Lancaster Link
System Design Document

1. Brief Outline of the Project

1.1 Purpose of this document

This document outlines the detailed design of LancasterLink. It translates the high-level objectives and features outlined in the project bid into solid architectural decisions, data flows, component responsibilities, and implementation plans.
It serves as a shared reference for the development team, ensuring that the system's behaviour, assumptions, and limitations are clearly defined before implementation.

1.2 Scope and assumptions

LancasterLink is designed to operate within a defined geographical area. The system supports journey planning and live transport visualisation across Lancaster, Preston, Blackpool, Fylde, and Wyre. It will support bus, rail, and tram operators. The system will provide route planning, disruption awareness, and live network visibility on a map. However, it will not support ticket purchasing, payments, refunds, or fare management.
The design assumes the availability of static timetabled information and live operational feeds from third-party APIs. If live data is unavailable or delayed, the system will fall back to the static timetable.
Explicit exclusions include:

- Ticketing and payment services
- National journey planning
- 100% guaranteed real-time accuracy.

2. Software Requirements

2.1 Functional requirements

ID	Requirement	Priority	Category
FR-JP-01	**Multi-Modal Routing:** Calculate routes involving Bus, Rail, and Tram between any two points in the target region (Lancaster, Preston, Blackpool, Fylde, Wyre)	MUST	**Journey Planner**
FR-JP-02	**Fallback Routing:** System must default to scheduled timetable data if live data streams are unavailable	MUST	
FR-JP-03	**Disruption Awareness:** Route calculations must penalise or avoid services flagged as "Cancelled" or "Significantly Delayed" in live feeds.	MUST	
FR-JP-04	**Future Planning:** Allow users to plan journeys up to 3 months in advance using static timetables	SHOULD	
FR-JP-05	**Interchange Buffering:** Prioritise routes with "well-served transport hubs" and avoid fragile connections (e.g., <5 min transfer at a small stop).	SHOULD	
FR-JP-06	**Recent Journeys:** Persist recent origins/destinations locally (e.g., via Local Storage/Cookies) for quick selection.	SHOULD	

FR-JP-06	**Multi-Platform Recent Journeys:** Persist recent origins/destinations for a user over multiple platforms by introducing user accounts	COULD	
FR-LM-01	**Network Visualisation:** Display static routes for Bus, Rail, and Tram overlaid on a geographical map	MUST	**Live Map**
FR-LM-02	**Live Vehicle Tracking:** Render current vehicle positions (interpolated or GPS) for supported operators (Stagecoach, Northern, etc.).	MUST	
FR-LM-03	**Stop Inspection:** Clicking a stop/station must display upcoming departures (scheduled vs. expected).	MUST	
FR-LM-04	**Filtering:** Allow users to toggle visibility of specific transport modes (e.g., hide buses, show only trains).	SHOULD	

2.2 Non-functional requirements

ID	Requirement	Priority	Category
NFR-PL-01	**Response Time**: Route calculation for a typical journey (e.g., Lancaster -> Blackpool) should complete within **< 5 seconds**	MUST	Performance and Latency
NFR-PL-02	**Map Refresh Rate:** Live vehicle positions on the map must update at least every **30 seconds** (subject to data availability).	SHOULD	
NFR-PL-03	**Load Handling:** The system must support concurrent use by the evaluation team (approx. 5-10 users) without degradation.	MUST	
NFR-RA-01	**Graceful API Failure:** If the live data feed API fails, the system **MUST** automatically revert to displaying scheduled timetable data without crashing.	MUST	Reliability and Availability
NFR-RA-02	**Stale Data Handling:** Vehicle data older than 5 minutes must be discarded or visually flagged as "stale" to prevent misleading users.	MUST	
NFR-SDP-01	**No PII Storage:** The application **MUST NOT** store any Personally Identifiable Information (names, emails) or track user movements.	MUST	Security and Data Privacy
NFR-SDP-02	**No Payment Processing:** The application **MUST NOT** process or store any financial data.	MUST	
NFR-SDP-03	**Input Sanitisation:** All user inputs (origin/destination text) must be sanitised to prevent Injection attacks.	MUST	
NFR-SDP-04	**Dependency Safety:** All third-party libraries must be audited for known vulnerabilities.	MUST	
NFR-UA-01	**Mobile Compatibility:** The web interface must be responsive and functional on mobile viewports (min-width 320px).	MUST	Usability and Accessibility
NFR-UA-02	**Interface Simplicity:** The UI must be "simple" and "consistent," minimising the number of clicks required to plan a route.	MUST	
NFR-UA-03	**Visual Accessibility:** The application should use high-contrast colours (WCAG AA standard) to distinguish routes.	SHOULD	
NFR-UA-04	**Error Feedback:** The system must provide human-readable error messages (e.g., "No route found" vs "Error 500").	SHOULD	

2.3 Data Ingestion & Integration Requirements

ID	Requirement	Priority
DR-01	**Data Linking:** Programmatically link NaPTAN stop points with NPTG localities to group stops into logical "towns" (e.g., grouping "Lancaster Rail Station" into "Lancaster").	MUST
DR-02	**Feed Normalisation:** Normalise diverse operator data (e.g., Northern vs. Stagecoach) into a single standard format.	MUST
DR-03	**Polling Frequency:** Poll live API endpoints at a rate sufficient for accuracy but within server rate limits.	MUST

2.4 Routing Logic & Reliability Heuristics

ID	Requirement	Priority
RL-01	**Fragile Connection Avoidance:** Do not provide routes with connection times less than a specific amount (e.g. <5 mins).	SHOULD
RL-02	**Hub Prioritisation:** Weight "well-served transport hubs" higher than remote stops.	SHOULD
RL-03	**Disruption Rerouting:** Automatically recalculate active journeys when a "Cancelled" or "Delayed" message is received.	MUST
RL-04	**Multi-Modal Stitching:** Connect bus legs to rail legs, accounting for walking time between stops and platforms.	MUST

2.5 Operational & Deployment Constraints

ID	Requirement	Priority
OD-01	**Containerisation:** All backend services must be deployable as Docker/Podman containers.	MUST
OD-02	**CI/CD Pipeline:** The repository must include a CI/CD pipeline for automated testing and building.	MUST
OD-03	**Resource Efficiency:** The system must run effectively within the resource limits of the provided lab VMs/Scafell server.	MUST

2.6 Accessibility & Interface Compliance

ID	Requirement	Priority
AIC-01	**Visual Clarity:** Use distinct visual styles (e.g., colour/dash) to differentiate "Live Data" from "Timetable Only" data.	MUST
AIC-02	**Contextual Geography:** Display key landmarks and town names on the map.	SHOULD
AIC-03	**Device Responsiveness:** The UI must be usable on mobile viewports	SHOULD
AIC-04	**WCAG Compliance:** Follow basic accessibility guidelines (contrast, sizing).	SHOULD

3. System Architecture

3.1 High-level architecture diagram

See appendix (Figure 6)

3.2 Component responsibilities

1. Data Sources

- Encapsulates where data will be sourced from i.e. BPLAN, OSM
- Allows live telemetry, timetables, and weather factors.

2. Management Layer

- 2a) Will gain information from data sources by API polling and utilising STOMP. (see Figure 13)
- 2b) Will format data into a sufficient structure for logic processing (i.e. mapping bus locations to their respective routes) and store in a database. (see Figure 14)
- 2c) Handles requests from logic processing to correctly search the database (see Figure 15)

3. Logic Processing

- 3a) Calculates the best route possible based on distinct characteristics specified (see Figure 11)
- 3b) Used to update the system when a disruption is detected (see Figure 12)
- 3c) Provides an interface for the application layer to access quality of life features.
- 3d) Provides an interface for application layer to access locational travel data.

