## 1. Vision & Problem Definition

1.1 Problem Statement

Auckland’s transport network is complex and often unreliable, leading to wasted time, commuter frustration, and increased carbon emissions. Public transport users face unpredictable travel times due to traffic congestion, delays, and incomplete information. City and transport planners lack timely, actionable data to proactively address congestion and improve urban mobility. Existing solutions, such as AT Mobile, do not provide predictive insights, eco-friendly route recommendations, or targeted analytics for planners, resulting in missed opportunities for smarter, greener transport.

1.2 GreenRoute AKL Vision

GreenRoute AKL aims to become Auckland’s intelligent transport platform—blending real-time and historic mobility data, AI-powered predictions, and carbon insights to help users travel smarter and city planners build a more sustainable city. The vision is to empower Aucklanders with faster, more reliable, and environmentally conscious transport choices, while equipping planners with actionable insights to reduce congestion and emissions. This project is designed to showcase the real-world potential of data-driven mobility and sustainability for New Zealand’s largest city.

1.3 Unique Value Proposition

* AI-powered travel delay predictions—not just static timetables or traffic info.
* Carbon impact calculator—shows the real emissions cost of each trip and nudges eco-friendly choices.
* Planner dashboard—provides powerful, visual analytics and real-time pain-point detection for transport authorities.
* Open, extensible, and privacy-focused platform—designed for easy growth and maximum user trust.

## 2. Core Features & MVP Scope

**2.1 Core Features**

GreenRoute AKL focuses on the following core features, designed to deliver maximum value to both public users and city/transport planners:

**a. Smart Route Planner (Public Users)**

* **Description:** An intelligent route planner that combines live and historic transport data to suggest the fastest, most reliable, and most sustainable travel options across Auckland.
* **Key Functionality:** Multi-modal journey planning (bus, train, ferry, walking), real-time updates, suggested routes optimized for time and sustainability.

**b. Travel Time & Delay Prediction (AI-powered)**

* **Description:** Uses machine learning to predict likely delays and provide accurate ETA estimates.
* **Key Functionality:** Predicts disruptions using historical and real-time data, builds commuter confidence, and differentiates GreenRoute AKL from standard planning apps.

**c. Carbon Footprint Calculator**

* **Description:** Calculates the carbon impact of each journey, giving users eco-friendly insights for every trip.
* **Key Functionality:** Shows carbon cost for route options, encourages greener travel choices, supports Auckland’s climate goals.

**d. Planner Dashboard (City/Transport Planners)**

* **Description:** A secure dashboard with advanced analytics, heatmaps, and data visualizations.
* **Key Functionality:** Identifies congestion hotspots, analyzes trends, helps planners make evidence-based improvements to the city’s transport network.

**2.2 MVP Scope: What’s In, What’s Out**

**Included in MVP (Must-Have)**

* Smart Route Planner (web, public)
* Real-time and predictive travel time/delay information
* Basic carbon calculator for trip results
* Planner dashboard with simple data visualizations (e.g., congestion map)
* User authentication (basic)

**Not Included in MVP (Nice-to-Have/Future)**

* Mobile app (PWA or native)
* Personalized commute tips & notifications
* User-generated reports (crowdsourced feedback)
* Gamified eco challenges
* Integration with bike/scooter/ride-share services

**2.3 Why This MVP?**

* **Addresses Auckland’s biggest pain points:** Delays, unreliability, and lack of eco-friendly travel options.
* **Proves technical skill:** Uses live data, ML prediction, and modern web tech—impressive for employers and future collaborators.
* **Delivers value fast:** Public users get actionable journey planning; planners get new data-driven insights to inform decisions.
* **Leaves room for growth:** Roadmap features are clear, but MVP remains focused and achievable.

## 3. Architecture & Tech Stack

**3.1 Project Structure**

GreenRoute AKL will use a **modular, monorepo** approach, with clear separation between backend, frontend, and documentation. This improves code organization, team collaboration, and deployment flexibility.

