A Public Transport Fare Calculator

Nathan Green — 6068

A-level Computer Science NEA — 7517/C

Table of Contents

Analysis 2

Documented Design 19

Technical solution 60

Testing 112

Conclusion 122

Analysis

Introduction

When catching a bus in the UK, it can often be difficult to work out how much it will cost to make a specific journey, or whether to purchase a single or return ticket, or a day or season pass. If customers are unprepared for the fare they must pay, it can cause stress, or they might simply not have the correct change or payment method. Unfortunately, there is no easy-to-use, publicly accessible service that shows individual fares to the general public.

Public transportation data in the United Kingdom has long been neglected, and until very recently there was no consistent and accurate source of fare data for UK bus journeys. Whilst the introduction of the Bus Open Data Service and legislation to ensure all bus operators publish open data from 2019 have helped, the data is still often confusing and complicated to use, meaning the number of applications currently using all the new data is extremely low.

Existing systems

There are, currently, almost no systems in existence which provide the user with simple and easy to access and understand information on trip fares, e.g., single tickets. The following services are examples of what data is available, and how it is currently used to provide transport customers with rich, real-time information, and also existing transportation apps that don't provide fare data.

Transport for London Plan a journey

TfL provides a comprehensive journey planner system to its customers, that is easy to use and provides many accessibility options to all users. The journey planner allows the user to enter the "From" and "To" locations for their journey, and later specify route options, such as mode of transport, whether the user would like to walk or cycle if it is quicker, etc. The user can then refine their search, e.g., to find a route with the fewest changes. Results

are displayed in a list of possible journeys, each with a breakdown of the timings, how to complete it, and the total price for that journey.

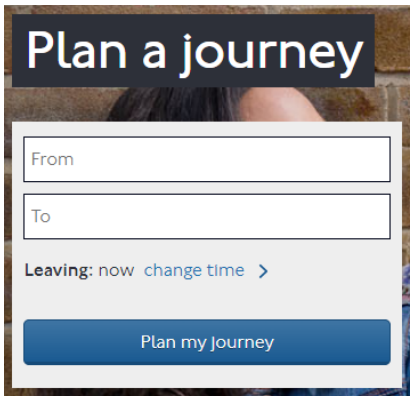


Figure 1. TfL Plan a journey homepage.[1]

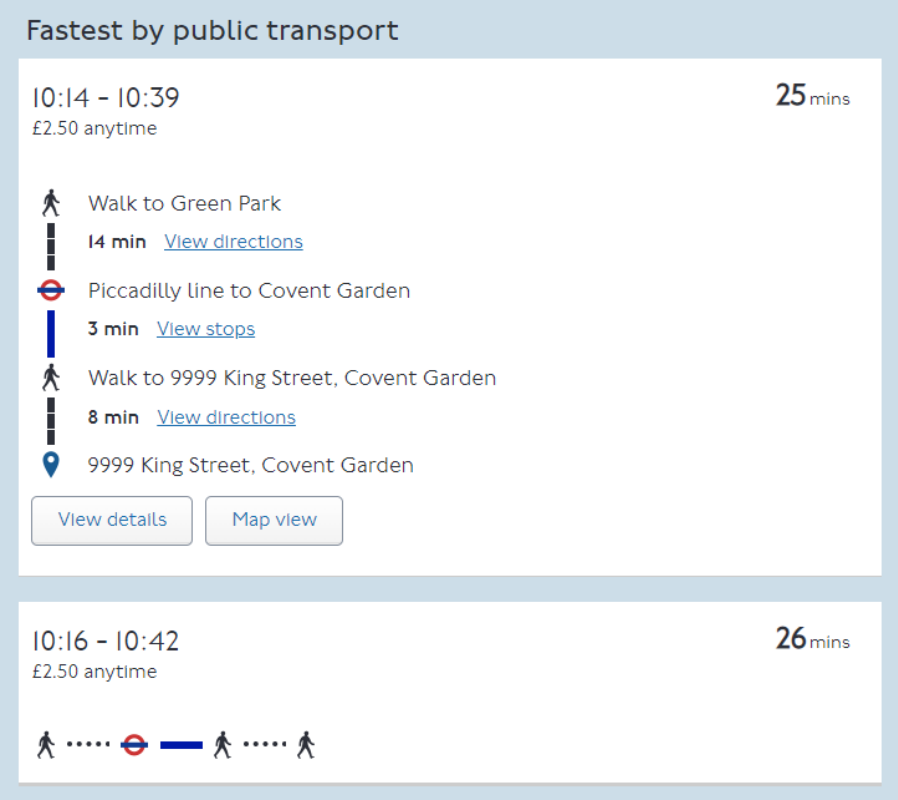


Figure 2. TfL Plan a journey results page.[2]

[1] tfl.gov.uk/plan-a-journey, Retrieved 2022-07-13.
[2] TfL Plan a journey results for Buckingham Palace to Covent Garden, Retrieved 2022-07-13.

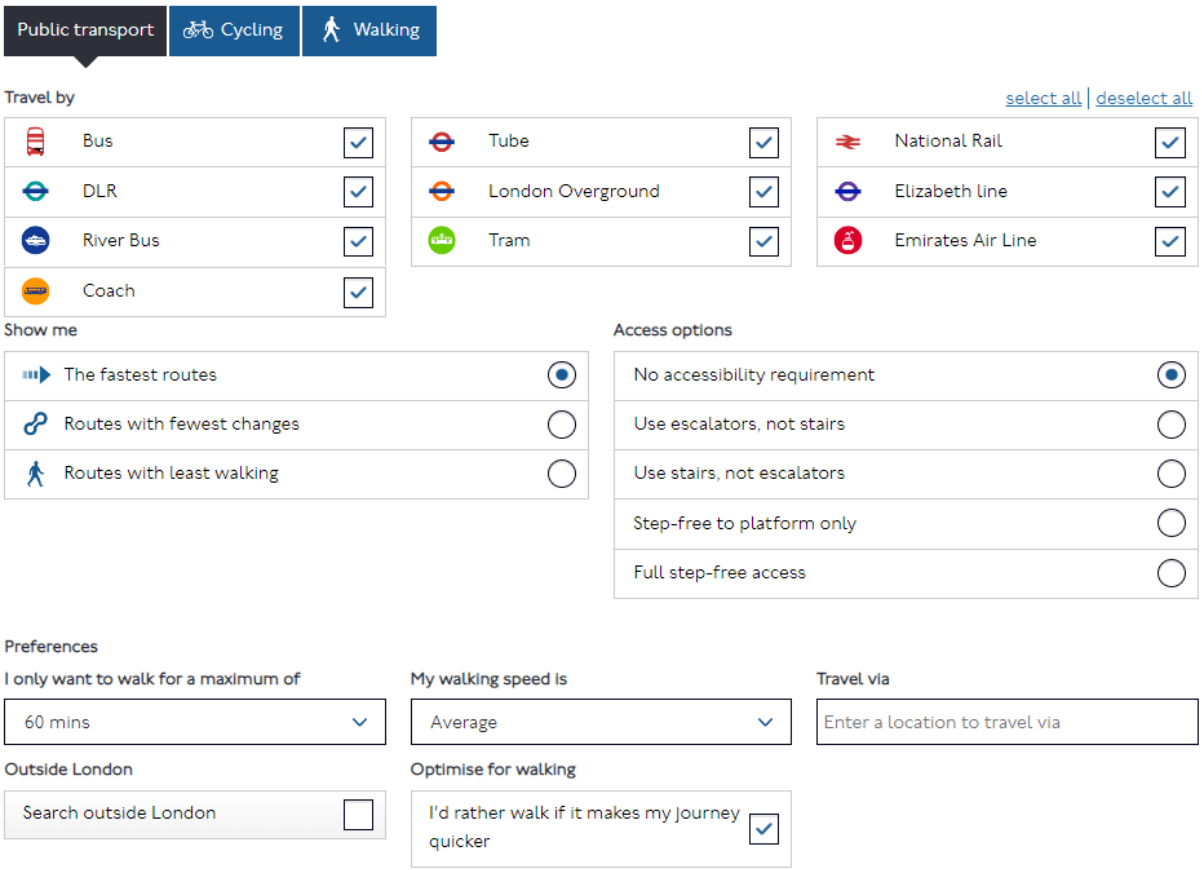


Figure 3. Journey filter options in TfL Plan a journey.

Transdev Go

The bus operator in my area, Transdev Blazefield, provides an online and app-based journey planner, which, similarly to TfL, allows users to enter the start and end points of their journey, and see possible routes to take, which are presented in a list and shown on a map. There are no journey fares displayed to the user, and although the app allows users to purchase mobile tickets, the only tickets available to purchase are passes (such as daily, weekly, etc.), which wouldn't necessarily be the cheapest for the journey, and the app offers little indication of whether a specific ticket is valid on a given journey.

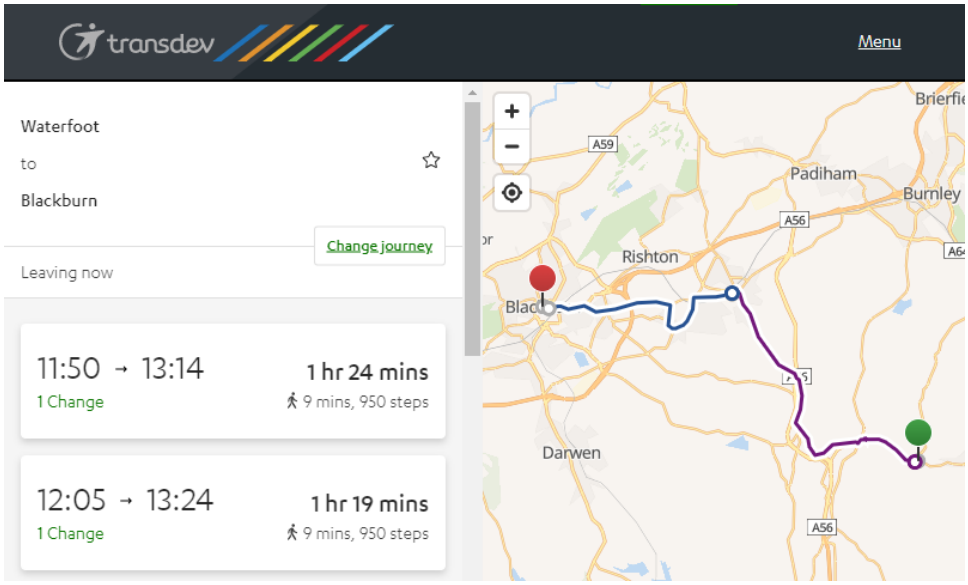


Figure 4. Transdev Go website.

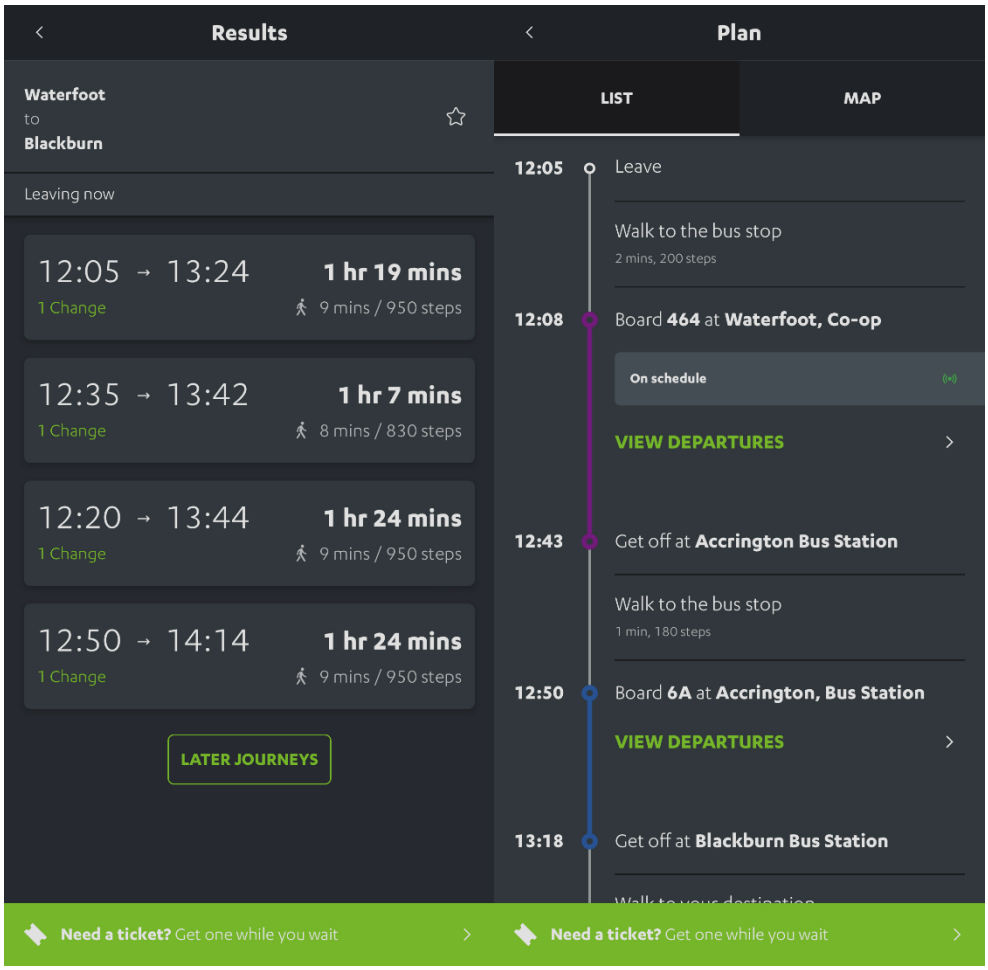


Figure 5. Transdev go mobile app.