4. User Application

See Figure 10.

- 4a) Allows the user to input desired characteristics about their journey.
- 4b) Visualises the map from OSM when prompted with calculated route nodes.
- 4c) Maps route finding nodes (see 3a) to points on the map, as well as disruption detection (see 3b) visualisation.
- 4d) Adds quality of life features like ‘recent journeys’, ‘nearby stops’, ‘accessibility access’, etc.
- 4e) Gives the user an interface with which to communicate with the system.

3.3 Design Principles That Influenced the Architecture of the Code

Separation of Concerns

The system is divided into four clearly defined layers (Section 3.2):

- **Data Sources** – External APIs and static datasets
- **Management Layer** – Data ingestion, normalisation, and storage
- **Logic Processing** – Routing, disruption handling, and heuristic evaluation
- **User Application** – Presentation and user interaction

Each layer has a single, well-defined responsibility. This ensures that changes to external data feeds (e.g., operator format changes) do not directly affect routing logic or the frontend, supporting DR-02 (Feed Normalisation) and NFR-RA-01 (Graceful API Failure).

SOLID PRINCIPLES

The design follows core SOLID principles to maintain extensibility and testability:

- **Single Responsibility Principle (SRP)**
  Components such as route calculation, disruption detection, and live data ingestion are implemented separately. For example, routing cost computation (Section 5.2) is isolated from data retrieval logic.

- **Open/Closed Principle (OCP)**
  The routing system is designed so new heuristics (e.g., additional reliability metrics) can be added without modifying the core graph structure.
- **Dependency Inversion Principle (DIP)**
  Logic Processing depends on abstracted data interfaces rather than concrete API implementations. This allows live feeds to be mocked during testing and replaced if operators change.

Loose Coupling and High Cohesion

Inter-component communication occurs via clearly defined interfaces (see Section 6). The frontend interacts only with the API layer and does not access the database or routing engine directly.

This reduces coupling between the presentation layer and business logic, improving:

- Testability (Section 8 – Testing)
- Deployment flexibility (OD-01 Containerisation)
- Fault isolation in case of API instability

Fault Tolerance and Graceful Degradation

Given the reliance on third-party APIs, resilience was a key architectural driver.

- Live data is treated as an enhancement layer over static timetables.
- Failures in live feeds do not prevent routing, satisfying FR-JP-02 and NFR-RA-01.
- Stale data handling prevents misleading outputs (NFR-RA-02).

This ensures that the system continues to operate even under partial data failure.

Scalability Within Project Constraints

The architecture supports modest concurrent usage (5–10 users) without overengineering.

- Stateless API interactions simplify scaling.
- Containerised backend services (OD-01) allow deployment flexibility.
- Data normalisation in the Management Layer prevents routing complexity from growing with additional operators.

4. Data Design

4.1 Structure of Data

LancasterLink will gather and process data from multiple external sources, such as static timetable data and live operational data. All data from external sources will be normalised into a consistent structure before being allowed to be used by components of the system, such as the routing logic or information displayed on the frontend.

The system will group and structure data into the following:

- User
- Saved Journeys
- Stops
- Routes
- Timetable
- Live vehicles
- Tracked vehicles.

User account data will be stored in a secure database; it will also be hashed. Each of the users recent /saved journeys will be linked via their email address. Additionally, operational data and user account data will be separated within the database to maintain clear and secure boundaries between system and user data.

4.2 Ownership of Data

Data ownership can be considered in terms of both information ownership and internal system components ownership of data.

Static timetable and live operational data used by LancasterLink will be obtained from third party APIs. The system will act solely as a consumer of this data within the defined scope of the project. User account data will be provided by users; this means that users own their respective data. LancasterLink will be responsible for securely storing this data in accordance with ethical and legal guidelines.

From an architectural point of view, responsibility for managing data is assigned and managed by specific components within the system, which are responsible for retrieving, updating, and maintaining data. Other components can interact with components that own this data when data access or modification is need, rather than directly doing this themselves, which can cause errors and mismatched data throughout the system. This model ensures that responsibility for stored data remains with the appropriate system component.

5. Journey Planning and Live Map Design

5.1 Routing approach

The Graph-Based Model

- The transport network will be treated as a **Time-Dependent Directed Graph**
    - **Nodes:** Represent NaPTAN stop-points (Bus stops, Tram Stops, Train Stations)
    - **Edges:** Represent travel ‘legs’ that are derived from static timetables (bus routes, tram lines, rail lines). This will also extend to walking legs between bus/tram stops and/or train stations.
    - **Weights:** Represent travel time

Time Dependency

- Unlike a standard road map, the graph representing the transport network will differ depending on the time and date requested.
    - For future journeys, the graph is constructed solely from static timetable data.
    - For "leave now" journeys, edge weights and availability are updated using live service data where available.

5.2 Reliability heuristics

- The Routing Algorithm will not just look for the fastest time, instead it will look for the **lowest cost.**
    - Cost will be defined using travel time, transfer time, and the reliability heuristic.
    - $Cost = Travel\,Time + Transfer\,Time + Reliabilty\,Heuristic$

How will the Reliability Heuristic be derived?

- There will be two variables that contribute to the reliability heuristic:
    - Hub Prioritisation: Stations and stops that facilitate more routes/services over a given period will have a smaller heuristic.

- **Delay Anticipation**: Stations and stops that are more prone to delays, in that they facilitate **routes/services that are delayed more frequently** than average will have a greater heuristic. (Not to be confused with edge weights being updated in real time from live data)

5.3 Bus and Tram Live Data Integration

- Bus and tram services typically provide GPS-based live vehicle locations. These live positions are integrated into the same graph model used for routing. Each update includes:
    - Vehicle ID, Current latitude and longitude, Route, and service number
- Live positions are matched to their scheduled routes, allowing the system to infer vehicle progress and expected arrival times at downstream stops.

Interaction with Routing and Live Map

- Live delay or cancellation information directly influences routing by penalising or removing affected edges, as described in Sections 6.1 and 6.2.
- The live map visualises current vehicle positions and service status, providing users with contextual awareness without requiring them to interpret raw data.

Fallback Behaviour

- As live data quality varies by operator:
    - The system retains the last known valid position if updates are temporarily unavailable.
    - If live data becomes stale or unavailable, timetable-based predictions are used instead.

5.4 Rail Live Data Integration

- Rail services primarily provide **event-based** updates rather than continuous GPS tracking. These events include arrivals, departures, and cancellations at known locations.

Position Inference

- Between reported events, train position is inferred using:
    - The last confirmed station or signalling point.
    - Scheduled running times between locations
- This inferred position is used for **live map visualisation only**, while routing decisions rely on confirmed timetable and disruption data to avoid overconfidence in estimates.

Disruption Handling

- **Delays** update expected arrival and departure times and increase routing penalties for affected services.
- **Cancellations** remove affected services from the routing graph, triggering alternative route suggestions where available.

6. API Design

6.1 Style of API

- LancasterLink uses a **REST-style API** with stateless request–response interactions.
    - **GET** requests are used to retrieve journey results, live vehicle positions, departures, and disruption information.
    - **POST** requests are used to submit journey planning queries.
    - **PUT** and **DELETE** operations are not required within the current project scope.
- The API design prioritises:
    - **Read-heavy usage**, reflecting the absence of user accounts and persistent user data.
    - **Graceful degradation**, falling back to timetable-based data when live feeds are unavailable (FR-JP-02, NFR-RA-01).
    - **Consistency**, with uniform response structures across transport modes and operators.