greenroute-akl/

│

├── backend/ # FastAPI app: APIs, ML, DB logic

│

├── frontend/ # React app: web UI for commuters & planners

│

├── docs/ # Architecture docs, wireframes, specs

│

└── database/ # (Optional) SQL scripts, migrations

**3.2 System Architecture Overview**

**Components and Data Flow:**

* **Frontend (React)**
  + Public web app for commuters (trip planning, carbon insights)
  + Planner dashboard (analytics, admin functions)
  + Communicates with backend via REST API
* **Backend (FastAPI)**
  + Aggregates live and historic transport data from APIs
  + Runs ML models (delay prediction, carbon calculator)
  + Handles business logic and exposes secure REST endpoints
* **Database (Supabase/PostgreSQL)**
  + Stores users, trips, predictions, analytics
  + Managed by Supabase for easy integration and scalability
* **External Data Sources**
  + Auckland Transport, Waka Kotahi NZTA, Auckland Council Open Data
* **DevOps/Hosting**
  + Frontend: Vercel or Netlify
  + Backend: Railway.app or Render
  + Database: Supabase cloud

**3.3 Chosen Tech Stack**

| **Layer** | **Technology** | **Rationale** |
| --- | --- | --- |
| **Backend** | Python (FastAPI) | Modern, fast, type-safe, async-ready |
|  | scikit-learn, pandas, SQLAlchemy | Data science, ML, ORM |
| **Frontend** | React (JS/TS), Recharts, Leaflet | Fast UI, rich data viz, mobile-friendly |
| **Database** | Supabase (PostgreSQL managed) | Easy setup, free tier, built-in Auth |
| **APIs** | AT GTFS/RT, NZTA, Council Open Data | Rich public datasets |
| **DevOps** | Vercel/Netlify, Railway/Render | Free, fast deploys for MVP |
| **Auth** | Supabase Auth or JWT | Secure, simple for MVP |

**3.4 Why This Architecture?**

* **Separation of concerns** for easier scaling and collaboration
* **Cloud-native and modern stack** for speed and cost-efficiency
* **Extensible and open** for future growth (mobile, new data sources, more features)
* **Strong NZ ecosystem fit** (uses local public data and hosting that works well in NZ context)

## 4. Data Strategy

**4.1 External Data Sources**

GreenRoute AKL will leverage trusted, real-time, and historic datasets to drive both public and planner features. Key data sources include:

* **Auckland Transport Open Data Portal**  
  <https://dev-portal.at.govt.nz/>  
  GTFS, real-time vehicle locations, service alerts, stop info.
* **Waka Kotahi NZTA APIs**  
  <https://opendata-nzta.opendata.arcgis.com/>  
  Traffic incidents, roadworks, travel time data.
* **Auckland Council Open Data**  
  <https://data.aucklandcouncil.govt.nz/>  
  Demographics, environmental data, active transport infrastructure.

**4.2 Database Design**

**Entities:**

* Users (public & planner roles)
* Trips (planned journeys, selected modes)
* Route options & history
* Delay predictions (historic & real-time)
* Carbon footprint records
* Analytics for dashboard (congestion, hotspot metrics)

**Schema principles:**

* Use normalized, relational tables (PostgreSQL)
* Leverage Supabase features for auth, easy querying, and analytics

**4.3 Data Ingestion & ETL Pipeline**

* **Frequency:**
  + Real-time data polled every few minutes for trip planning
  + Nightly/weekly pulls for historic/training data
* **Processing:**
  + Clean and validate incoming data (remove duplicates, standardize formats)
  + Store raw feeds for auditing, processed for app use
  + Aggregate data for analytics and ML (delay prediction, carbon calc)
* **Tools:**
  + Python scripts (scheduled jobs), pandas for ETL
  + FastAPI endpoints for triggered updates

**4.4 ML Pipeline for Prediction**

* **Input:**  
  Historic travel times, weather, events, live vehicle locations
* **Modeling:**
  + scikit-learn for initial delay prediction model
  + Simple regression/classification for MVP
  + Carbon estimates based on route, vehicle type, and standard factors
* **Serving:**
  + ML model predictions run server-side (API)
  + Results cached for fast frontend display

**4.5 Data Privacy & Security**

* Personal data (location, travel history) is **never shared** without user consent
* Data is anonymized for analytics and reporting
* Follows New Zealand privacy best practices (minimum required data, user control)

## 5. Backend Development

**5.1 API Design**

* **RESTful API** using FastAPI
* **Versioned endpoints** (e.g., /api/v1/)
* **OpenAPI/Swagger docs** auto-generated for easy integration

**Core endpoints:**

* /plan-route — Returns optimal routes, real-time updates, and ETA
* /predict-delay — Returns delay predictions for a given route/trip
* /carbon-calc — Returns carbon emission estimates for journeys
* /dashboard/analytics — Provides analytics data for planner dashboard
* /auth — User registration/login (JWT or Supabase Auth)