HSL — Helsinki Region Transport[3]

The journey planner provided by HSL in Finland offers, like most of the mentioned apps, an easy way to enter two locations and find the journey between them. It offers this in a simple, informative interface, that allows the user to refine their search by time or transit mode, for example.

However, unlike many other systems, there is also a fare-finding functionality, where the user can buy the correct single ticket for their journey directly in the app or on the website. This will be harder to achieve for my local operators, as our single fares are decided by many smaller fare zones, instead of there being only 4 larger fare zones.

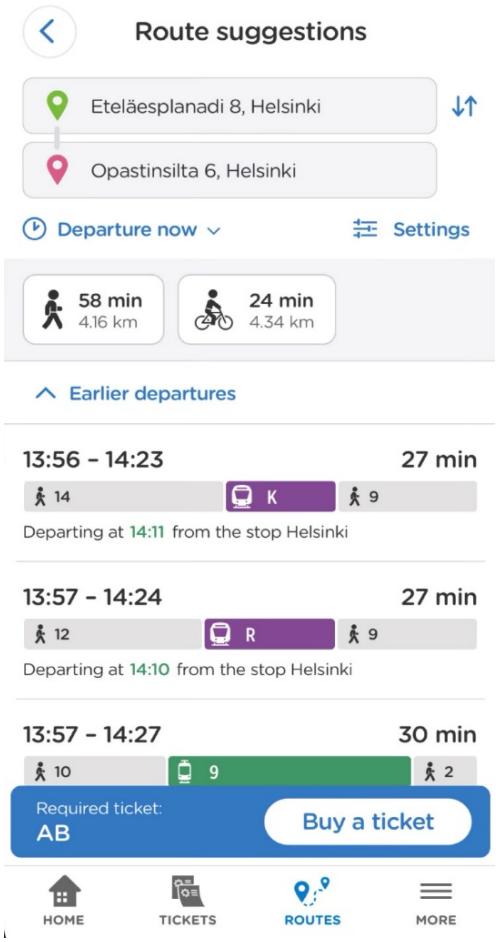
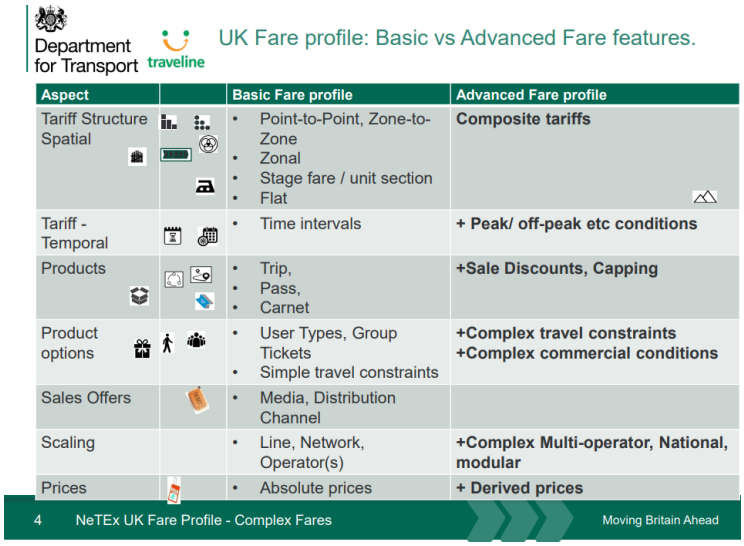


Figure 6. HSL mobile app.

[3] reittiopas.hsl.fi, Retrieved 2022-09-07.

Bus Open Data Service

The Bus Services Act 2017 brought in new legislation that requires all operators of local bus services in England to publish certain data to the Department for Transport's (DfT's) Bus Open Data Service. Operators must publish timetables, bus locations and fares. This legislation was a big step forward for public access to this data, but the data is still hard to process, as it must be published in a mix of different formats for each type of data. This includes timetable data in the *TransXChange 2.4 v1.1A* format, bus location data in the *SIRI-VM v2.0* format, fare data in the *UK NeTEx 1.10* format, and stop data in the *NaPTAN* format. For example, to process live bus location data, a new ZIP file must be downloaded every 10 seconds, extracted, and then the XML file within it searched for the relevant data. The Public Service Vehicles (Open Data) (England) Regulations (2020) set out statutory deadlines for data publishing that bus operators are legally obliged to meet. For example, from 7 January 2021, all operators are obliged to provide basic fares and tickets data to the Bus Open Data Service, and by 7 January 2023, complex fares and ticket data. This means that most of the data that I will need for my application should already be available (if operators are complying with their obligations), with more being published very soon.



Aspect	Basic Fare profile	Advanced Fare profile
Tariff Structure Spatial	• Point-to-Point, Zone-to-Zone • Zonal • Stage fare / unit section • Flat	**Composite tariffs**
Tariff - Temporal	• Time intervals	**+ Peak/ off-peak etc conditions**
Products	• Trip, • Pass, • Carnet	**+Sale Discounts, Capping**
Product options	• User Types, Group Tickets • Simple travel constraints	**+Complex travel constraints** **+Complex commercial conditions**
Sales Offers	• Media, Distribution Channel	
Scaling	• Line, Network, Operator(s)	**+Complex Multi-operator, National, modular**
Prices	• Absolute prices	**+ Derived prices**

Figure 7. Fare slide from DfT BODS presentation.[4]

[4] netex.uk/farexchange/doc/2019.07.16_Workshops/DFT-Workshop2-UKBusFares_NeTEx_3_Complex-2019.07.16-v09.pdf, Retrieved 2022-07-08.

Basic UK Bus fare products?

Access rights		Tariff Structure						
Type of Product	PREASSIGNED FARE PRODUCT	Flat	Point to point	Named Zones	Zone/ Stage Count	Peak / Off Peak	Group Ticket	Temporal Conditions
TRIP ("single ride")	*Short hop*	☑	☑	☑	☑	?	-	No break
	Single trip	☑	☑	☑	☑	☑	☑	Has use by date?
	Time-limited ("Hopper")	-	-	☑	-	☑	☑	Max trip duration, Can interchange
	Period Return	☑	☑	☑	-	☑	☑	Has use by date?
	Day return	☑	☑	☑	-	☑	☑	Must use same day
PASS	*Day pass*	-	-	☑	-	☑	☑	1D (elapsed or calendar)
	Termtime	-	?	☑	-	-	-	Use during term 1Y
	Season pass	☑	☑	☑	-	-	☑	n x D,W,M, 1Y

[5]

The *NeTEx* standard is owned by the *European Committee for Standardisation* (CEN), and the UK profile, *TransXChange* provides UK specific bus fare standards. This profile breaks down the data as follows[6]:

1. Fare Network
    - *Fare Network components:* Stops, tariff zones, fare stages, etc relevant for fare structures.
2. Tariffs
    - *Tariff structures:* point-to-point, zone-to-zone, zonal, section, flat, etc (All described as "access rights" to the network).
    - *Time Intervals:* Durations for day and season period passes e.g., daily, weekly, monthly, annually, etc.
    - *Time Demand Types:* Peak, off-peak and other time bands, expressed as day types.
    - *Unit Distances* for section (stage based) fares.
    - *Unit Quantities* for multi-trip carnets sold at a discount.
3. Products

[5] netex.uk/farexchange/scope.htm, Retrieved 2022-07-08.
[6] netex.uk/farexchange/overview.htm, Retrieved 2022-07-11.

- *Fare Products for travel:* Products for travel such as single and return tickets, multi-trip carnets period passes, etc.
- *Fare Products for travel:* Products defining indirect rights such as sales discount rights, usage-based discounts, stored value, etc.
- *User Types:* User types eligible for different types of fare product (adult, child, student, senior, etc).
- *Group Tickets:* Representing minimum and maximum number of different user types allowed on certain tickets.
- *Usage conditions:* conditions of travel, such as whether interchanges are allowed, etc.
- *Commercial conditions:* Conditions of sale such as exchanges, refunds, purchase windows, etc.

4. Sales Packages
    - *Sales Packages of Fare Products:* bundling one or more Fare products with media and distribution information.
    - *Distribution:* Information on the available distribution channels and fulfilment methods (including restrictions by media).
    - *Ticket media:* paper, smartcard, mobile apps, etc.
5. Prices
    - *Fare Prices* for any combination of relevant fare component.

This data is split into multiple XML files, which can then be processed to find the required information.

TransportAPI

TransportAPI supplies UK transport data as a managed service to many large companies, such as *First Group*, *Heathrow Airport*, and *Ford*. The data is available through six different managed services: *TAPI Journey Planner*, *TAPI Bus Information*, *TAPI Bus Fares*, *TAPI Bus Performance*, *TAPI Rail Information*, and *TAPI Places*, however the Bus Fares and Performance services are only in “coming soon” availability, so the platform still doesn’t

provide a way to give customers the rich data they need. Additionally, data from this service is only available commercially, so isn't accessible to the general public.

Transport for London Unified API

In 2015, Transport for London introduced a new 'Unified API' for its services across the capital, focussing on making key public information, including fares, across all modes of transport simpler. It achieves this by presenting all similar data in the same format, with consistent structures, no matter the mode of transport or what forms the underlying data is in. This allows developers to write code once and access all types of data easily, making multi-mode application development easier. Complexity not needed for consumer facing applications is removed. Unfortunately, this API is only available for transport data in London and a similar simplified service that includes fare data does not exist for the rest of the country. My solution will therefore, most likely, need to interact directly with data from the Bus Open Data Service.

bustimes.org

The bustimes.org website provides an easy way to view very detailed data about bus services across the UK. However, the fare data on the platform is not yet fully useful, as it is only experimental, and a user must know the operator/service, and then manually navigate the fare table in order to find the information they are looking for.

bustimes.org Search Search Map

Home ›

Fares

An overview of fares data. This page isn't very useful – eventually, fares data will be integrated into other parts of the website.

Known issues:

- Some data confuses pounds and pence – e.g. a £3 fare is given as £0.03
- Fare stages may have slightly different names in different bits of data. bustimes.org isn't clever enough to work out they're the same yet
- bustimes.org only supports point to point fares so far, not day/week/month/zone tickets

Datasets

Publisher	Modified	Description
Go-Ahead	8 Jul 2022	Go North East
Rotala Group of Companies	8 Jul 2022	Diamond Bus Soutl
Go-Ahead	7 Jul 2022	Go South West, inc Plymouth Citybus a Cornwall Bus fares.
Arriva UK Bus	7 Jul 2022	Fares for Arriva Me (AMSY) services

Figure 8. bustimes.org fares page.[7]

Transdev_6236_20220106 11:56:29

Fares data from Transdev/Bus Open Data Service, 30 May 2022

Transdev Lancashire

From:

To:

Find fare

Tariffs

Transdev Lancashire Dales Rover fares (30 May 2022–30 May 2122)

Transdev Lancashire Day Return fares

Transdev Lancashire Daytripper Duo fares (19 March–30 April 2019)

Transdev Lancashire DayTripper Plus fares (30 May 2022–30 May 2122)

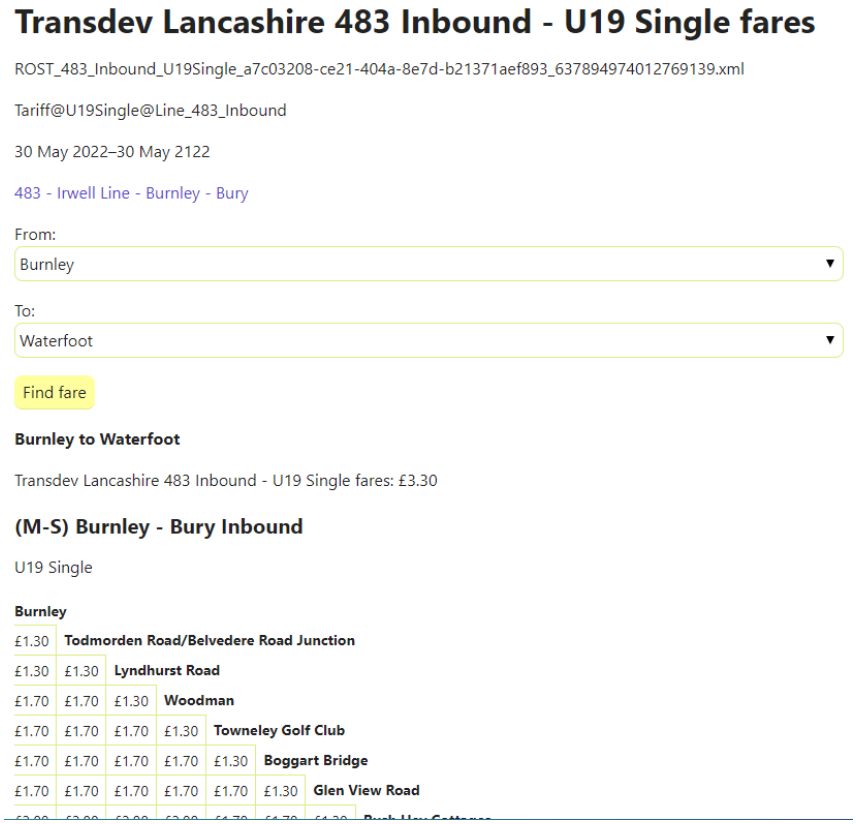
Transdev Lancashire DayTripper Plus fares

Transdev Lancashire DayTripper fares

Figure 9. bustimes.org fares for operator.[8]

[7] bustimes.org/fares, Retrieved 2022-07-08.
[8] bustimes.org/fares/datasets/1135, Retrieved 2022-07-08.