6.2 Endpoint Grouping

- Endpoints are organised by user-facing functionality rather than by transport operator:
  - **Journey Planning**: Exposes routing results produced by the graph-based model and reliability heuristics described in Section 6.
  - **Live Data**: Provides current vehicle positions, service status, and disruption alerts.
  - **Map State**: Returns the combined set of routes, vehicles, and alerts required to render the live map efficiently.
  - **Stops and Departures**: Supports stop-level inspection used by both the journey planner and live map.

6.3 Example Endpoints

Endpoint	Purpose
/api/journey	Returns journey planning results
/api/map/state	Returns routes, live vehicles, and disruptions for map rendering
/api/live/buses	Returns live bus and tram positions
/api/live/trains	Returns live rail service status
/api/departures/{locationId}	Returns upcoming departures for a stop or station

Figure 5 shows the API design.

7. User Interface Design

Figure 1 shows the landing page from which they can search for a new journey or select a recently searched journey or a saved journey.
Once searched for a journey, the user is displayed (*Figure 2*) with a list of available journeys that meet their search criteria. Once a specific journey is selected, they are taken to a live map view of that journey (*Figure 3*), and a detailed overview of the journey is shown.
Alternatively, they can switch to the Live map view (*Figure 4*), via the top navigation bar. The live map page will include an interactive map showing the live locations and data of included transportation. As well as a quick selection of nearby stops and favourited stops. Once a user selects a specific stop they will see the next departures for that stop.

8. Testing

Testing activities will be split into two categories, technical and acceptance testing. Technical testing will focus on system correctness, while acceptance testing will assess the suitability of the system against the project goals.

8.1 Technical Testing

8.1.1 Unit Testing

Unit testing will be used on individual system components to ensure that they behave as intended in isolation. Where applicable, unit testing will be written in python, with each major component having its own set of tests and python test file. Where needed, controlled inputs and predefined expected outputs will be used to validate functionality and edge cases.

8.1.2 Integration Testing

Integration testing will be used to verify that individual complete components work correctly when merged and that needed data is flowing correctly. This will include components like live data integration, routing logic, and the frontend live map interface.
Where needed, controlled API responses will be used to simulate realistic operating conditions that the program will experience. Integration tests will confirm that data can flow correctly between components and that system fallback mechanisms are triggered when errors occur.

8.1.3 Automated Testing

Automated testing will be implemented using a CI/CD pipeline provided by GitHub actions. The pipeline will automatically execute both unit and integration tests when a developer tries to commit code, this ensures system stability. A failed test will cause the pipeline to fail, preventing these changes from being merged into the main branch.

8.2 Acceptance Testing

8.2.1 Acceptance Test Objectives

Acceptance testing will verify that core user tasks can be completed successfully and that system behaviour aligns with the defined requirements and user expectations.

8.2.2 Acceptance Test Scenarios

Acceptance testing will be conducted using structured, task-based scenarios based on the project requirements. Each scenario will represent a realistic user task and will include the following:

- A clear starting goal
- A clear end goal
- The expected system behaviour
- A pass / fail criterion.

Scenario testing will cover normal tasks as well as edge and failure cases, ensuring the full scope of the system has been tested.

8.2.3 User-Based Testing

User-based testing will be conducted with a small group of participants who have not been involved in the system development process. Participants will be asked to complete the predefined acceptance test scenarios without guidance to assess the usability of the system. Prior to user testing, ethical approval will be obtained in accordance with guidelines.
During testing, observations will be recorded to collect quantitative data, including task completion, user errors, hesitation, and points of confusion throughout the task. All feedback will be anonymised and used solely for evaluation of the system.

8.2.4 Usability Metrics

The data collected during user-based testing will be analysed using predefined usability criteria. Metrics gathered from participants will be reviewed to identify common issues, inefficiencies, or usability barriers.
Results will be evaluated to determine whether core features of the system can be completed reliably and with minimal user frustration. Patterns of common mistakes or tasks that cannot be completed within a reasonable period will be used to identify areas where the system may require refinement.

8.2.5 Qualitative Feedback

In addition to the quantitative data collected, qualitative feedback will also be collected from participants following the completion of the test scenarios, this will be gathered in the form a short questionnaire. Participants will be asked to rank each task based on ease of use and overall satisfaction. They will also be given the opportunity to give written comments on each task, allowing them to justify the scores given to each task.

9. Risks and Mitigations

9.1 Risk Matrix

Likelihood / Consequence	Insignificant Risk is easily mitigated	Minor Delays or additional costs increased by 10%	Moderate Delays or additional costs increased by 30%.	Major Delays or additional costs increased by 50%	Catastrophic Project abandoned
Certain ≥ 90% chance	High	High	Extreme	Extreme	Extreme
Likely ≥ 50% - < 90% chance	Moderate	High	High	Extreme	Extreme
Moderate ≥ 10% - < 50% chance	Low	Moderate	High	Extreme	Extreme
Unlikely ≥ 3% - < 10% chance	Low	Low	Moderate	High	Extreme
Rare < 3% chance	Low	Low	Moderate	High	High

9.2 Risk Analysis and Mitigation Plan

Risk	Likelihood	Consequence	Risk Level	Strategy	Plan
Requirements misunderstood by team	Unlikely	Major	High	Minimisation	Clarify requirements early through weekly meetings and ensure a clearly defined MVP is agreed before development begins
Expansion of project requirements beyond the agreed scope	Moderate	Moderate	High	Minimisation	Enforce change control, ensuring any additional features are deferred to after the main project scope is completed
Data inconsistency between transport operators	Moderate	Moderate	High	Minimisation	Have a data normalisation layer to ensure consistent data representation
Dependency on third-party APIs	Moderate	Major	Extreme	Contingency	Design the system to degrade

					gracefully if third party APIs are not available
Live transport data feed instability	Moderate	Major	Extreme	Minimisation	Cache recent live data and fall back to predefined timetables when live feeds are unavailable
Database does not support required queries	Unlikely	Major	High	Minimisation	Design the database schema around known use cases
Integration issues between system components	Moderate	Major	Extreme	Minimisation	Integrate components incrementally and perform integration testing to identify issues early
Testing delayed	Moderate	Major	Extreme	Minimisation	Follow a structured testing plan including unit, integration, and manual testing throughout development
Key staff ill	Moderate	Moderate	High	Minimisation	More overlap of team so wider knowledge of critical components
Delay in delivery of critical features	Moderate	Major	Extreme	Minimisation	Prioritise critical features and include contingency time in the time schedule / Gantt chart
Delay in delivery of non-critical features	Moderate	Minor	Moderate	Minimisation	Defer non-critical features if critical features already delayed

10. Delivery Plan

10.1 Milestones/Responsibilities

Toby

- 2c) Data Requests
- 3d) App Layer API
- 3c) QoL Characteristics

Tom

- 2a) Data Sourcing
- 3a) Route Finding

Keegan

- 2b) Data Storing
- 3b) Disruption Detection

Marc & Jake

- 4a) Route Planner
- 4b) Map Visualisation
- 4c) Node Plotting
- 4d) QoL Features
- 4e) User Interface

10.2 Time Frames

- See ‘Critical Path Table’ appendix (Figure 7)
- See ‘Critical Path Diagram’ appendix (Figure 8)
- See ‘Gantt Chart’ appendix (Figure 9)

The estimated total time of the project is **55 days**.