**5.2 Data Models**

* **Pydantic models** for input/output validation
* **SQLAlchemy ORM models** for PostgreSQL database tables

**Key entities:**

* User
* Trip
* RouteOption
* DelayPrediction
* CarbonCalculation
* Feedback (future)

**5.3 ML Integration**

* **Travel delay prediction**:
  + Pre-trained scikit-learn model loaded in backend
  + Model receives live/historic data as input via API call
* **Carbon calculator**:
  + Calculation logic (per route/mode) embedded as a Python service or callable function

**5.4 Authentication & Authorization**

* **Supabase Auth or JWT** for user registration and login
* **Role-based access control**:
  + Public users: journey planning endpoints
  + Planners: dashboard and analytics endpoints

**5.5 Error Handling & Validation**

* Use Pydantic for all input validation
* Sanitize and format error messages for users (never expose stack traces)
* Custom exceptions for common issues (e.g., InvalidRoute, UserNotFound)
* Centralized error logging for diagnostics

**5.6 Documentation & Developer Experience**

* Built-in FastAPI docs at /docs
* Additional API usage guides in the /docs folder
* Example requests/responses for each endpoint

## 6. Frontend Development

**6.1 App Structure**

* **Monorepo:** All frontend code is contained within the frontend/ folder.
* **Single-Page Application (SPA):** Built using React for fast, interactive user experience.
* **Separation of Concerns:**
  + **Public user interface:** Trip planning, real-time updates, and carbon feedback.
  + **Planner dashboard:** Analytics, heatmaps, congestion insights (role-based access).

**6.2 UI/UX Design**

* **Responsive design:** Works seamlessly on desktop and mobile browsers.
* **User Flows:**
  + **Public Users:** Home → Plan Journey → View ETA & Carbon → Save/Share Trip
  + **Planners:** Login → Dashboard → View Analytics/Hotspots → Download Reports
* **Wireframes:** Initial wireframes and mockups will be created for main screens using tools like Figma or Excalidraw (to be stored in /docs).

**6.3 State Management & Data Fetching**

* **State Management:**
  + Lightweight approach (React Context API or Redux Toolkit, if needed).
  + Separate global state for user/auth and journey planning.
* **Data Fetching:**
  + API calls to backend using Axios or Fetch.
  + Use React Query (TanStack Query) for caching, async state, and smoother UX.
  + Graceful loading and error states for all data requests.

**6.4 Data Visualization Components**

* **Maps:**
  + Use React-Leaflet for route maps, congestion overlays, and planner insights.
* **Charts & Analytics:**
  + Use Recharts for delay trends, modal share, carbon footprint, etc.
* **Dashboard:**
  + Card and grid layouts for quick insights, configurable by planners.

**6.5 Authentication & Routing**

* **Authentication:**
  + Integration with Supabase Auth or backend JWT flow.
  + Role-based rendering (users see trip planner; planners see dashboard).
* **Routing:**
  + Use React Router for SPA navigation.
  + Protected routes for planner dashboard.

**6.6 Accessibility & Quality**

* **WCAG-compliant:** Color contrast, keyboard navigation, screen-reader support.
* **Testing:**
  + Use Jest and React Testing Library for unit/component tests.
  + Manual QA for all main flows.

**6.7 Documentation**

* **Developer Onboarding:**
  + README in /frontend with setup, run, and build instructions.
* **Component Library:**
  + Document common UI components for easy reuse.

## 7. Deployment & DevOps

**7.1 Local Development Workflow**

* **Monorepo Setup:**
  + Run backend (backend/) and frontend (frontend/) locally in parallel.
  + Use .env files to manage environment variables and API keys securely.
* **Developer Tools:**
  + VS Code recommended for unified workflow.
  + Pre-configured scripts (npm run dev, uvicorn app.main:app, etc.).

**7.2 Hosting & Deployment Plan**

* **Frontend:**
  + Deploy to **Vercel** or **Netlify** for fast, free static hosting of React app.
* **Backend API:**
  + Deploy FastAPI app to **Railway.app** or **Render** (free/low-cost, supports Python and environment variables).
* **Database:**
  + **Supabase** cloud (managed PostgreSQL, built-in Auth).
  + Regular automatic backups enabled.