This is difficult for the end user, as they must trawl through what is potentially hundreds of different tariffs and journey types just to find the right fare.



Transdev Lancashire 483 Inbound - U19 Single fares

ROST_483_Inbound_U19Single_a7c03208-ce21-404a-8e7d-b21371aef893_637894974012769139.xml

Tariff@U19Single@Line_483_Inbound

30 May 2022–30 May 2122

483 - Irwell Line - Burnley - Bury

From:

Burnley

To:

Waterfoot

Find fare

Burnley to Waterfoot

Transdev Lancashire 483 Inbound - U19 Single fares: £3.30

(M-S) Burnley - Bury Inbound

U19 Single

Burnley						
£1.30	**Todmorden Road/Belvedere Road Junction**					
£1.30	£1.30	**Lyndhurst Road**				
£1.70	£1.70	£1.30	**Woodman**			
£1.70	£1.70	£1.70	£1.30	**Towneley Golf Club**		
£1.70	£1.70	£1.70	£1.70	£1.30	**Boggart Bridge**	
£1.70	£1.70	£1.70	£1.70	£1.70	£1.30	**Glen View Road**

Figure 10. bustimes.org fare table.[9]

Users can use the drop down to select the stops or fare zones they want to travel between, and then select *Find fare* to get the price of that journey. There is also a fare table, which presents all the fares for the selected tariff and service, which although complicated, users can navigate to explore different fare options, for example saving money by walking to the next bus stop to be in a different fare zone. The fare data used on this website comes from the Bus Open Data Service, as discussed above.

Client Survey

I conducted a client survey to inform my decisions during the design stage. I asked people from a variety of different groups, from classmates to adults in my community, as I want

[9] bustimes.org/fares/tariffs/1234923?origin=36922&destination=37297, Retrieved 2022-07-08.

my solution to be helpful to almost everyone; even people who rarely use public transport may want to use it more but find the lack of fare data a barrier to more regular use. I have summarised the questions and my findings for each below.

1. Have you used a public bus service in the last year?
    - Yes.
    - No.

    Most of the respondents said yes, with 78% having caught a bus within the last year. This suggests a significant potential user base for my solution, who likely understand some of the issues with many current public transport apps.

2. Are any of these barriers for you to using a public bus service?
    - Services are not frequent enough.
    - Fares confusing/hard to find information about.
    - Bus doesn't run where/when I want.

    Over half of respondents found all off the options to be barriers to using a public bus service. Of those who hadn't used a public bus service in the past year, 80% of respondents believed that confusing fares and a lack of information about them were a barrier to them. This shows that the use case of my solution may be in relatively high demand, and that fare data should be presented in a clear, easy to understand manner so that it is accessible to as many people as possible. To help address the other barriers, my solution should display when and where buses run to help users make informed decisions about their journeys.

3. Do you fall into any of the following special fare categories?
    - Under 11.
    - Under 19.
    - I have a card which entitles me to cheaper fares.

Over half of respondents fell into at least one of these categories, highlighting the importance of presenting all the available fares for a journey, such as reduced child fares, not just the 'default' adult fares.

4. Have you ever purchased a pass such as a day rover because you didn't know if the single price would be cheaper?
    - Yes.
    - No.

With 65% of respondents saying yes, this suggests my solution needs to clearly display the cheapest price for a journey. It could also potentially include a comparison with the cheapest day ticket available, for example, to highlight the savings.

5. In a bus fare app, would you prefer the fare calculation to be built into a journey planner, or separate?
    - Built in.
    - Separate.

Over 90% of respondents would prefer a fare calculator built in as part of a journey planner, and not just standalone, which reflects similarly to my comments on the fares section of the existing system *bustimes.org*. Therefore, my solution will need to provide a robust journey planning platform, with fare data provided alongside this.

6. How important is access to real-time information, e.g., live bus arrival times, to you?
    - Essential.
    - It would be nice to have, but I don't need it.
    - I wouldn't use it.

The overwhelming majority of respondents indicated that this would be a 'nice to have' feature, rather than an essential, and therefore whilst I will consider it, it

won't be prioritised over the core functionality due to the complex nature of the implementation of live data.

Platform

I have chosen to write the backend in Go, due to its speed, portability of built programs, excellent standard library of functions, and great scaling for concurrency. The frontend will be written using standard web technologies: HTML, CSS, and JavaScript/TypeScript. I have also chosen to use the web framework Svelte (and its app framework SvelteKit), which allows a robust, performant, and reactive application to be built quickly and easily, as passing data around the app is made simple.

For the actual route planning, I have chosen to use OpenTripPlanner, a popular open-source trip planner. Coding my own routing solution would be too complex for the scope of this project. This is detailed more on page 24. OpenTripPlanner is written in Java, and provides a REST API that can be consumed by the frontend and Go backend.

For data storage, I will use and store data from the Bus Open Data Service in a database, namely PostgreSQL, as it provides a JSON datatype, which allows the complex and irregular structure of the fare XML files to be represented, and indexed across multiple keys. This would be extremely difficult in a traditional table format, and would require extensive and complex ORM (object-relational mapping) in order to use the data in the backend.

System Objectives

The system will provide a web interface that allows the user to enter a start and end location for a journey. It will then calculate potential routes to complete that journey, using a mixture of walking and available public transport networks, calculate the different prices to complete each route, and then display the results to the user.

This system can be broken down into a few components.

1. Web frontend — provides the user interface, allowing users to search for different locations, choose journey options and initiate the search.
2. Main backend — this will download and process fare data, find fares for journeys, and return them to the frontend to be displayed to the user.
3. OpenTripPlanner backend — this will provide the country-wide routing system, which provides data to both the frontend and main backend.

Which can then be broken down further, into detailed system objectives.

1. User can enter a ‘from’ location for the journey.
2. The ‘from’ field can autocomplete entered locations from a geocoding API.
3. User can enter a ‘to’ location for the journey.
4. The ‘from’ field can autocomplete entered locations from a geocoding API.
5. User can select a date for the journey.
6. User can select a time for the journey.
7. User can select whether the journey should leave at, or arrive by, the time selected.
8. User can click a button to initiate the journey search.
9. A list of possible journeys will be displayed to the user in a list.
10. In the list, a summary of each journey will be presented.
11. The summary will include the start and end times of the journey, and its length.
12. The summary will show the legs of the journey as proportional bars.
13. The summary will show the total fare for that journey, if available.
14. The summary will show the live arrival time of the first transit leg in the journey, if available.
15. The user will be able to select a journey to see a detailed view of the legs of that journey.
16. A list of possible fares will be shown for each transit leg in the journey.
17. There is a map for the user to plan and visualise journeys.
18. If the user selects a point on the map, a popup is presented.
19. The popup contains the name of the location that was pressed.
20. The popup contains ‘from here’ and ‘to here’ buttons that can set the ‘from’ and ‘to’ fields.

21. The selected journey from the results is drawn onto the map.
22. Each leg of the journey is drawn as a line on the map.
23. Walking legs will be drawn as a dotted line.
24. Transit legs will be drawn in the line's colour, with the name of the line overlaid on it.
25. The map will zoom to fit the whole journey when it is selected.
26. The frontend will send a request to an OpenTripPlanner instance for a journey plan with the parameters in objectives 1, 3, 5, 6, and 7, when they select the button in objective 8.
27. The frontend will display the received journey plan in the format described between objectives 9 and 25.
28. The frontend will send the journey plan to the backend, requesting fares.
29. The frontend will add the fare data to the already shown journey data once it is received.
30. The backend will be able to download fare data from the Bus Open Data Service.
31. The fare data will be stored in a database.
32. The backend will expose an API that can handle an OpenTripPlanner journey sent by the frontend.
33. The backend will query the database and then filter the results to find all the possible fares for each leg of the journey.
34. The API will return the fares to the frontend once they are found.
35. The backend will also expose an API to which returns the colours of different bus routes.
36. The backend will store and retrieve the colour data of bus routes.

Simplified Data Flow Diagram

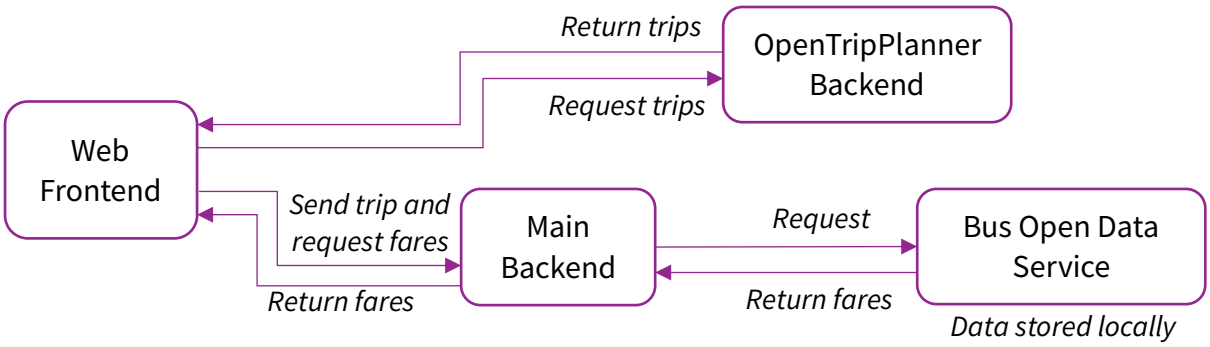


Figure 11. Simplified model of app data flow.

In summary, the frontend combines relevant data from OpenTripPlanner and the Bus Open Data Service, displaying rich, useful information to the user.

Documented Design

Introduction 20
Technology and Platform 20
Modular Design 20
Data Structure/Class Diagram 22
OpenTripPlanner 24
Database structure 28
Finding fares for a trip 29
  Finding the fares for an entire trip 29
  Finding the fares for an itinerary leg 31
  Finding fares from a specific `FareObject` 34
  Trimming OTP IDs 37
Managing the dataset 37
  Getting the latest datasets 37
  Adding a directory of data 38
  Adding a ZIP file 39
  Adding an XML file 39
  Adding a `FareObject` to the database 39
Converting between NOC and agency ID 40
Getting the colour of a line 41
Exposing the API 42
  Getting fares with the API 42
  Getting colours with the API 43
Running the backend 43
Creating the frontend 45
Embedding the frontend in the backend 46
Designing the frontend 47
  Initial layout 47
  Improving usability 49
  Responsive design 50
Managing state and data 54
Frontend utilities 56
  Constants 56
  Time management 56
  Colour management 56
  Geocoding 58

Introduction

I am creating a bus fare calculator, where the user can enter the locations of a journey they would like to make, and be presented with a list of the potential ways they could make that journey, and how much it would cost.

Technology and Platform

As discussed on page 15, my program can be divided into two parts, a backend, and a frontend. The backend is written in Go, and the frontend in JavaScript/TypeScript/Svelte.