11. Costings For the Project

Staff Costs

Role	Calculations	Cost (£)
Senior Dev / Project lead	55 days x £475	26, 125
Software Dev (x4)	55 days x £310 per day per dev	68, 200
Staff Subtotal		94, 325

Operational Overhead Costs (3 months)

Item	Cost (£)
Rent	2,100
Electricity & Heating	540
Water	150
Internet	100
Operation Subtotal	2,890

Infrastructure & Hosting (1 Year)

Item	Monthly Cost (£)	Annual Cost (£)
Production Server(s)	300	3,600
Database Hosting	80	960
Automated Backups	40	480
Infrastructure Subtotal		5,040

Description	Amount (£)
Subtotal (Before Contingency)	102, 255
Contingency (10%)	10, 226
Total Fixed Project Cost	112, 481

Commercial Profit (15%)	16,872
Total Estimated Cost	129, 353

12. Appendix

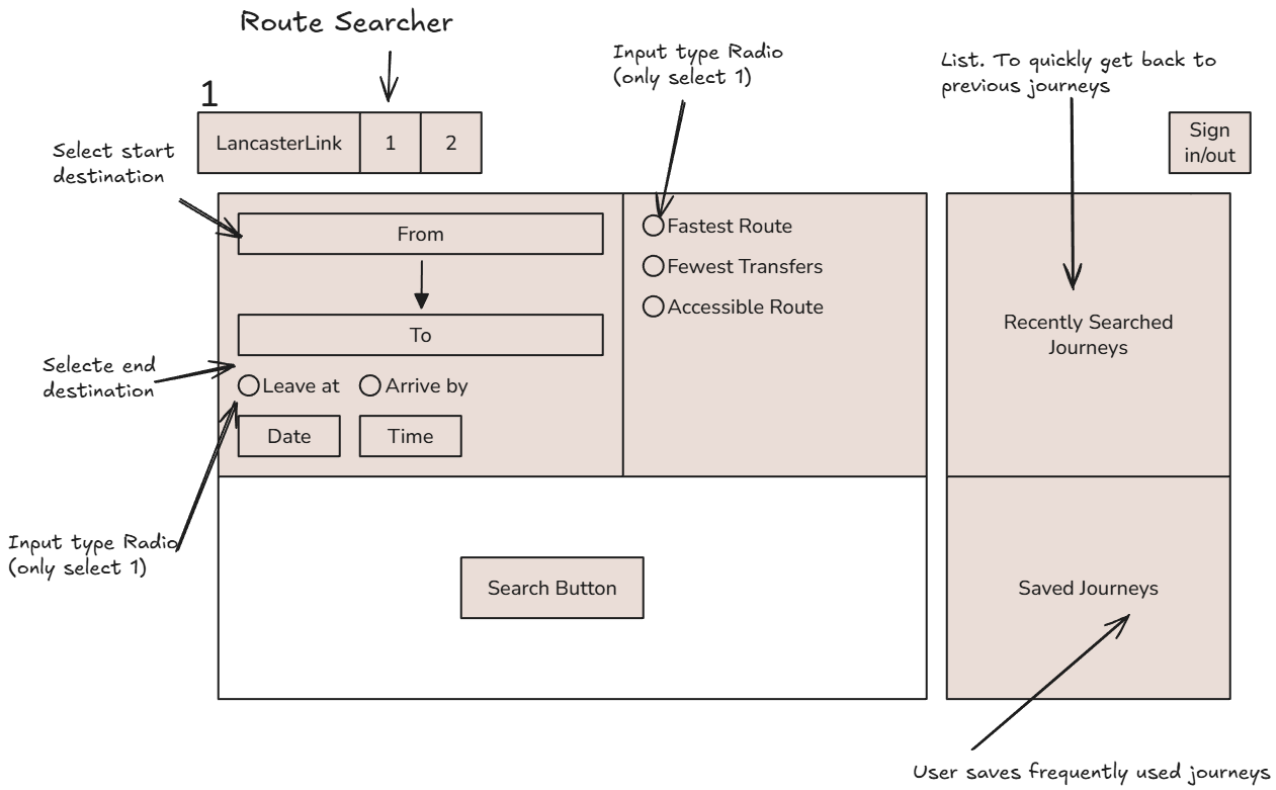

Figure 1

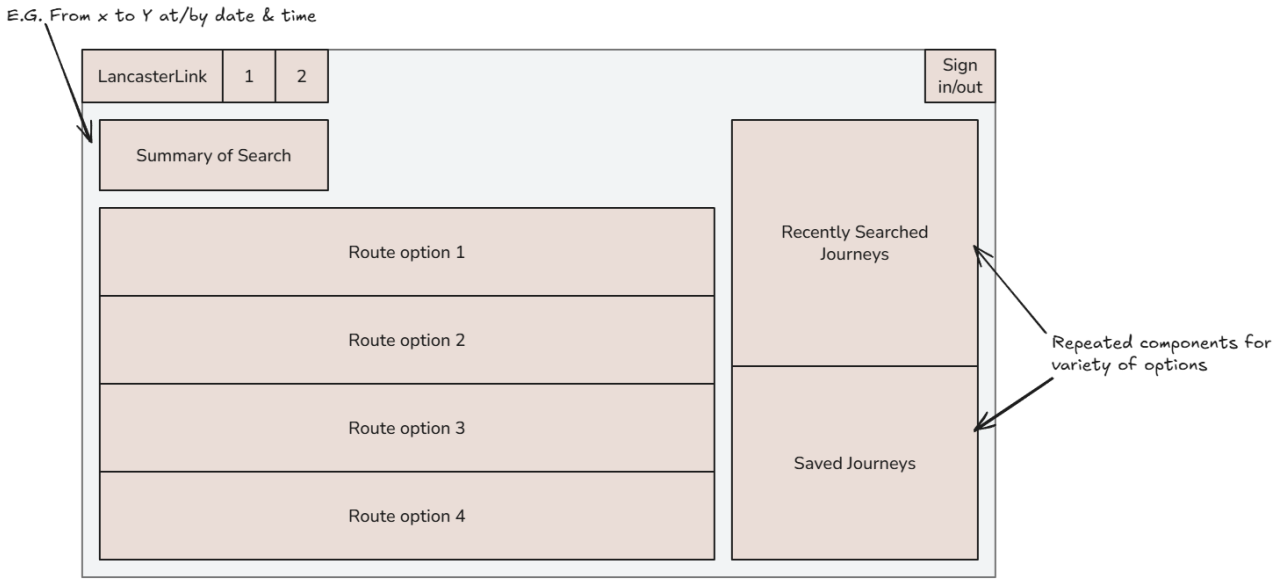

Figure 2

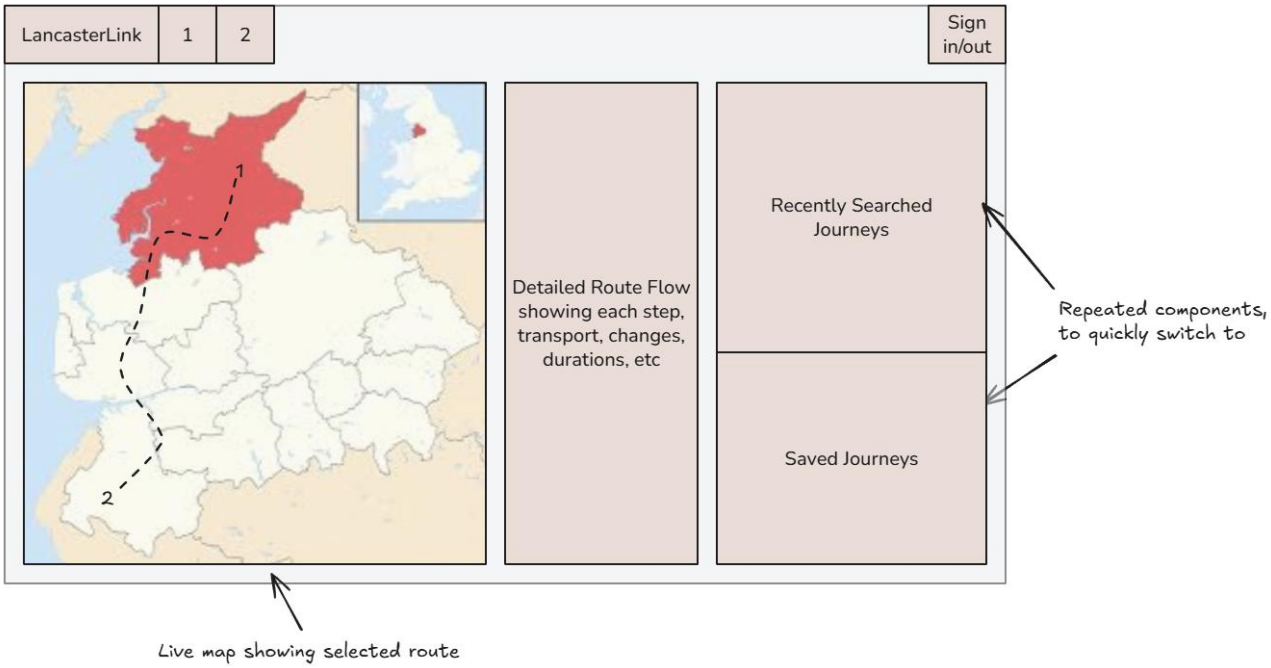

Figure 3

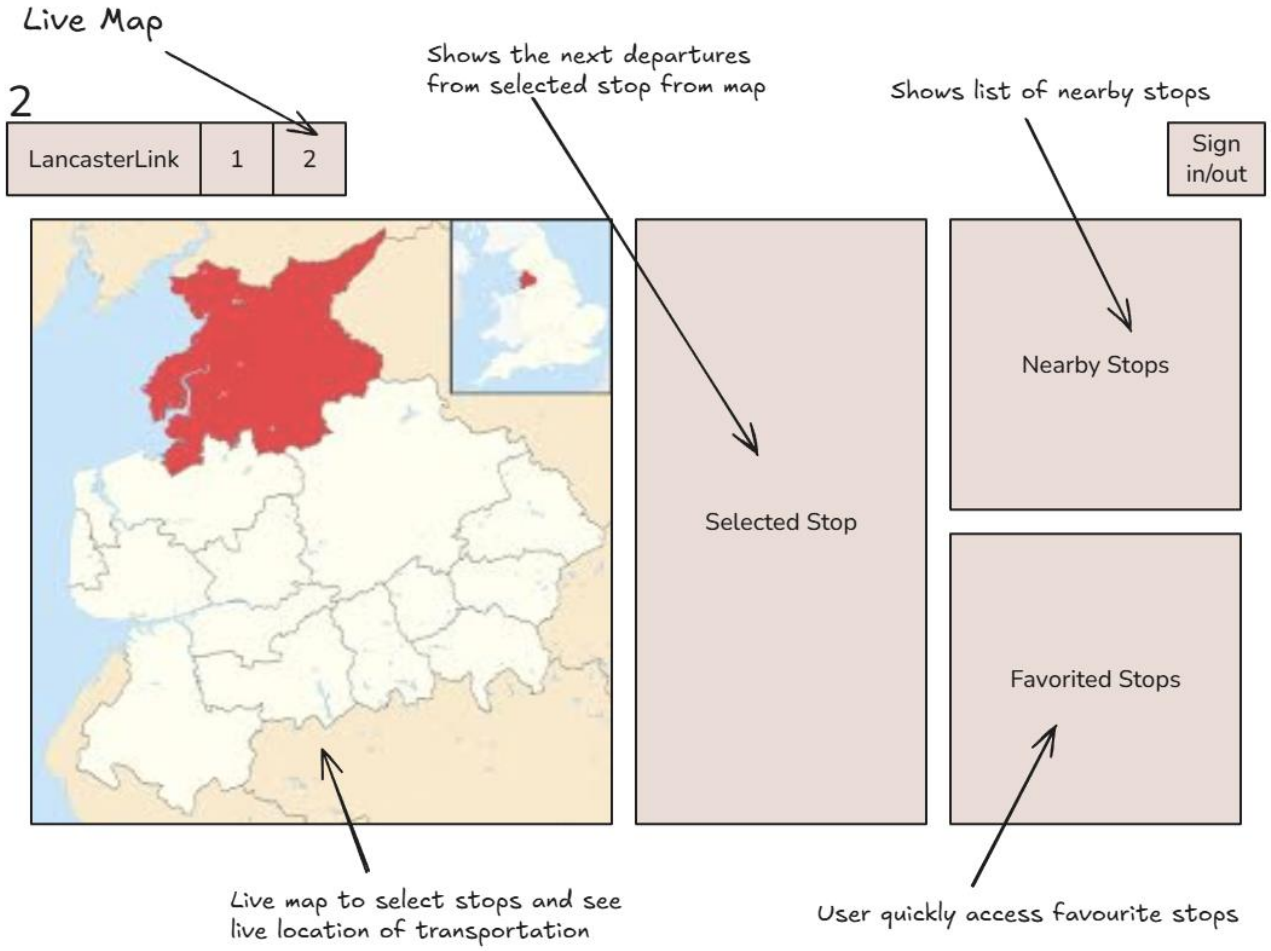

Figure 4

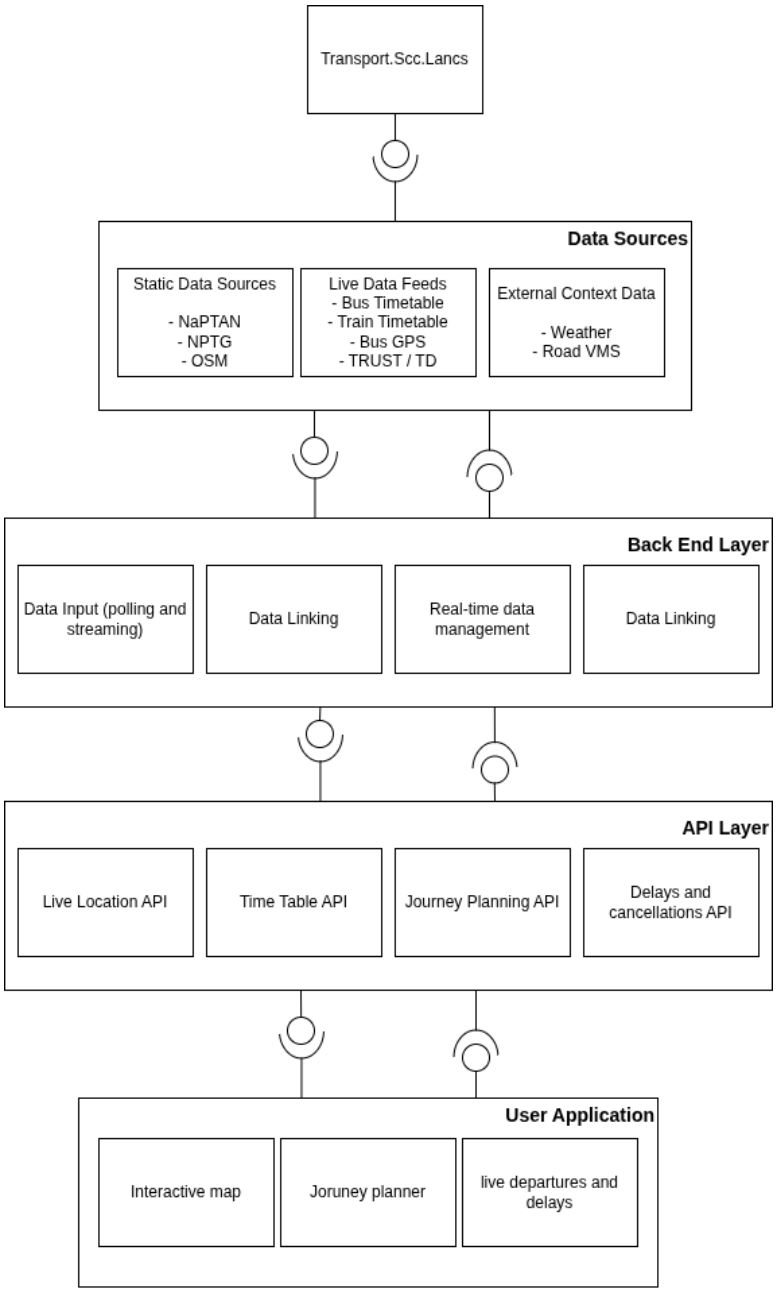
Transport.Scc.Lancs
Data Sources
Static Data Sources
- NaPTAN
- NPTG
- OSM
Live Data Feeds
- Bus Timetable
- Train Timetable
- Bus GPS
- TRUST / TD
External Context Data
- Weather
- Road VMS
Back End Layer
Data Input (polling and streaming)
Data Linking
Real-time data management
Data Linking
API Layer
Live Location API
Time Table API
Journey Planning API
Delays and cancellations API
User Application
Interactive map
Joruney planner
live departures and delays