**7.3 Continuous Integration & Delivery (CI/CD)**

* **Automated Deploys:**
  + GitHub Actions or built-in CI from Vercel/Netlify for frontend.
  + Automatic deploy on main/production branch push.
* **Testing:**
  + Run backend and frontend test suites on each PR/commit.
  + Linting checks for code quality (e.g., Black for Python, ESLint/Prettier for React).
* **Manual QA:**
  + Checklist for post-deploy sanity checks (key user journeys, planner dashboard).

**7.4 Environment Management**

* **Secrets & Configs:**
  + Never commit secrets to code.
  + Use .env files for local, environment variables in cloud dashboards for production.
* **Config Versioning:**
  + Sample .env.example files included for onboarding new developers.

**7.5 Monitoring & Logging**

* **Basic Logging:**
  + Backend logs for API errors and ML predictions (use standard Python logging).
* **Error Reporting:**
  + Frontend error boundaries and browser logging.
* **(Future) Monitoring:**
  + Optional: Integrate with Sentry or similar for error tracking at scale.

**7.6 Documentation**

* **Deployment Guides:**
  + Step-by-step README sections for deploying backend, frontend, and database.
  + Troubleshooting common deployment issues.

## 8. Security & Privacy

**8.1 Authentication & Authorization**

* **User Authentication:**
  + Use Supabase Auth (recommended) or JWT-based system for secure login.
  + Strong password requirements for planners and admin accounts.
* **Role-Based Access Control:**
  + Public users access journey planning only.
  + Planner accounts access analytics and dashboard endpoints.
* **Session Management:**
  + Short-lived tokens; refresh logic for secure, persistent sessions.

**8.2 Data Protection**

* **Secure Storage:**
  + Sensitive data (user credentials, personal details) stored securely in Supabase/PostgreSQL with encryption at rest.
* **Transport Security:**
  + Enforce HTTPS for all API and frontend traffic.
* **Input Validation:**
  + All user and API input validated with Pydantic (backend) and client-side checks (frontend) to prevent injection attacks.

**8.3 Privacy Best Practices**

* **Minimum Data Principle:**
  + Collect only what is essential for service delivery (e.g., anonymize location/journey data wherever possible).
* **User Consent:**
  + Explicit opt-in for storing personal or journey history data.
  + Transparent privacy policy shared with all users.
* **Data Anonymization:**
  + Use de-identified data for analytics, reporting, and dashboard visualizations.
* **Right to Erasure:**
  + Users can request deletion of their data at any time.

**8.4 Compliance**

* **NZ Privacy Act 2020:**
  + Follows New Zealand’s legal requirements for personal data collection, use, and storage.
* **Third-Party Data:**
  + Respect data licenses and terms of Auckland Transport, NZTA, and Council datasets.

**8.5 Threat Mitigation**

* **Rate Limiting:**
  + Prevent API abuse with sensible rate limits per user/IP.
* **Brute Force Protection:**
  + Lock accounts after repeated failed login attempts.
* **Audit Logging:**
  + Log access to sensitive endpoints (future: review audit logs for suspicious activity).
* **Regular Security Updates:**
  + Keep all dependencies updated and monitor for vulnerabilities.

## 9. Testing & Quality

**9.1 Backend Testing**

* **Unit Tests:**
  + Write unit tests for core business logic, API endpoints, and ML prediction functions using pytest.
  + Mock external APIs and database connections to isolate backend logic.
* **Integration Tests:**
  + End-to-end tests for API routes (using httpx or similar) to verify data flow from frontend through backend to the database.
  + Validate that responses are accurate, properly formatted, and error-handling works as expected.
* **Test Coverage:**
  + Aim for 80%+ coverage on key backend modules (especially those used in production).
  + Include a coverage report in CI/CD pipeline for every push.

**9.2 Frontend Testing**

* **Component Testing:**
  + Use Jest and React Testing Library for UI components.
  + Test major user flows: journey planning, viewing delays, dashboard analytics.
* **Integration & E2E Testing:**
  + Use Cypress or Playwright for simulating real user interactions across the app.
  + Automate key flows like login, plan a route, view dashboard.
* **Accessibility Checks:**
  + Run accessibility audits (e.g., axe, Lighthouse) to ensure WCAG compliance.

**9.3 Manual QA Checklist**

* Test all main features: route planning, delay prediction, carbon calculator, dashboard.
* Verify responsive design on multiple devices and browsers.
* Simulate error cases (invalid input, API failure, offline mode).
* Ensure proper error messages and recovery options are shown to users.