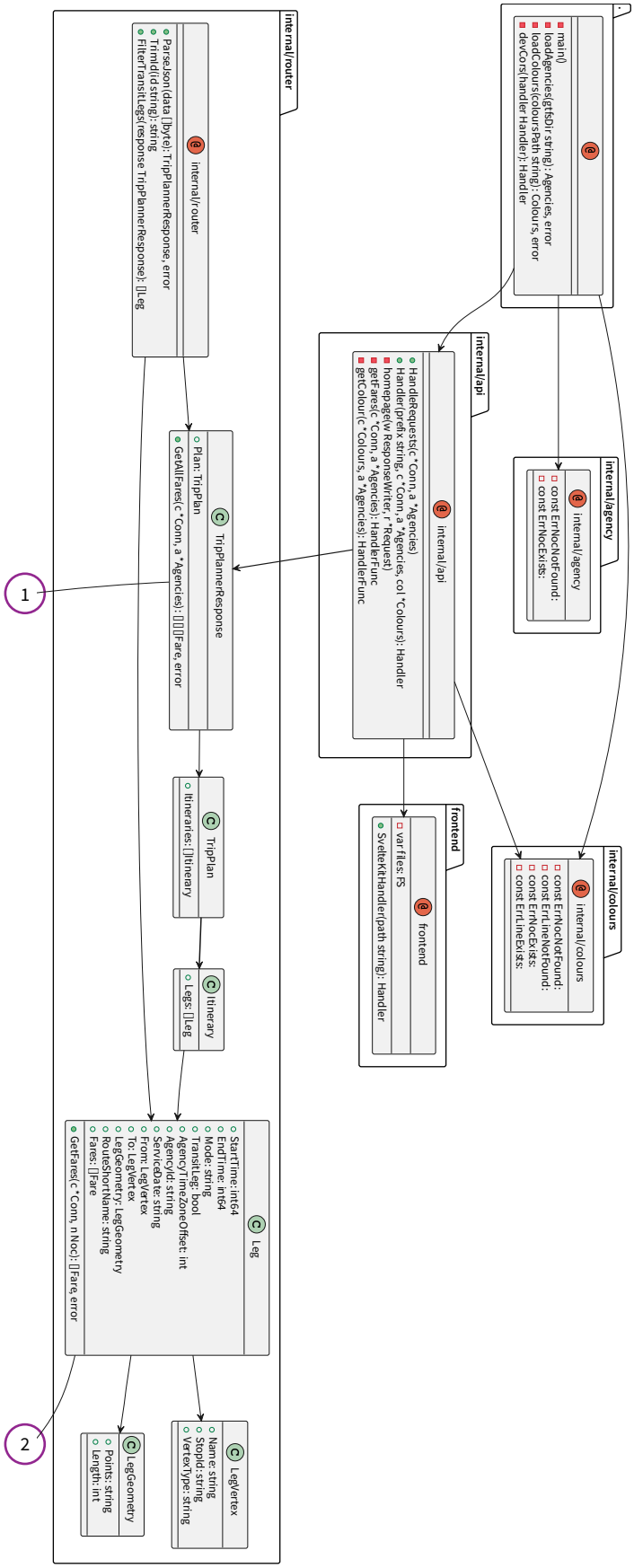
Modular Design

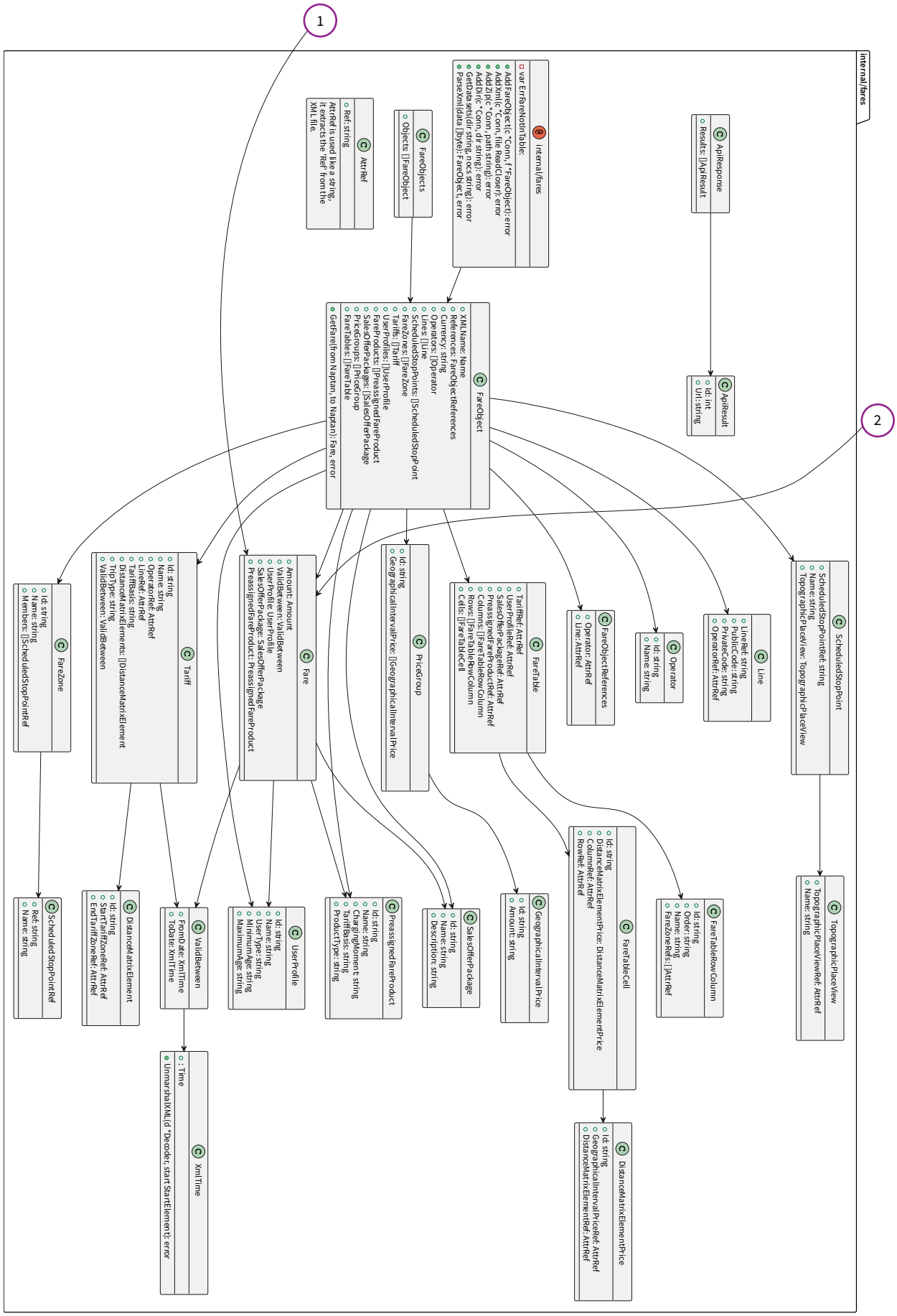
The backend of my project will be designed with a modular structure, with sections that perform different tasks being written independently from each other in different modules. This is useful as it allows each module of the program to be written and tested without depending on other modules, meaning I can use mock data to easily test different parts, and am able to extend or change logic in one module without impacting the functionality of other ones. To achieve this, each module will provide relevant data structures and methods that can be passed into and called from other modules, allowing an efficient flow of data throughout the program. The main modules I have identified are as follows:

- Main — Launches the program and other modules.
- Router — Processes OpenTripPlanner trip plans, and uses the fares module to get the possible fares for each transit trip leg.
- Fares — Downloads, updates, and imports fare datasets from the Bus Open Data Service. Calculates fares between given stops.
- API — Provide a REST interface to interact with internal modules, e.g., from the frontend.
- Agency — Utilities to convert between agency IDs used in OpenTripPlanner GTFS timetable data (e.g., ‘OP291’), and National Operator Codes used in fare data (e.g., ROST).

- Frontend — Embedded web application which interacts with the API and provides a user interface for the backend services.

Data Structure/Class Diagram




internal/fares
ApiResult
Results: []ApiResult
ApiResult
Id: int
Uri: string
internal/fares
var ErrFareNotInTable:
AddFareObject(c *Conn, f *FareObject): error
AddXml(c *Conn, file ReadCloser): error
AddZip(c *Conn, path string): error
AddDir(c *Conn, dir string): error
GetDatasets(dir string, nocs string): error
ParseXml(data []byte): FareObject, error
FareObjects
Objects: []FareObject
AttrRef
Ref: string
AttrRef is used like a string, it extracts the 'Ref' from the XML file.
FareObject
XMLName: Name
References: FareObjectReferences
Currency: string
Operators: []Operator
Lines: []Line
ScheduledStopPoints: []ScheduledStopPoint
FareZones: []FareZone
Tariffs: []Tariff
UserProfiles: []UserProfile
FareProducts: []PreassignedFareProduct
SalesOfferPackages: []SalesOfferPackage
PriceGroups: []PriceGroup
FareTables: []FareTable
GetFare(from Naptan, to Naptan): Fare, error
ScheduledStopPoint
ScheduledStopPointRef: string
Name: string
TopographicPlaceView: TopographicPlaceView
Line
LineRef: string
PublicCode: string
PrivateCode: string
OperatorRef: AttrRef
Operator
Id: string
Name: string
FareObjectReferences
Operator: AttrRef
Line: AttrRef
FareTable
TariffRef: AttrRef
UserProfileRef: AttrRef
SalesOfferPackageRef: AttrRef
PreassignedFareProductRef: AttrRef
Columns: []FareTableRowColumn
Rows: []FareTableRowColumn
Cells: []FareTableCell
PriceGroup
Id: string
GeographicalIntervalPrice: []GeographicalIntervalPrice
TopographicPlaceView
TopographicPlaceViewRef: AttrRef
Name: string
FareTableRowColumn
Id: string
Order: string
Name: string
FareZoneRefs: []AttrRef
FareTableCell
Id: string
DistanceMatrixElementPrice: DistanceMatrixElementPrice
ColumnRef: AttrRef
RowRef: AttrRef
GeographicalIntervalPrice
Id: string
Amount: string
SalesOfferPackage
Id: string
Name: string
Description: string
PreassignedFareProduct
Id: string
Name: string
ChargingMoment: string
TariffBasis: string
ProductType: string
DistanceMatrixElementPrice
Id: string
GeographicalIntervalPriceRef: AttrRef
DistanceMatrixElementRef: AttrRef
Fare
Amount: Amount
ValidBetween: ValidBetween
UserProfile: UserProfile
SalesOfferPackage: SalesOfferPackage
PreassignedFareProduct: PreassignedFareProduct
UserProfile
Id: string
Name: string
UserType: string
MinimumAge: string
MaximumAge: string
ValidBetween
FromDate: XmlTime
ToDate: XmlTime
XmlTime
: Time
UnmarshalXML(d *Decoder, start StartElement): error
Tariff
Id: string
Name: string
OperatorRef: AttrRef
LineRef: AttrRef
TariffBasis: string
TripType: string
DistanceMatrixElements: []DistanceMatrixElement
ValidBetween: ValidBetween
DistanceMatrixElement
Id: string
StartTariffZoneRef: AttrRef
EndTariffZoneRef: AttrRef
FareZone
Id: string
Name: string
Members: []ScheduledStopPointRef
ScheduledStopPointRef
Ref: string
Name: string
1
2

The data structures in my project are mostly defined by those used in the fare data I will be importing, as this allows the data to be imported more easily. For example, the `FareObject` type represents an imported fare XML file, and the structures stored within it, such as `FareTables` and `ScheduledStopPoints` are near-to-exact representations of the data structure used in the original files, albeit with extra information not needed to calculate fares removed. As the number of `FareObjects` required is very large (almost 4000 for just one bus operator), loading them into and storing them in memory became very inefficient, with memory usage growing quickly into the gigabytes, and searching for a required `FareObject` requiring a linear traversal of all those in the dataset. As a solution for this, I have decided to store `FareObjects` in a database, namely PostgreSQL. My data structures don't easily map to a traditional table layout, as each `FareObject` needs to be easily accessed as a whole, so I will be using the database as more of a fast persistent store of `FareObjects`. I have chosen to use PostgreSQL as it has a 'jsonb' datatype, which allows each `FareObject` to be stored as JSON in a table, and efficiently indexed by the keys that need to be searched for.

OpenTripPlanner

To get bus fares for a given journey, my program first needs to know what that journey is. To complete this task, I have chosen to use OpenTripPlanner, a popular open-source trip planner that runs entirely on open data, so can be run locally without relying on third-party services. Additionally, coding my own routing solution would be both too complex for the scope of this project, and counter-productive when a more efficient solution already exists. There is no need to reinvent the wheel (or trip planner) here. Another reason I chose OpenTripPlanner over commercial solutions such as the Google Maps routing API, is that OTP returns data in much more detail, with the identifiers of stops needed to search fare data still included in query responses. My OpenTripPlanner instance will run under Docker, as this allows for easy deployment in almost any environment. For the purposes of this project, I will only be using data for the North of England, to make it easier to manage. With more powerful hardware, the project could be quite easily scaled up to work for the whole country.

To configure the OTP instance, we first need to download the latest OpenStreetMap data, which contains information about roads and places.

```
wget -I osm_links.txt
```

We first download the latest OSM extracts, where `osm_links.txt` contains links to the following data files:

```
http://download.geofabrik.de/europe/great-britain/england/cheshire-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/cumbria-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/durham-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/east-yorkshire-with-hull-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/greater-manchester-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/lancashire-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/merseyside-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/northumberland-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/north-yorkshire-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/south-yorkshire-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/tyne-and-wear-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/west-yorkshire-latest.osm.pbf
http://download.geofabrik.de/europe/great-britain/england/lincolnshire-latest.osm.pbf
```

```
osmium merge *.osm.pbf -o north-england-latest.osm.pbf
```

This OSM data can then be combined into one file, using an open-source tool called Osmium, creating a single file with all the OSM data.

```
osmium tags-filter north-england-latest.osm.pbf w/highway w/public_transport=platform w/railway=platform w/park_ride=yes r/type=restriction r/type=route -o north-england-latest-filtered.osm.pbf -f pbf,add_metadata=false
```

The data is then filtered to include only the tags that OTP needs to create its routing graph.

