurate real-time guidance using Mapbox's routing algorithms and telemetry data for consistent turn-by-turn navigation.
- Maintaining seamless integration with the CRANC backend through RESTful APIs, allowing synchronized route data, GPS logs, and performance feedback.

5. Conclusion

The CRANC project aims to provide safe, efficient, and accessible navigation for cyclists, addressing the challenges of real-time route computation, accurate mapping, cross-platform usability, and data-driven personalization. As urban cycling continues to grow, cyclists require tools that account for bike-friendly infrastructure, dynamic obstacles, and individual preferences—challenges that existing navigation solutions often fail to fully meet. Ensuring that these technological requirements are addressed is critical to delivering a system that is both reliable and practical for everyday use.

Key Challenges and Solutions:

- **Real-Time Routing Computation:** We chose the Open-Source Routing Machine (OSRM) for its zero cost, strong open-source community support, and low latency applications, making it a perfect fit for bicycle navigation.
- **Isochrone Visualization Upgrade:** We chose an Enhanced Isochrone Visualization approach that shows reachable areas and supports filterable points of interest, greatly improving usability and accessibility.
- **Mobile GPS Logging & User Profiles:** We chose AWS Amplify with Amazon Location Service to ensure secure authentication, accurate GPS tracking, and seamless backend integration, enabling scalable user data collection and analysis.
- **Cross-Platform Navigation System:** We adopted a Flutter based solution with Mapbox and OSM plugins, allowing consistent, responsive, and offline capable user experience across Android, iOS, and web platforms.
- **Scalability and Performance Optimization:** We selected a distributed load balancing approach with targeted code-level optimizations. This hybrid solution provides a balanced combination of scalability, performance, and maintainability, ensuring low-latency routing and reliable operation as user demand grows.

Through structured analysis of alternatives against project-specific criteria, we have demonstrated a clear rationale for each technological decision. Our comparison matrices and preliminary testing provide evidence that these choices are well-founded rather than arbitrary. The integrated system architecture presented in Section 4 confirms that our selected technologies form a coherent, implementable solution.

Key Findings:

- All major technological challenges have viable solutions within our constraints
- Selected technologies are compatible and integrate effectively
- Preliminary testing confirms feasibility of critical functionality
- Cost projections remain within budget parameters
- Timeline for implementation aligns with project schedule

Next Steps:

With technological feasibility established, our team will proceed to:

1. Develop comprehensive demonstrations for our Technology Demo deliverable
2. Begin detailed system design and architecture refinement
3. Establish development environment and toolchain
4. Create sprint plans for the initial implementation phase.