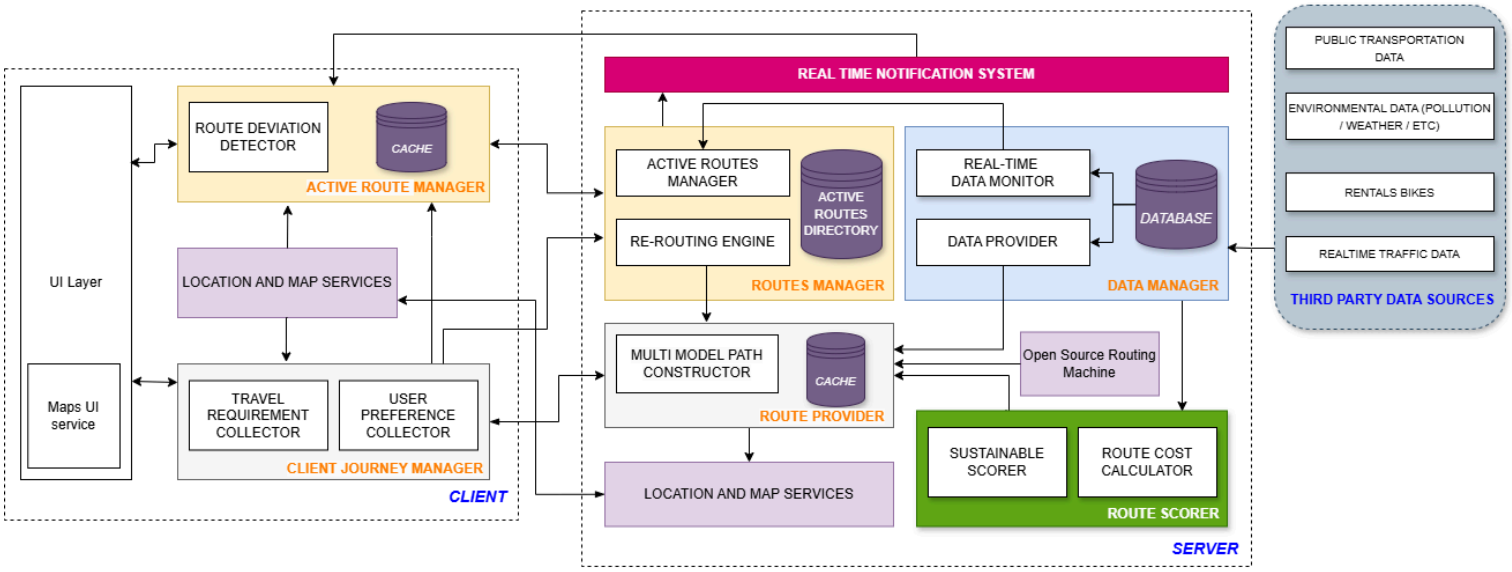
Contents

1. Functional Architecture

1.1 Functional Diagram



DYNAMIC SUSTAINABLE WAYFINDING FUNCTIONAL DIAGRAM

Fig 1. Functional Architecture of Dynamic Sustainable Wayfinding system

1.2 Component Differences

Open Source Routing Machine (OSRM) Integration

The Open Source Routing Machine (OSRM) was integrated as a new module within the Route Provider section, replacing earlier API-based routing methods. By allowing the loading of complete map datasets locally, OSRM enables more precise and customizable route calculations. It supports both walking and driving routes using OpenStreetMap data, giving the team greater control over routing logic and improving response times by reducing dependence on external services. This integration provides access to accurate route geometries and detailed navigation information, supporting the delivery of high-quality routing capabilities while also reducing latency and external API dependency.

Focused Bike Rental Data

To streamline data handling and reduce integration complexity, the real-time data ingestion was narrowed to focus specifically on bike rental services. Car rental and ride-sharing integrations—such as Uber, Lyft, and local taxi services—were deprioritized due to their inherently complex, multi-vendor API ecosystems. Each platform involves distinct authentication mechanisms, differing data schemas, and maintenance overhead. Integrating these services would have significantly increased development scope and complexity. In alignment

with Agile principles, the decision was made to start with a minimal, high-value feature set centered around bike rentals. This allowed the core routing and navigation system to be validated more quickly, with the possibility of extending support to additional transport options in future sprints based on user needs and feedback.

Removal of Parking Data

The parking data component was deliberately excluded from the initial product due to the high complexity and low value it added at this stage. Parking data is often hyper-local, subject to frequent changes, and typically requires continuous updates across multiple city jurisdictions. Maintaining such a dataset would introduce substantial overhead, especially in ensuring real-time accuracy. Additionally, acquiring reliable parking data often comes with high subscription costs. Guided by the Agile principle of "build the simplest thing that could possibly work," the team chose to focus on the core navigation and sustainability features. This decision helped avoid early feature bloat and allowed development efforts to be concentrated on high-impact areas. It also reduced technical debt and system complexity while keeping the door open for future parking data integration if user demand justifies it.