Figure 5

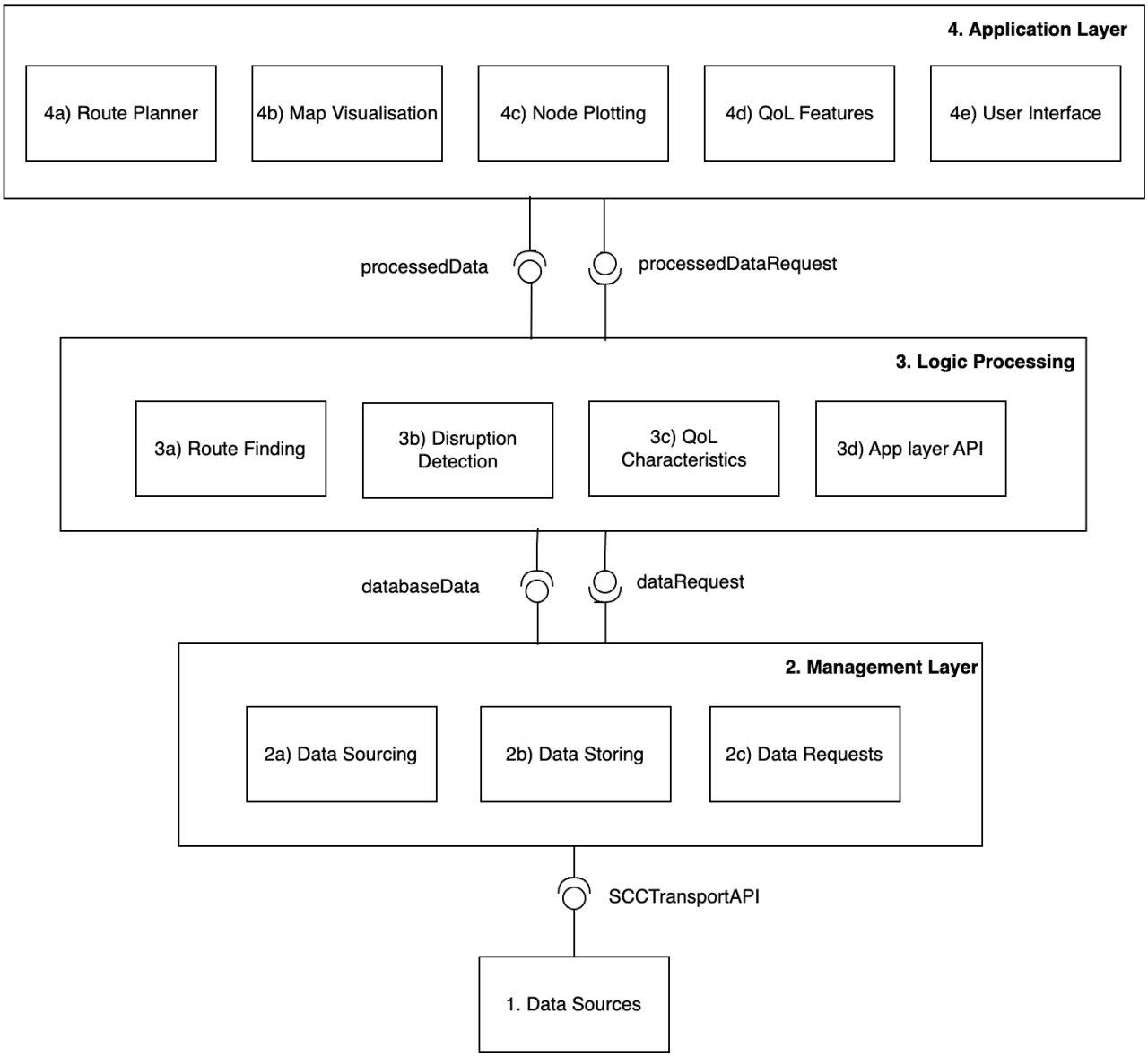

Figure 6

System Section	Activity	Predecessor	Duration (days)
1. Sourcing	1	-	1
2. Management	2a	1	7
	2b	2a	7
	2c	2b	7
3. Logic Processing	3a	2c	14
	3b	2c	7
	3c	2c	4
	3d	2c	4
4. Application Layer	4d	3c	4
	4e	-	21
	4c	3a,3b,3d	7
	4b	4c,4e	5
	4a	4b	7
		Critical Path:	55 Days

Figure 7

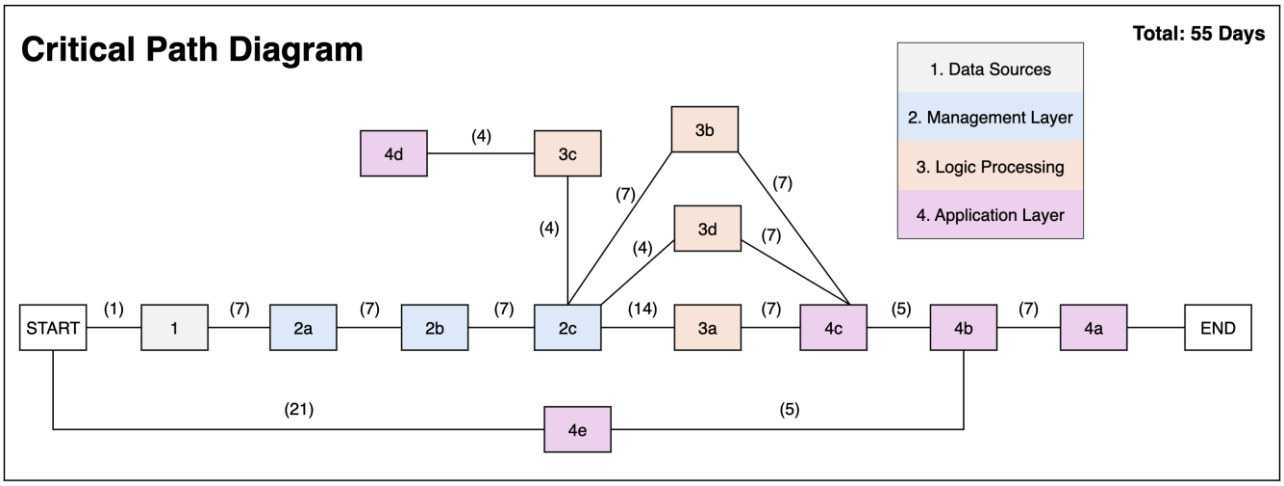

Figure 8

		Activity Days							
		7	14	21	28	35	42	49	56
	4e			21					
	1	1							
	2a	2	8						
	2b		9	15					
	2c			16	22				
Activities	3a				23		36		
	3b				23	29			
	3c				23	26			
	3d				23	26			
	4d				27	30			
	4c						37	43	
	4b							44	48
	4a								49	55