The next data we need is GTFS timetable data from the Bus Open Data Service, to know which services run when, and where.

```
wget --content-disposition -i gtfs_links.txt
```

This is downloaded similarly to above, where `gtfs_links.txt` contains the following:

```
https://data.bus-data.dft.gov.uk/timetable/download/gtfs-file/north_east/
https://data.bus-data.dft.gov.uk/timetable/download/gtfs-file/north_west/
https://data.bus-data.dft.gov.uk/timetable/download/gtfs-file/yorkshire/
```

Next, a national elevation map is downloaded from Ordnance Survey, which is used to calculate walking time for routes with inclines, for example. It is stored as `OSTerr50.tif`.

```
{
  "osmWayPropertySet": "uk",
  "platformEntriesLinking": true
}
```

Now, the build settings are configured with `build-config.json`, which configures OTP for use in the UK.

```
java -Xmx68G -jar otp-2.2.0-shaded.jar --buildStreet .
```

The first part of the build is now run, which uses the OSM and elevation data to generate a street graph, on which transit data can later be layered. This is saved as `streetGraph.obj`. The Java Virtual Machine is given access to 68GB RAM, as the build process is extremely memory intensive.

```
java -Xmx68G -jar otp-2.2.0-shaded.jar --loadStreet --save .
```

Next, the transit graph is built onto the street graph, creating `graph.obj`, which can be used to run the OTP instance.

Now all the data needed to run the instance has been generated. To run OTP in Docker, a new Docker image needs to be built, as the publicly available image, `opentripplanner/opentripplanner`, is only compatible with amd64 architectures, whereas my server runs on the arm64 platform. The Dockerfile to generate the image is as follows:

```
FROM arm64v8/eclipse-temurin:17
RUN mkdir /opt/app
```

```
WORKDIR /opt/app
COPY otp-2.2.0-shaded.jar .
EXPOSE 8077
CMD [ "java", "-Xmx12G", "-jar", "otp-2.2.0-shaded.jar", "--port", "8077",
"--load", "." ]
```

The Docker image is based on `eclipse-temurin`, which is the official image for running `.jar` files with OpenJDK, a distribution of Java.

The Dockerfile creates the working directory, `/opt/app`, and then copies a copy of the OTP binary into it. The port that the instance will run on, 8077, is exposed from the Docker container, and then the file is run with the `--load` parameter, loading all the data files from the same directory. Only 12GB RAM is required to run the built graphs.

As the Docker image loads the graph and config files from the /opt/app directory, this is where those files need to be mounted in the image. I am using docker-compose, which allows for an easy deployment and configuration on any platform. The file uses the image built previously, and mounts the required graph and configuration files into the container. The image also contains configuration for Traefik, the reverse proxy running on my server, which allows the OTP instance to be accessed externally. The `docker-compose.yml` file contains the following:

```
version: '3.1'
services:
  otp:
    image: nathangreen/otp:2.2.0-arm64
    container_name: otp
    restart: unless-stopped
    volumes:
      - ./graph.obj:/opt/app/graph.obj
      - ./streetGraph.obj:/opt/app/streetGraph.obj
      - ./otp-config.json:/opt/app/otp-config.json
      - ./router-config.json:/opt/app/router-config.json
    networks:
      - traefik-net
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=traefik-net"
      - "traefik.http.routers.otp.rule=Host(`otp.nat.omg.lol`)"
      - "traefik.http.routers.otp.tls=true"
      - "traefik.http.routers.otp.tls.certresolver=letsencrypt"
networks:
  traefik-net:
    external: true
```

Now, running `docker-compose up -d` starts the OTP server.

Database structure

The SQL used to create the table and related indexes for the database is as follows:

```
create extension if not exists "uuid-ossp";
drop table if exists fares;
create table fares (
  id uuid default uuid_generate_v4() primary key,
  fare_object jsonb not null
);
drop index if exists scheduled_stop_points_index;
drop index if exists lines_index;
create index if not exists lines_index on fares using gin
(
  (fare_object->'Lines') jsonb_path_ops
);
create index if not exists scheduled_stop_points_index on fares using gin
(
  (fare_object->'ScheduledStopPoints') jsonb_path_ops
);
```

First, the UUID extension is initialised, which is needed to generate a unique ID for each fare object in the table. After dropping any relevant existing tables, a new table `fares` is created, with two columns, an `id` of type UUID (universally unique identifier) as the primary key, and the `fare_object` as type jsonb, which stores JSON data in a decomposed binary format that makes it significantly faster to process than 'pure' JSON. It also allows indexing, which is what is set up next. First, any existing indexes are dropped, and then two indexes are created, `lines_index` and `scheduled_stop_points_index`. These are GIN (Generalised Inverted Index) indexes, which stores the locations that the indexed items occur in, and *"can be used to efficiently search for keys or key/value pairs occurring within a large number of jsonb documents"*[10] The details of the lines that each fare object refers to and the scheduled stop points within it are indexed, as these are used to query the database later.

[10] PostgreSQL 15 Documentation, 8.14.4 `jsonb` Indexing.

Finding fares for a trip

Overall, and as explained below, three methods are used to find fares for a trip. Firstly, `TripPlan.GetAllFares`, which calls `Leg.GetFares`, which calls `FareObject.GetFare`.

Although not displayed in the pseudocode, my design includes robust error checking throughout, using Go's standard methods for this. Almost all methods return at least two values, the expected value and an `err` of type `error`, which can be checked using `if err ≠ nil` (the Go equivalent to null) and handled accordingly.

Finding the fares for an entire trip

To find the fares for a trip, the trip plan received from the API is first decoded from JSON into a struct (an object), only keeping the information needed to find fares. Then, a method that runs on the `TripPlannerResponse` is called, '`GetAllFares`', which returns a 3D array of fares, representing the same structure from the trip plan of many journeys, each which can contain many legs, each which can contain many fares.

For example, a `TripPlan` with the following structure:

- Itinerary
    - Leg (Walk)
    - Leg (Bus)
    - Leg (Walk)
- Itinerary
    - Leg (Walk)
- Itinerary
    - Leg (Walk)
    - Leg (Bus)
    - Leg (Walk)
    - Leg (Bus)

Would return a fare array with a corresponding structure, where each leg can contain any number of fares:

```
[
  [ [], [fare, fare, etc.], [] ],
  [ [] ],
  [ [], [fare, fare], [], [fare] ]
]
```

In the following pseudocode for `GetAllFares`, `r` is the decoded trip plan as type `TripPlannerResponse`, and `TrimId` and `GetNoc` are utility functions, explained later. The `Leg.GetFares` method is also called, which is explained in the next section.

```
itineraryLengths ← []
FOR itinerary IN r.Plan.Itineraries
  APPEND(itineraryLengths, LEN(itinerary.Legs))
ENDFOR

itineraryFares ← [][][] (LEN(r.Plan.Itineraries))

FOR i ← 0 TO LEN(r.Plan.Itineraries) - 1
  itineraryFares[i] = [][] (itineraryLengths[i])

  FOR l ← 0 TO LEN(r.Plan.Itineraries[i].Legs) - 1
    IF r.Plan.Itineraries[i].Legs[l].TransitLeg THEN
      agencyId ← TrimId(r.Plan.Itineraries[i].Legs[l].AgencyId)
      noc ← GetNoc(agencyId)

      legFares ← r.Plan>Itineraries[i].Legs[l].GetFares(noc)

      itineraryFares[i][l] ← legFares
    ENDIF
  ENDFOR
ENDFOR

RETURN itineraryFares
```

Firstly, an array is created which contains the number of legs in each itinerary in the `TripPlan`. This is then used to create the 3D array, `itineraryFares`.

The outer array is created with a length that matches the number of itineraries in the plan, and then two nested for loops are used to create the two inner arrays. The outer for loop creates an empty array for each itinerary, with the length from the same position in the `itineraryLengths` array. Then, the inner for loop iterates over each leg in that itinerary, and, if the leg in the plan has the Boolean property `TransitLeg` set to `true`, then the fares can be found for that leg. To do this, the `agencyId` is fetched from the TripPlan, and

its corresponding NOC is found. The reasoning and method behind this process is explained later. Now, the fares for that leg can be found, by calling the `GetFares` method on the leg (explained next), passing the NOC as a parameter. This method returns an array of `Fare` objects, which are then stored in the `itineraryFares` array at the current position, creating the third dimension of this array. Once completed, the finished `itineraryFares` array is returned.

Finding the fares for an itinerary leg

To find the fares for a given leg, the `GetFares` method is called on that leg. This method takes a connection to the database, and the NOC for the leg, represented by `c` and `n` in the following pseudocode, respectively. The leg itself is represented by `l`. The `FareObject.GetFare` method is also called, which is explained in the next section.

```
from ← TrimId(l.From.StopId)
to ← TrimId(l.To.StopId)
fareSlice ← []

query ← // This is detailed in the explanation below
rows ← c.Query(query)

FOR obj in rows
  fare ← obj.GetFare(from, to)
  IF fare ≠ null THEN
    APPEND(fareSlice, fare)
  ENDIF
ENDFOR

SORT fareSlice BY fareSlice.PreassignedFareProduct.Id

filteredFareSlices ← [LEN(fareSlice)]
prevID ← ""
currentID ← ""
j ← 0
FOR i ← 0 TO LEN(fareSlice) - 1
  IF i = 0 THEN
    prevID ← fareSlice[i].PreassignedFareProduct.Id
  ELSE
    prevID ← currentID
  ENDIF
  currentID ← fareSlice[i].PreassignedFareProduct.Id
  IF prevId ≠ currentID THEN
    j ← j + 1
  ENDIF
  APPEND(filteredFareSlices[j], fareSlice[i])
ENDFOR
```

```
IF LEN(filteredFareSlices) > 0 THEN
  filteredFareSlices ← filteredFareSlices[:j+1]
ENDIF

legFares ← []
FOR fSlice IN filteredFareSlices
  newestFare ← fSlice[0]
  FOR fare IN fSlice
    IF fare.ValidBetween.FromDate.Time >
newestFare.ValidBetween.FromDate.Time THEN
      newestFare ← fare
    ENDIF
  ENDFOR
  APPEND(legFares, newestFare)
ENDFOR

RETURN legFares
```

The query used in the pseudocode above is as follows:

```
select * from fares
where fare_object→'Lines' @> '[{"PublicCode": "l.RouteShortName",
"OperatorRef": {"Ref": "noc:n"}}]' and fare_object→'ScheduledStopPoints'
@> '[{"ScheduledStopPointRef": "atco:from"}, {"ScheduledStopPointRef":
"atco:to"}]';
```

This selects all the possible FareObjects that match the route name in the leg, the NOC, and also contain the scheduled stop point references for both the from and to stops. These are matched by the indexed JSON key/value pairs in the arrays, with the PostgreSQL jsonb `@>` operator, which checks whether the left JSON value (e.g., `fare_object.Lines`), contains the right JSON value (in this case, an extract of an array) at the top level (e.g., `[{"PublicCode": "464", "OperatorRef": {"Ref": "noc:ROST"}}]`). This is essentially checking whether the full arrays contain the smaller sections used in the query.

Then, for each row returned from the query, the `GetFare` method is called on the `FareObject`, and if a `Fare` is returned, it is appended to the array fareSlices.

Note: Whilst I refer to the use of arrays in this documentation, my final Go code will use slices (evident from variable names in the pseudocode), which are very similar to arrays, but are much more flexible, for example their length is variable: *"Slices wrap arrays to give a more general, powerful, and convenient interface to sequences of data. Except for items*

with explicit dimension such as transformation matrices, most array programming in Go is done with slices rather than simple arrays."[11]

The fares are then sorted by type (`PreassignedFareProduct.Id`), which allows them to be grouped into separate arrays by those types, for further filtering.

To do this, a new array, `filteredFareSlices`, is created, with an initial length matching that of `fareSlices` (as this is the maximum length needed if there was only one `Fare` of each fare type). A loop is then used which uses `i` for the current position in `fareSlices`, `j` for the current position in `filteredFareSlices`, and `prevId` and `currentId` to store the fare type that has just been iterated over. Initially, both the previous and current ID are set to that of the first `Fare`, and then the `Fare` is appended to the `j` position in `filteredFareSlices`. For the subsequent iterations, the `j` position is incremented when the `prevId` and `currentId` no longer match, meaning each type is separated. To clean up, any empty positions in `filteredFareSlices` are then discarded.

For example, data such as the following:

```
[type1, type1, type2, type2, type2, type3]
```

Would be split into 3 separate arrays:

```
[
  [type1, type1],
  [type2, type2, type2],
  [type3]
]
```

Now, the fares are filtered to keep only the newest (i.e., with the most recent `ValidBetween.FromDate`) `Fare` of each type. To do this, a loop over the outer array of `filteredFareSlices` is used. For each fare slice, the variable `newest` is set to the first `Fare` in the slice, and then the fares are iterated over, and any newer `Fare` replaces the one currently stored in `newest`. After each fare slice, the newest `Fare` is appended to the `legFares` array, which is the final array that is returned once the loop is complete.