P.T.O

2. Technical Architecture

2.1 Technical Diagram

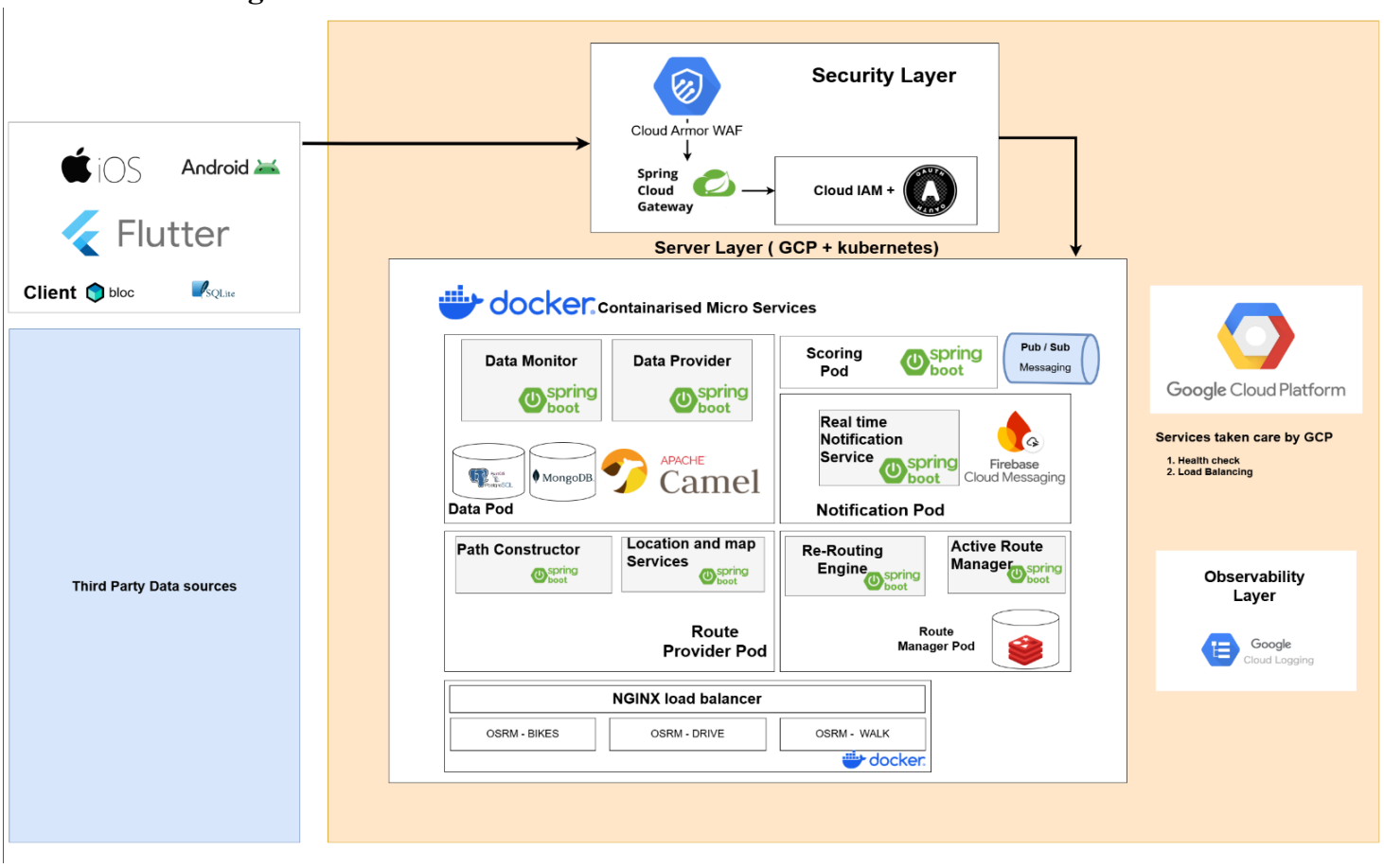


Fig 2. Technical Architecture of the Dynamic Sustainable Wayfinder

2.2 Component Differences

Kubernetes Removal

Initially, our architecture included Kubernetes to orchestrate our microservices across multiple servers and regions. However, in line with Agile principles—particularly the idea of "simplicity: the art of maximizing the amount of work not done"—we decided to scale back to a single-server deployment. This shift allowed us to focus on delivering core functionality rather than investing time and resources into managing complex infrastructure. The change significantly reduced the team's operational overhead and the learning curve associated with Kubernetes, enabling us to accelerate development cycles by streamlining the deployment process. While Kubernetes is powerful and well-suited for large-scale orchestration, it introduces unnecessary complexity at our current scale. The single-server setup provides enough capacity for our needs and allows us to prioritize value delivery over infrastructure management.

Nginx Implementation

As we simplified our infrastructure, we also introduced Nginx as a load balancer between microservices that now include real-time data from OpenStreetMap Routing Machine (OSRM). This change reflects our intent to

maintain a robust and scalable architecture within a more manageable system. By implementing Nginx, we achieved more efficient traffic distribution, as it effectively routes incoming requests across microservice instances to prevent any one instance from becoming overwhelmed. It also enhances reliability by redirecting traffic away from unhealthy services, improving overall fault tolerance. Additionally, Nginx simplifies the process of scaling; we can add new service instances without changing how client applications interact with the system. This maintains the flexibility of a microservices architecture while avoiding the complexity introduced by our original Kubernetes-based design.

Streamlined Observability

One of the most impactful architectural changes we made involved simplifying our observability stack. Our initial implementation used Prometheus and Grafana for metrics collection and visualization. While these are powerful tools and implemented for thin slice implementation, we encountered several limitations. Prometheus's memory usage posed concerns, particularly as the volume of collected metrics increased. Combined with our single-server setup, this raised potential scalability issues. Additionally, managing Prometheus and Grafana introduced maintenance overhead and required specific expertise. As a result, we transitioned to Google Cloud Metric Explorer, which offered a lightweight and effective solution. Google Metrics provided server-level performance insights with minimal configuration and supported client-side error logging, making it a practical fit for our environment. Although extending metrics visibility to users could add value, we determined that doing so at this stage would complicate the system unnecessarily. As we noted in internal discussions: “While extending the functionality of metrics to users is good, it made things too complex.” Google Metrics strikes a balance by offering essential observability without the burden of a full monitoring stack.