**9.4 Code Quality**

* **Linting:**
  + Use Black for Python, ESLint & Prettier for React to enforce code style and prevent common bugs.
* **Code Reviews:**
  + Use GitHub Pull Requests for all new features/fixes, with peer or self-review before merging.
* **Refactoring:**
  + Regularly improve code structure and remove technical debt as features evolve.

**9.5 Continuous Improvement**

* **Bug Tracking:**
  + Use GitHub Issues or similar for tracking bugs and test failures.
* **Feedback Loops:**
  + Encourage testers and users to report issues and suggest improvements.
* **Regression Testing:**
  + Re-run all automated and manual tests before each new release.

## 10. Documentation

**10.1 Project Overview & Onboarding**

* **Root README:**
  + Clear project summary, features, and goals.
  + High-level folder structure and navigation guide.
  + Badges for build status, coverage, etc. (if using CI).
* **Quick Start Guide:**
  + Step-by-step instructions to run backend, frontend, and database locally.
  + Prerequisites (Python, Node, etc.), environment setup, and basic troubleshooting.
* **Contribution Guide:**
  + How to clone, branch, commit, and make pull requests.
  + Coding conventions for Python and React (see your GreenRoute AKL Code Guidelines).
  + Issue template and code review checklist.

**10.2 API Documentation**

* **Auto-generated Docs:**
  + FastAPI’s Swagger/OpenAPI available at /docs for all API endpoints.
* **Endpoint Reference:**
  + Inputs, outputs, authentication requirements, and example requests for each endpoint.
  + Error code meanings and sample error payloads.

**10.3 Frontend/User Documentation**

* **User Manual:**
  + How to plan a journey, interpret carbon results, and use the planner dashboard.
  + Screenshots or short GIFs to illustrate user flows.
* **FAQs & Troubleshooting:**
  + Common problems (e.g., login issues, data not loading) and solutions.
  + Contact/support channel for feedback.

**10.4 System Architecture & Design Docs**

* **Diagrams:**
  + System/component diagrams, data flow charts, and database schema.
  + Wireframes or UI mockups for main screens.
* **Data Dictionary:**
  + Definitions for core data fields/entities (users, trips, analytics, etc.).
* **ML Model Details:**
  + Outline of training data, modeling process, and limitations for predictive features.

**10.5 Deployment & Operations**

* **Deployment Guide:**
  + Step-by-step for deploying backend, frontend, and database to production.
  + Environment variable and secrets management.
* **Update Process:**
  + How to roll out new features or bugfixes, including post-deploy QA checklist.

**10.6 Maintenance & Roadmap**

* **Known Issues:**
  + Document any open technical debt or feature gaps.
* **Changelog:**
  + Track major feature additions, bugfixes, and changes between releases.
* **Future Roadmap:**
  + List planned features or ideas for next phases (mobile app, gamification, etc.).

## 11. Metrics & Feedback

**11.1 Success Metrics**

To measure the effectiveness and growth of GreenRoute AKL, the following key performance indicators (KPIs) will be tracked:

* **User Adoption & Engagement**
  + Number of unique users planning trips weekly/monthly
  + Average sessions per user
  + User retention rate (how many return users)
* **Feature Usage**
  + Percentage of trips using the carbon calculator
  + Usage rates of delay prediction and real-time updates
  + Dashboard logins/active planner users
* **Prediction Accuracy**
  + ML model accuracy for delay predictions (e.g., % of ETAs within target window)
  + Reduction in average reported commute time for users
* **Impact Metrics**
  + Estimated reduction in total carbon emissions from route choices
  + Number of actionable insights delivered to planners (e.g., identified congestion hotspots)
  + Feedback from city planners on dashboard utility

**11.2 Feedback Collection**

* **In-app Feedback Tools**
  + Short surveys or thumbs up/down on route results and dashboard insights
  + Optional comment box for suggestions or bug reports
* **User Interviews & Surveys**
  + Periodic user and planner interviews (for deep qualitative insights)
  + Follow-up surveys after key features are released
* **Bug & Feature Requests**
  + Simple in-app form or GitHub Issues for users to report problems or request new features
* **Analytics Dashboard**
  + Internal dashboard to visualize KPIs and monitor trends over time