Figure 9

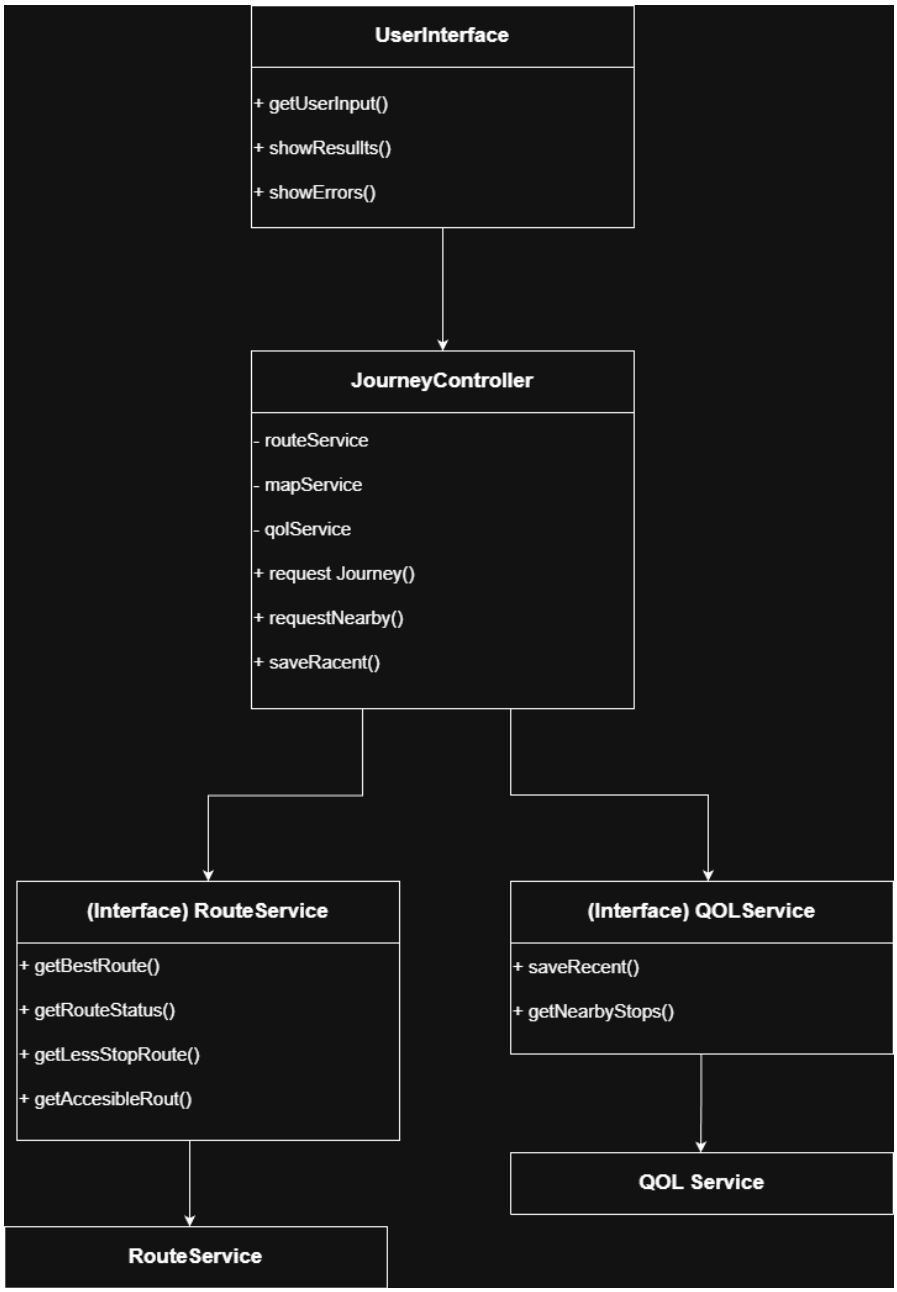
UserInterface
+ getUserInput()
+ showResullts()
+ showErrors()
JourneyController
- routeService
- mapService
- qolService
+ request Journey()
+ requestNearby()
+ saveRacent()
(Interface) RouteService
+ getBestRoute()
+ getRouteStatus()
+ getLessStopRoute()
+ getAccesibleRout()
(Interface) QOLService
+ saveRecent()
+ getNearbyStops()
QOL Service
RouteService