[11] Go Documentation — Effective Go, https://go.dev/doc/effective_go#slices

Finding fares from a specific `FareObject`

The fares package will have an exported method that runs on a `FareObject`, ‘`GetFare`’, which takes IDs for the stops that the journey will go from and to, and returns a `Fare`.

In the following pseudocode for the method, `f` is the `FareObject` that the method will run on, and `from` and `to` are stop IDs.

```
fromRef ← "atco:" + from
toRef ← "atco:" + to
```

Firstly, the stop IDs are prefixed with ‘`atco:`’, as this is how they are represented in fare datasets.

```
FOR fareZone IN f.fareZones
  FOR member IN fareZone.members
    IF member.Ref = fromRef
      fromZone ← fareZone.Id
      BREAK
    ENDIF
    IF member.Ref = toRef
      toZone ← fareZone.Id
      BREAK
    ENDIF
  ENDFOR
ENDFOR
```

Now, the code loops over each fare zone in the dataset, and the members (stops) of each zone, until the ID of the zones that contain the ‘from’ and ‘to’ stop are found. This is necessary as the price is based on the fare zones that the stops fall in, not the stops themselves.

```
distanceMatrixElementIds ← []

FOR d IN f.Tariffs[0].distanceMatrixElements
  IF (d.StartTariffZoneRef.Ref = fromZone
      AND d.EndTariffZoneRef.Ref = toZone)
     OR (d.StartTariffZoneRef.Ref = toZone
      AND d.EndTariffZoneRef.Ref = fromZone)
    APPEND(distanceMatrixElementIds, d.Id)
  ENDIF
ENDFOR
```

Each combination of fare zones has a ‘distance matrix element’ which contains references to the fare zones in it, and an ID that can be looked up in the fare table to find the exact fare. The code loops over the distance matrix elements in the dataset, until one is found

which contains start and end references that match the from and to fare zones. In some datasets, the `StartTariffZoneRef` and `EndTariffZoneRef` are the wrong way round, so we need to check both ways, appending found IDs to an array in case there are multiple.

```
geographicalIntervalPriceRef ← ""
userProfileRef ← ""
salesOfferPackageRef ← ""
preassignedFareProductRef ← ""
tarrifRef ← ""
FOR d IN distanceMatrixElementIds
  FOR t IN f.FareTables
    FOR c IN t.Cells
      IF c.DistanceMatrixElementPrice.DistanceMatrixElementRef.Ref = d
        geographicalIntervalPriceRef ← c.DistanceMatrixElementPrice
                                        .GeographicalIntervalPriceRef.Ref
        userProfileRef ← t.UserProfileRef.Ref
        salesOfferPackageRef ← t.SalesOfferPackageRef.Ref
        preassignedFareProductRef ← t.PreassignedFareProductRef.Ref
        tarrifRef = t.TariffRef.Ref
        BREAK
      ENDIF
    ENDFOR
  ENDFOR
ENDFOR
```

Next, the algorithm searches for the distance matrix element ID in the fare table, iterating until a cell with a matching ID is found. Then, the references in the cell are stored, to be later used to find more information about the fare.

```
IF geographicalIntervalPriceRef = ""
  RETURN {}
ENDIF
```

If no matching distance matrix element was found in the fare table, the algorithm stops and returns an empty `Fare`.

```
fareAmount ← 0
FOR p IN f.PriceGroups
  IF p.GeographicalIntervalPrice[0].Id = geographicalIntervalPriceRef
    fareAmount ← p.GeographicalIntervalPrice[0].Price
    BREAK
  ENDIF
ENDFOR
```

Now, the algorithm iterates through data for the fare references that were stored, first searching the `PriceGroups` for the actual price of the fare and storing it.

```
userProfile ← {}
FOR u IN f.UserProfiles
  IF u.Id = userProfileRef
    userProfile ← u
    BREAK
  ENDIF
ENDFOR

salesOfferPackage ← {}
FOR s IN f.SalesOfferPackages
  IF s.Id = salesOfferPackageRef
    salesOfferPackage ← s
    BREAK
  ENDIF
ENDFOR

preassignedFareProduct ← {}
FOR p IN f.PreassignedFareProducts
  IF p.Id = preassignedFareProductRef
    preassignedFareProduct ← p
    BREAK
  ENDIF
ENDFOR

validBetween ← {}
FOR t IN f.Tariffs
  IF t.Id = tariffRef
    validBetween ← t.ValidBetween
    BREAK
  ENDIF
ENDFOR
```

The same procedure is carried out for the other references, searching in `UserProfiles` (e.g., what ages a fare is valid for, although most datasets set this to ‘anyone’, even for age-restricted fares), `SalesOfferPackages` (how a fare can be purchased, e.g., onboard), `PreassignedFareProducts` (the type of fare, e.g., ‘U19 Single’), and also in `Tariffs` to find the validity of the fare (e.g., what date this price is valid from or to). For each of these, an object is stored with the various properties of each.

```
fare ← {
      Amount: fareAmount,
      ValidBetween: validBetween,
      UserProfile: userProfile,
      SalesOfferPackage: salesOfferPackage,
      PreassignedFareProduct: preassignedFareProduct
}

RETURN fare
```

Now, with all the data required, a `Fare` object can be created and returned, with all the found properties stored in a single object to represent this fare.

Trimming OTP IDs

Used a few times in the pseudocode above is the utility function `TrimId`, which takes and returns a string, and is necessary due to the formatting of some IDs given by the OTP API. For example, all stop IDs (in the NaPTAN format), are prefixed by a number and a colon. This prefix is irrelevant for our use case, so needs to be removed. `TrimId` uses a regular expression, replacing all the times it matches in the given string with an empty string, and returning the result. The pattern used is `^[0-9]+:` which matches when the beginning of the string contains one or more digits, followed by a colon.

Managing the dataset

The dataset of `FareObjects` is managed using command line flags, which allow the latest data to be downloaded and added to the database. These flags can call two different functions, which are for internal use, and cannot be accessed via the API.

The first is `GetDatasets`, which takes two parameters, a directory to store the data in, and a comma separated list of NOCs (bus operators), and then downloads the data latest data from those operators to the specified folder.

The second is `AddDir`, which, as the name suggests, adds the specified directory of data to the database. As the datasets are downloaded into separate ZIP files, the function then calls an `AddZip` function for each file, which subsequently calls an `AddXml` function for each XML fare file inside the ZIP. Finally, `AddXml` calls `AddFareObject`, which adds the final decoded `FareObject` to the database.

Getting the latest datasets

In the following simplified pseudocode for `GetDatasets`, the directory is represented by `dir`, and the list of NOCS by `nocs`. The environment variables `BODS_API_BASE` and `BODS_API_KEY` are also used, as is an external library called `grab` which allows for the batch downloading of files.

```
create directory dir (in case it doesn't exist)
url ← BODS_API_BASE + "/fares/dataset/?noc=" + nocs +
"&status=published&api_key=" + BODS_API_KEY
resp ← http.Get(url)
apiResponse ← decode(resp.Body)

reqs ← []
FOR result in apiResponse.Results
  req ← grab.NewRequest(dir, result.Url)
  reqs ← append(reqs, req)
ENDFOR

client ← grab.NewClient()
client.DoBatch(4, reqs)
```

First, the directory is created if necessary. Then, the API request to the BODS API is created from the list of NOCs, and called as a HTTP GET request. The response is then decoded into the type `ApiResponse` and stored in `apiResponse`. This is then looped over, with a new `grab` `Request` created for each response. Finally, a new `grab` `Client` is initialised, which downloads up to 4 of the requests simultaneously.

Adding a directory of data

The `AddDir` function takes a connection to the database, `c`, which is passed through to the functions it calls, and a directory, `dir`, as a string. Whilst a for loop has been used for simplicity in this pseudocode, the underlying Go code uses the `filepath.WalkDir` function, which calls its given function for each file or directory it encounters.

```
FOR file in dir
  IF file.extension = "zip" THEN
    AddZip(c, file.path)
  ENDIF
  IF file.extension = "xml" THEN
    xml ← open (file)
    defer closing (xml)
    AddXml(c, xml)
  ENDIF
ENDFOR
```

If the file is a ZIP (of XML files), then `AddZip` is called for the file. If the file is an XML file directly (which can happen occasionally, especially for some smaller operators), then `AddXml` is called immediately, skipping the ZIP stage.

Adding a ZIP file

The `AddZip` function takes a connection to the database, `c`, which is passed through to `AddXml` when called, and a file path, `path`, as a string.

```
openZipFile ← open zip file (path)
defer closing (openZipFile)
FOR file in openZipFile
  IF file.extension = "xml" THEN
    openFile ← open (file)
    defer closing (openFile)
    AddXml(c, openFile)
  ENDIF
ENDFOR
```

The ZIP is opened, and its contents looped through. For each XML file in the ZIP, it is opened and passed through to the `AddXml` function. The ZIP and XML files are closed automatically once finished with by Go's `defer` keyword and `io.ReadCloser.Close` method.

Adding an XML file

The `AddXml` function takes a connection to the database, `c`, and a file, `file`, of type `io.ReadCloser` (i.e., something that can be read and then closed). It decodes the XML data in `file` into an object of type `FareObject`, `obj`. `c` and `obj` are then passed to the function `AddFareObject`. The XML file is then closed.

Adding a `FareObject` to the database

The `AddFareObject` function takes a connection to the database, `c`, and a `FareObject`, `f`. The simplified pseudocode is as follows:

```
fareObjectJson ← json.Marshal(f)
c.Exec(insert into fares (fare_object) values (fareObjectJson))
```

The `FareObject` is first marshalled (converted) into JSON, before it can be added to the database in the `fare_object` column of the `fares` table by executing the above query. The UUID in the `id` column is generated automatically, based on how the table was originally defined.

Converting between NOC and agency ID

The agency module of the backend has the purpose of converting the agency IDs used in OTP API responses to the NOCs used in FareObjects. This complication is required due to the way that the Bus Open Data Service creates GTFS files, where each operator is assigned an agency ID in that data, for example the NOC `ROST` has a corresponding agency ID `OP291`. The mappings between these values are stored in the `agency.txt` file present in each GTFS ZIP file.

It uses the custom data type `Agencies`, which is a `map` (sometimes called a dictionary in other languages) of key/value pairs where the `AgencyId` type is the key and the `Noc` type is the value. Both of these types are simple extensions of the built-in `string` type.

`Agencies` has 3 methods. `GetNoc` takes an `AgencyId` as a parameter, looks it up, and returns the corresponding `Noc`, if it was found. `Add` is used during the initialisation, takes an `AgencyId` and an `Noc`, and adds a new key/value pair to the map if it doesn't yet exist. `AddFromCsv` calls the `Add` method for each row in a given CSV file, e.g., `agency.txt`.

The simplified pseudocode for `Add` is as follows: (where `Agencies` is `a`, the `AgencyId` is `id`, and the `Noc` is `noc`)

```
a.GetNoc(id)
IF noc not found THEN
  a[id] ← noc
ENDIF
```

The simplified pseudocode for `AddFromCsv` is as follows: (where `Agencies` is `a` and the CSV file is `file`)

```
defer closing (file)
csvReader ← csv.NewReader(file)
csvReader.Read()
FOR record in csvReader
  a.Add(record[0], record[6])
ENDFOR
```

Where above, `csv.NewReader` uses Go's built-in CSV handling functionality, and `csvReader.Read()` is used to skip the first line of the file, which contains the column headers. The values for the `AgencyId` and `Noc` are stored at the first and seventh

positions on each line, respectively (so are in index positions 0 and 6). For example, one line of `agency.txt` could read as follows:

```
OP291,"Rosso","https://www.traveline.info",Europe/London,EN,"","ROST"
```

The `Agencies` are loaded during the program start-up, where the function `loadAgencies` is called, which takes the path to the directory of GTFS files, `gtfsDir`, as a `string`, and returns an `Agencies` map.

```
agencies ← new instance of Agencies
gtfsFiles ← []
FOR file IN gtfsDir
  IF file.extension = ".zip" THEN
    gtfsFiles ← append(gtfsFiles, file.path)
  ENDIF
ENDFOR

FOR gtfsFile IN gtfsFiles
  openGtfsFile ← open zip file (gtfsFile)
  defer closing (openGtfsFile)
  FOR file IN openGtfsFile
    IF file.name = "agency.txt" THEN
      openFile ← open (file)
      agencies.AddFromCsv(openFile)
    ENDIF
  ENDFOR
ENDFOR
RETURN agencies
```

First, a new instance of `Agencies` is created. Then, the path of each ZIP file (GTFS files are ZIP files) in the given directory is added to an array. Each file in the array is then opened, and if it contains `agency.txt`, the agency IDs and NOCs from that file are added to the map. Once all the files have been imported, the map is returned.

Getting the colour of a line

In the frontend, bus lines are coloured based on the public brand of that route. However, there is no single and easy-to-use public API to get these colour values, so they need to be recorded manually. These are stored in a CSV file `colours.csv`, which is loaded into the program when it starts, and contains three columns, the operator (`Noc`), line number or name, and a hex colour code for it. A line of `colours.csv` could read as follows:

```
ROST,464,#72177A
```

The logic for the `colours` module, which stores and allows access to the colour data, is almost identical to agencies, except for an extra dimension in the map. In this sense, the outer map is a map of Nocs, where each Noc corresponds to a map of Lines, where each Line corresponds to a colour.