**11.3 Continuous Improvement**

* **Iterative Releases**
  + Use collected metrics and feedback to guide the MVP and future feature roadmap
  + Rapidly address pain points, usability issues, and user suggestions
* **Stakeholder Reporting**
  + Share regular updates (e.g., monthly) with city planners, mentors, or collaborators
  + Highlight wins (e.g., reduction in delay, improved user satisfaction) with data

## 12. Future Roadmap & Scalability

**12.1 Post-MVP Feature Roadmap**

* **User-Generated Reports**
  + Allow commuters to report delays, incidents, or inaccurate predictions—enriching the dataset for all users.
* **Personalized Commute Tips**
  + Suggest travel times, modes, or routes based on user preferences and travel history.
* **Gamified Eco Challenges**
  + Reward users for low-carbon travel habits (e.g., “Green Week” challenges, leaderboards, digital badges).
* **Integration with Bike/Scooter/Ride-Share**
  + Incorporate real-time availability and routing for micro-mobility and on-demand transport.
* **Mobile App (PWA or Native)**
  + Extend the reach of GreenRoute AKL with a fully-featured mobile app for offline access and push notifications.
* **Push Notifications & Real-Time Alerts**
  + Inform users of major disruptions, alternative routes, or eco opportunities.

**12.2 Scaling Plan**

* **Data & Infrastructure**
  + Design APIs and database to handle growth in data volume (e.g., city-wide adoption, additional transport modes).
  + Modular architecture allows for easy addition of new data sources, ML models, or external services.
* **User Base**
  + Start with targeted rollouts (e.g., student and commuter groups) and expand as features stabilize.
  + Prepare for city-wide deployment if adoption metrics are strong.
* **Team & Collaboration**
  + Documentation and clear codebase structure allow new developers, collaborators, or open-source contributors to onboard quickly.

**12.3 Vision for Long-Term Impact**

* **Smart City Integration**
  + Collaborate with Auckland Council, AT, or other agencies to embed GreenRoute AKL insights in urban planning.
* **Template for Other Cities**
  + Modular, open-source platform design enables adaptation for Wellington, Christchurch, or other global cities with similar challenges.
* **Sustainability Leadership**
  + Set an example for sustainable urban mobility—using data, AI, and community feedback for a greener future.

## 13. Risks & Mitigation

**13.1 Real-Time Data Reliability**

**Risk:**  
Public APIs and transport feeds may experience downtime, delays, or inconsistent formatting, impacting route accuracy and user trust.

**Mitigation:**

* Implement fallback logic (e.g., cache last known good data).
* Monitor data sources and alert if feeds are stale.
* Document API limits and reliability for users/planners.

**13.2 Machine Learning & Prediction Challenges**

**Risk:**  
ML models may struggle with accuracy due to limited or biased data, affecting user confidence in delay predictions.

**Mitigation:**

* Start with transparent, simple models (explainable predictions).
* Continuously retrain with new data and user feedback.
* Display confidence intervals or warnings when predictions are uncertain.

**13.3 Privacy & Security Concerns**

**Risk:**  
Handling location and journey data may raise user privacy concerns or create compliance risks under NZ law.

**Mitigation:**

* Anonymize or aggregate sensitive data before analysis/reporting.
* Obtain explicit user consent for any personal data storage.
* Regular security reviews and privacy policy updates.

**13.4 Feature Creep & MVP Focus**

**Risk:**  
Expanding the scope beyond MVP could delay launch or lead to unfinished features.

**Mitigation:**

* Maintain a clear, documented MVP boundary.
* Use a prioritized backlog and review regularly.
* Gather user feedback early to validate must-have features.

**13.5 Scaling & Performance**

**Risk:**  
Growing user numbers or data volumes could strain APIs, ML models, or hosting resources.

**Mitigation:**

* Use scalable cloud services (Supabase, Vercel, Render).
* Profile and optimize performance for key endpoints.
* Plan for staged rollouts and stress testing before wide release.

**13.6 Stakeholder Engagement**

**Risk:**  
Limited feedback or engagement from public users or city planners could stall development or reduce impact.

**Mitigation:**

* Regularly seek feedback via in-app tools, interviews, and surveys.
* Engage early with stakeholders (Auckland Transport, city planners).
* Share progress updates and highlight project impact to maintain interest.

**13.7 Third-Party Dependency Risks**

**Risk:**  
Changes in third-party APIs, pricing, or terms of service could disrupt the app.

**Mitigation:**

* Monitor for API/service changes proactively.
* Maintain abstraction layers in code for easy swapping of providers.
* Document alternatives for key external dependencies.