Figure 10

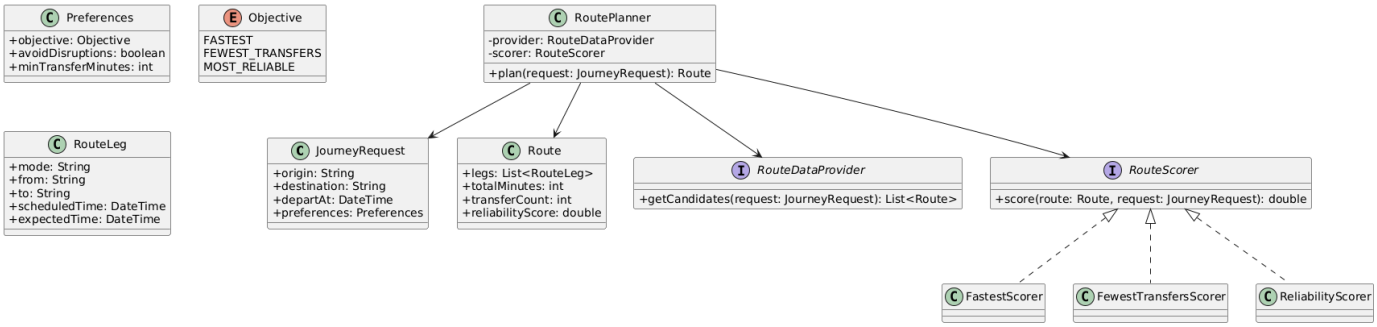

Figure 11

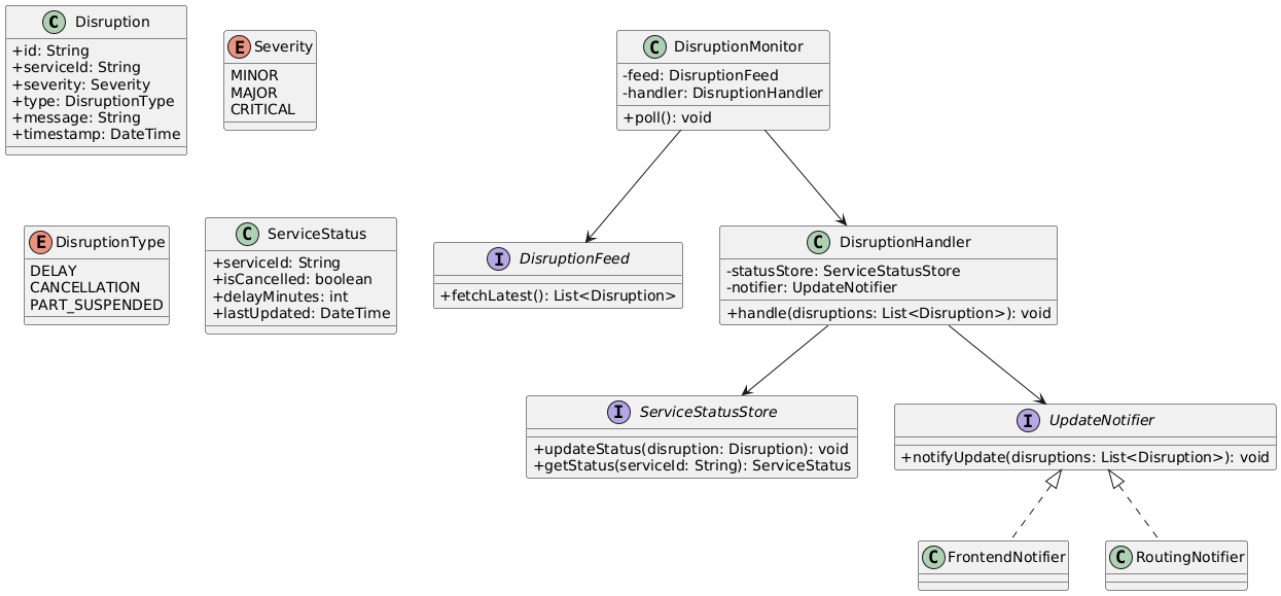

Figure 12

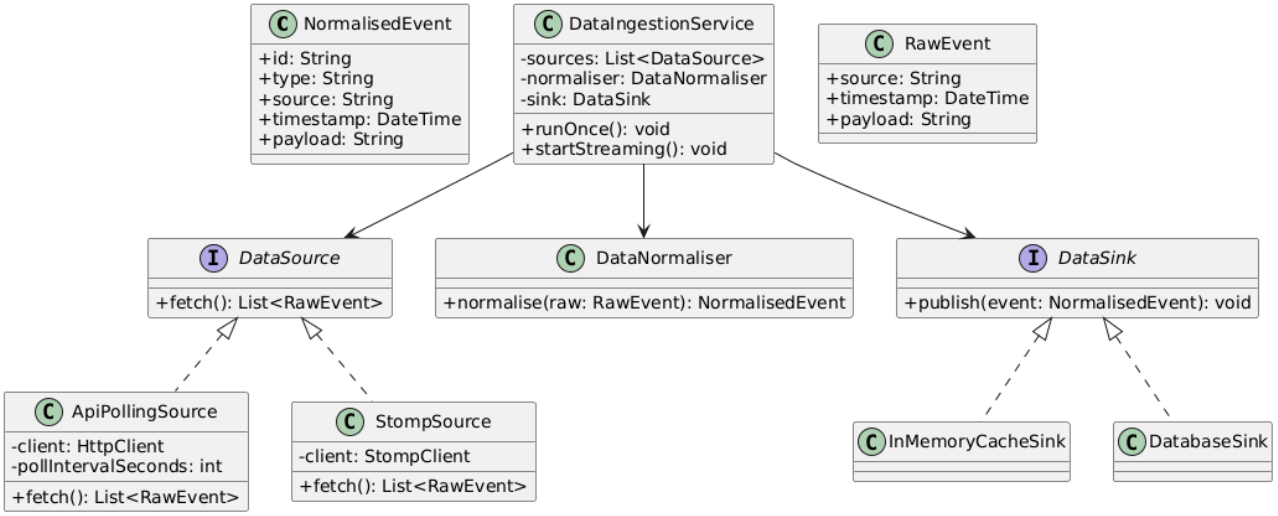

Figure 13

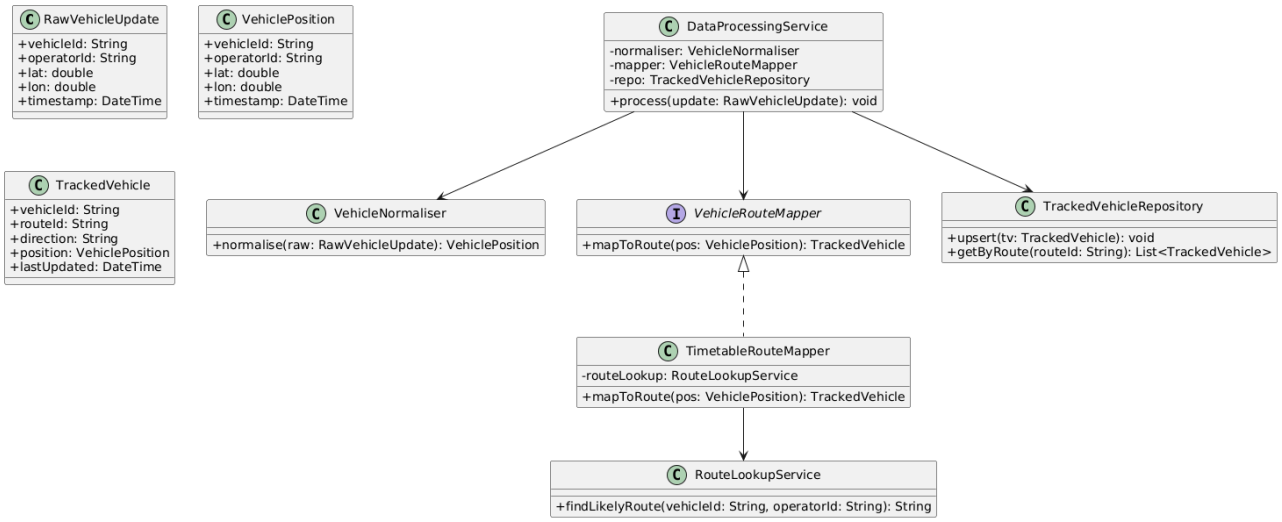

Figure 14

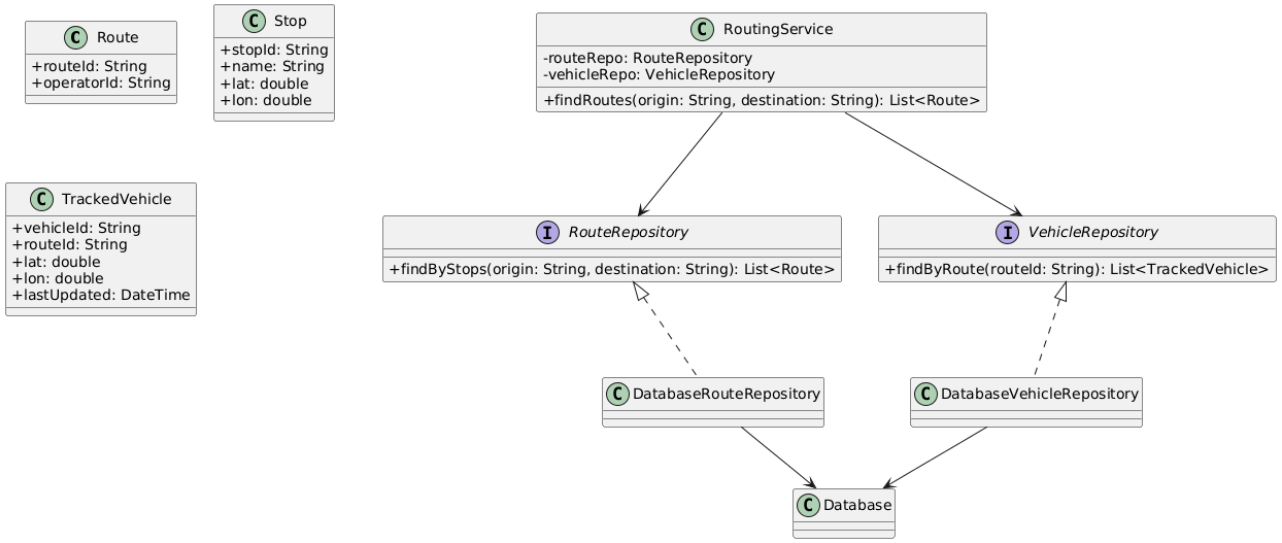

Figure 15