The `Colours` are also loaded during the program start-up, but from a single CSV file, so the loading process is much simpler. The function `loadColours` takes the path to the CSV file, `coloursPath`, and returns an instance of `Colours`.

```
cols ← new instance of Colours
coloursFile ← open file (coloursPath)
defer closing (coloursFile)
cols.AddFromCsv(coloursFile)
return cols
```

Exposing the API

The `api` module provides a `Handler`, which (as the name suggests) handles HTTP requests that are directed to it. All requests to the `/api` endpoint of the program are handled by this module. The HTTP requests are forwarded to different functions based on patterns in the path.

Within the API, any requests made to the root (`/api`), are handled by the `homepage` function, which simply responds with “busfares api”, and nothing else.

Getting fares with the API

Any requests to the fares endpoint (`/api/getfares`) are handled by the `getFares` function, which is passed a connection to the database, `c`, and the `Agencies`, `a`. As a Handler function, it also has access to the HTTP request, `r`, and can write to the HTTP response, `w`. Simplified pseudocode for the function is as follows:

```
defer closing r.Body
tripPlan ← new instance of TripPlannerResponse
json.Decode r.Body into tripPlan
fares ← tripPlan.GetAllFares(c, a)
json.Encode fares into w
```

The request (an OTP plan) is decoded and stored in `tripPlan` as a `TripPlannerResponse`, before `GetAllFares` is run on it. The result of this call, `fares`, is then converted to JSON, and written as the HTTP response.

Getting colours with the API

Any requests to the colour endpoint (`/api/getcolour`) are handled by the `getColour` function, which is passed the `Colours`, `c`, and the `Agencies`, `a`. As above, it also has access to `r` and `w`. It can fetch the colour from either an agency Id or an NOC, based on the query parameters used in the request. Simplified pseudocode for the function is as follows:

```
defer closing r.Body
ag ← get query parameter "agency" from r.URL
IF ag ≠ "" THEN
  noc ← a.GetNoc(ag)
ELSE
  noc ← get query parameter "noc" from r.URL
ENDIF
colour ← c.Get(noc, get query parameter "line" from r.URL)
w ← colour
```

First, the `agency` from the request is stored, and if it isn't empty, its corresponding NOC is found and stored in `noc`. If it is empty, the NOC from the request is stored in `noc`. Now, the colour is found, with the value in `noc`, and the `line` from the request used as parameters. This result is written to `w` as the response.

Running the backend

The `main` function of the program, run when it launches, is responsible for setting up and initialising the required data and configuration.

The first part of this is configuring the application based on command line flags that have been set, and environment variables. There are three Boolean flags that can directly change the behaviour of the application.

`-dev` runs the program in development mode, allowing the frontend and backend to run separately from each other. This option is stored as `devMode`.

`-get-datasets` downloads the latest datasets from BODS into the configured directory when the program runs. This option is stored as `getDatasets`.

`-add-datasets` resets the database, and then imports downloaded datasets from the configured directory into it. This option is stored as `addDatasets`.

There are also 7 configuration options, these are set by environment variables by default, and can be overridden or set with command line flags. They are as follows:

Flag	*Environment variable*	*Stored as*	*Example*
`-datasets`	`DATASET_DIR`	`datasetDir`	`data/fares`
The directory to download datasets into and/or load them into the database from. Only needed if getting or adding datasets.			
`-db`	`DATABASE_URL`	`dbUrl`	`postgres://user:pass@server:5432/db-name`
The URL to use to connect to the database.			
`-gtfs`	`GTFS_DIR`	`gtfsDir`	`gtfs`
The directory containing the GTFS files.			
`-colours`	`COLOURS`	`coloursPath`	`data/colours.csv`
Path to CSV containing line colours			
`-nocs`	`NOCS`	`nocs`	`ROST,BPTR`
Comma separated list of NOCS to download. Only needed if getting or adding datasets.			
`-bods-key`	`BODS_API_KEY`	`bodsApiKey`	`0ffffffffffffffffffffffffffffffffffffff0`
API key for the Bus Open Data Service API. Only needed if getting or adding datasets.			
`-bods-base`	`BODS_API_BASE`	`bodsApiBase`	`https://data.bus-data.dft.gov.uk/api/v1`
Base URL to use for the Bus Open Data Service API. Only needed if getting or adding datasets.			

After parsing the configuration, the program then connects to the database specified in `dbUrl`, saving this connection as `conn`. Its closing is deferred, i.e., not until it is no longer needed.

Next, if `getDatasets` is true, `fares.GetDatasets` is called with `datasetDir` and `nocs` as parameters.

If `addDatasets` is true, `create-tables.sql` is opened, and then executed on the database connection. `fares.AddDir` is then called, with `conn` and `datasetDir` as parameters.

Now, `loadAgencies` is called with `gtfsDir` as its parameter, and the agencies are stored as `agencies`. `loadColours` is then called with `coloursPath` as its parameter, and the colours are stored as `cols`.

Finally, the HTTP routing needs to be configured. A new `ServeMux` is created (essentially a router), that directs traffic based on URL paths. Any traffic that matches the pattern "`/api/`" is routed to `api.Handler`, which is also passed `conn`, and access to `agencies` and `cols`. Traffic that matches the pattern "`/`" (i.e., all other traffic) is routed to `frontend.SvelteKitHandler`, part of the `frontend` Go module that will be explained in the next section.

The `ServeMux` is then stored as `handler`. If devMode is true, `handler` is wrapped in another `http.Handler`, the devCors function, which adds the "`Access-Control-Allow-Origin *`" header to any HTTP requests. This is needed in development mode as the frontend is running separately from the backend, so the API will receive requests from external origins. Only adding this header in development allows for increased security when running the app in production.

Now, to start the app, `http.ListenAndServe` is called, which takes the listen address, `:8080`, and the handler, `handler`, as parameters. This allows the app to receive incoming requests from the frontend, and serve the frontend files to client devices.

Creating the frontend

To allow the user to interact with the API, the frontend displays a web interface, which can get an appropriate response from the OTP API, send it to the fares backend, and display the responses alongside each other.

The frontend is built using SvelteKit, a "framework for rapidly developing robust, performant web applications using Svelte"[12]. Svelte is a web framework for writing

[12] SvelteKit Documentation — Introduction

interactive user interface components, essentially a way of using JavaScript (or TypeScript) to easily add data and reactivity to HTML.

Embedding the frontend in the backend

To allow the backend and frontend to be distributed easily as a single binary, the frontend is built and embedded within the backend as a Go package. SvelteKit apps use a concept called adapters, which take the built app as an input and generate output for a specific deployment target. My app is configured to use `@sveltejs/adapter-static`, which allows the app to be built as a static site, i.e., one that can function fully from just its 'flat' files. This allows the static files to be stored and hosted from anywhere, which is perfect for embedding them within the Go binary.

The frontend folder contains one Go file, `embed.go`, which declares a simple package called `frontend`. This package declares how the frontend is built and embedded, and creates a `Handler`, `SvelteKitHandler`, which is used by the `ServeMux` in `main` to serve the frontend alongside the backend. The file declares configuration options through the use of special comments and a variable:

```
//go:generate npm i
//go:generate npm run build
//go:embed all:build
var files embed.FS
```

The first two, the 'generate' comments, are run when `go generate` is called during the build sequence. These run `npm` (Node Package Manager) commands, to install the required dependencies and then build the frontend. The 'embed' comment is then configured to embed all the contents of the `build` folder (the output of the frontend build) at build time. Finally, the `files` variable is initialised with the special type `embed.FS`, which provides access to the embedded virtual filesystem.

The Handler `SvelteKitHandler` takes the request `path` as its single parameter. The subdirectory `build` of `files` is opened, and then converted to type `http.FileSystem`, so its contents can be served over the network.

A Handler function is now created and returned, which has access to the HTTP request, `r`, and can write to the HTTP response, `w`.

```
path ← trim string prefix(r.URL.Path, path)
filesystem.Open(path)
IF file doesn't exist THEN
  path ← path + ".html"
ENDIF
r.URL.Path ← path
http.FileServer(filesystem).ServeHTTP(w, r)
```

Here, the path of the file to serve is trimmed to remove the domain and other parts of the URL, and then the file is attempted to be opened. If the file isn't found, then `.html` is appended to the path, in case the static file is stored in this way. The `http.FileServer` function is then used to serve the actual file from the embedded filesystem.

Designing the frontend

This section outlines the design decisions that I made whilst building the frontend. The technical details of how these were implemented are explained later.

Initial layout

Initially, I built the frontend without any styling or much thought into layout; just the basic components needed to interact with and display data from the backend. The following is a screenshot from an initial build of the frontend:

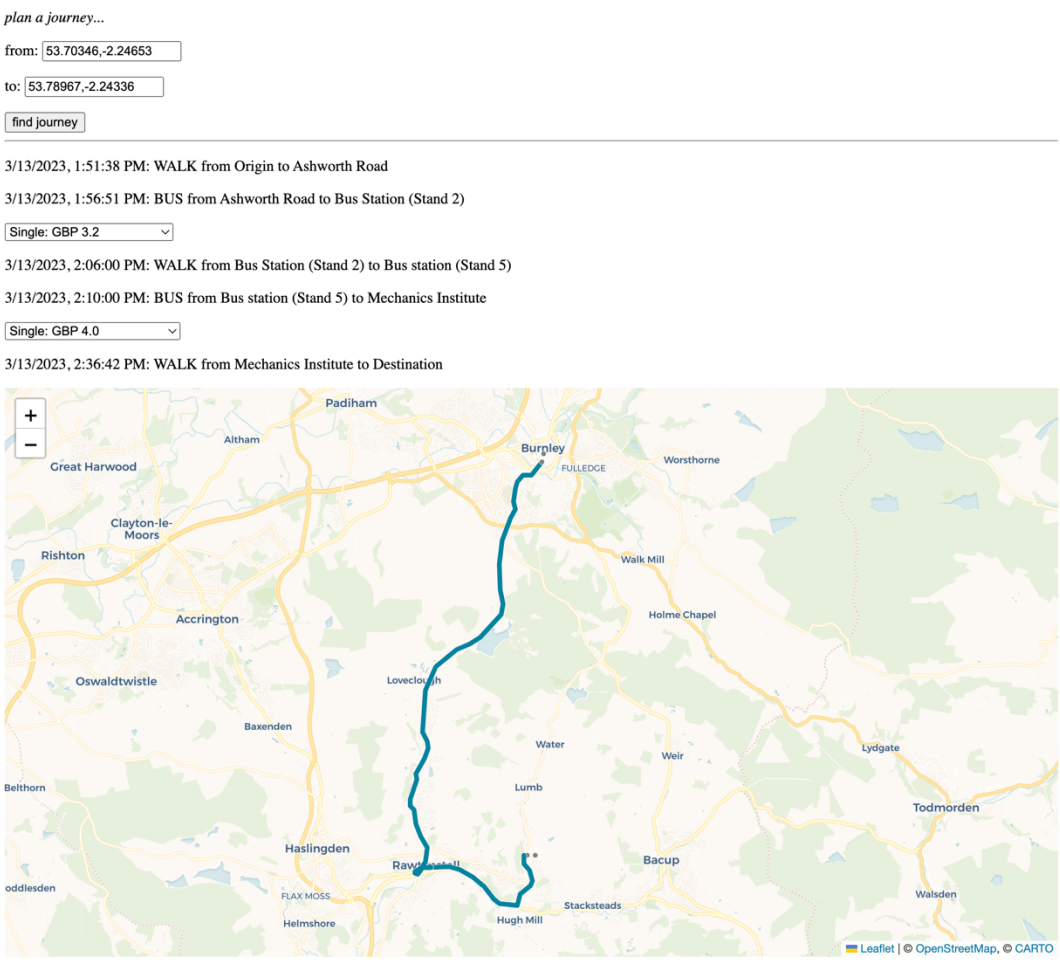


Figure 12. Initial build of frontend.

The user is able to enter a from and to location, and trigger a journey search. The resulting journey is displayed on a map, and a simple list is displayed of the steps to make the journey. For each transit leg, a fare selector is presented, which allows the user to choose from the available fare options for that journey. Of course, there are many important features missing here. Firstly, there is no way for the user to enter locations without knowing the coordinates of where they are going from and to. Secondly, only the first route from the OTP API is displayed, which can often be a single leg journey, to walk the entire route. This is obviously not very helpful, alongside only being able to plan a journey that leaves at the current time. Additionally, this layout does not adapt particularly well to different screen sizes, and is generally unintuitive to use. Behind the scenes, this version of the frontend was also quite inefficient, as the entire trip plan was sent back from the backend with the fares attached, instead of just the relevant fares, as outlined in the section *Finding the fares for an entire trip*.

Improving usability

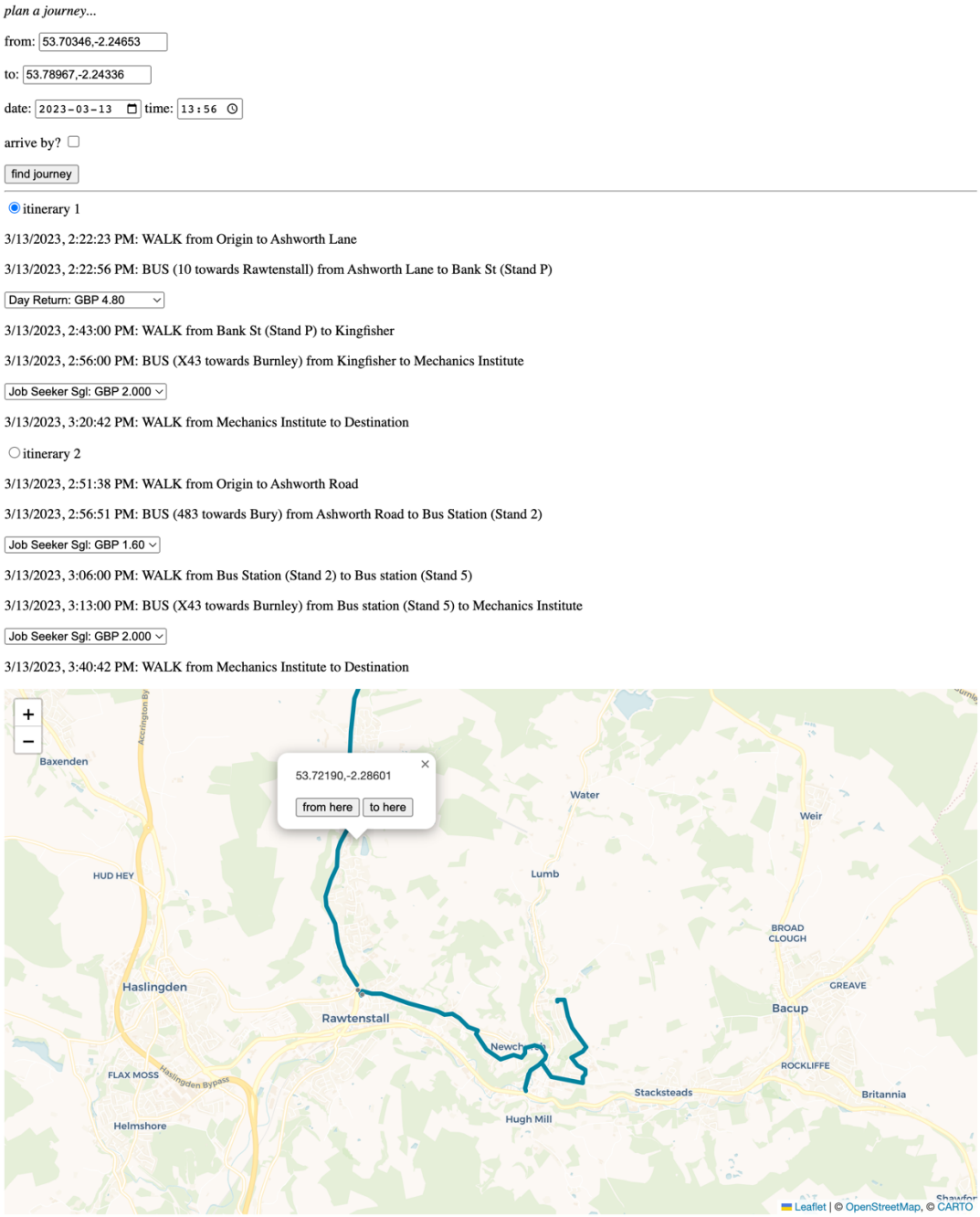


Figure 13. busfares frontend, 0.0.1-alpha.5

The next version of the frontend addressed some of these issues. Firstly, options to choose when to travel were added, with a date and time picker, and an option whether to plan a journey that departs or arrives at the time selected. Locations can now be entered by clicking on the map and choosing ‘from here’ or ‘to here’. Multiple itineraries can be displayed in a list, and selecting a different itinerary displays it on the map. Finally, the

frontend now receives an array that just contains fares, without having to re-receive the entire trip plan over the network.

Responsive design

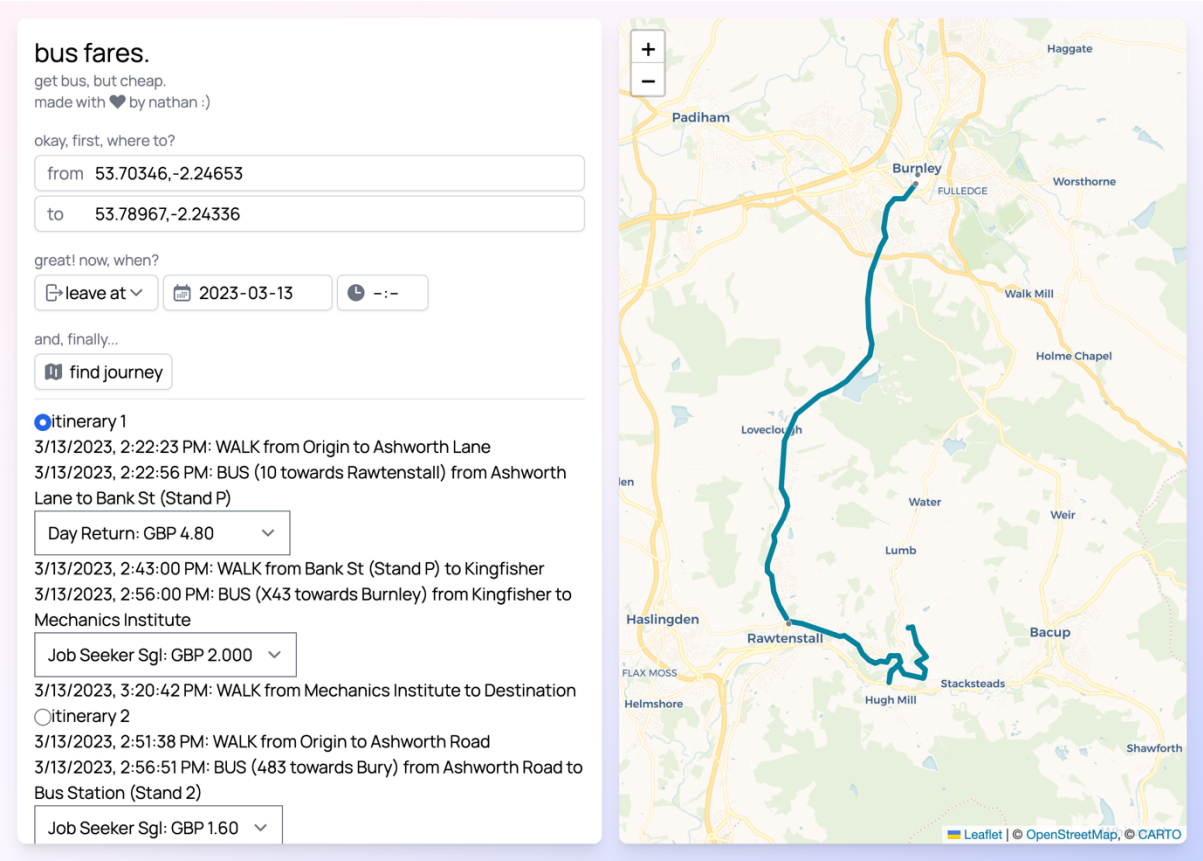


Figure 14. Frontend with Tailwind CSS.

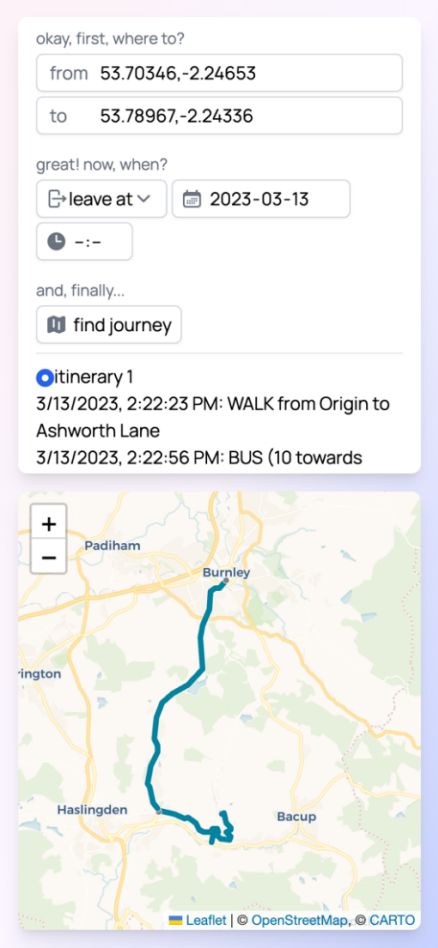


Figure 15. Responsive mobile frontend.

The next version introduced styling to the frontend, using Tailwind CSS, a highly customisable CSS framework, which makes the site responsive, adapting to different screen sizes. For example, the map and journey panels are arranged in a vertical orientation on screens with low widths, and horizontally across wider screens. Buttons are larger, and so their touch targets are big enough to easily hit on a mobile device. The search fields are arranged with labels to improve the user experience, as it is now clearer what to do to initiate a search.

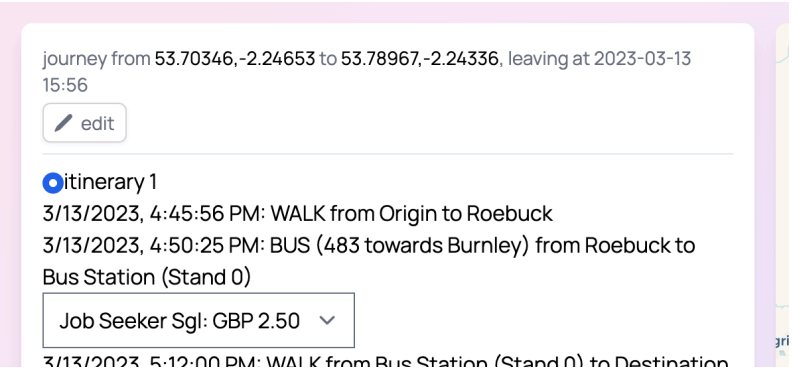


Figure 16. Trip summary at top of results.

Next, a new view was introduced for the search section once a search had already been made, where it shrinks down to just a summary of the search details. This makes it easier to browse the results, especially on devices with limited screen space.

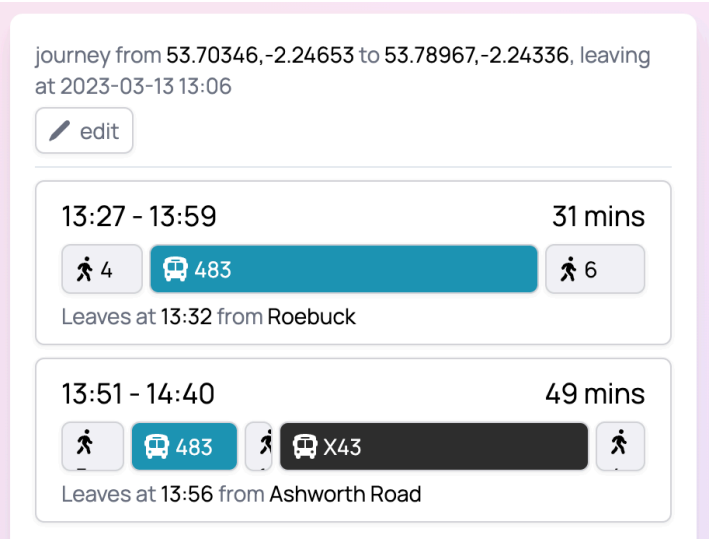


Figure 17. Summary cards in results.

The next iteration introduced trip summary cards, where each itinerary is presented in a small card, and selecting a card opens a details page with the full itinerary. The details of the first transit journey in each itinerary are presented below the summary. Additionally, this version integrated with the backend's colour API, allowing routes to be coloured in summaries and on the map.

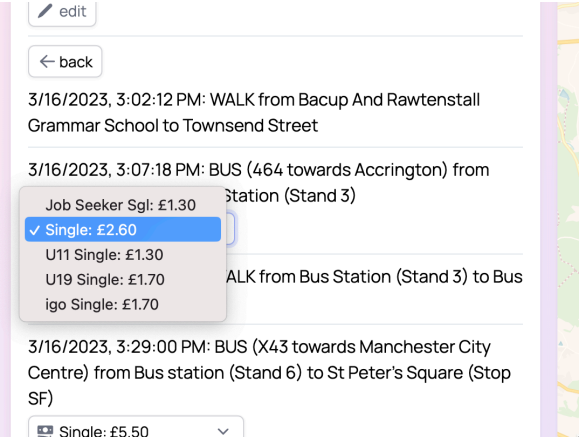


Figure 18. Dropdown in trip summary to select a fare.

When a user selects a card, the details page shows them each step of the journey, and for legs of the journey completed by bus, allows them to select from the available fares in a dropdown.

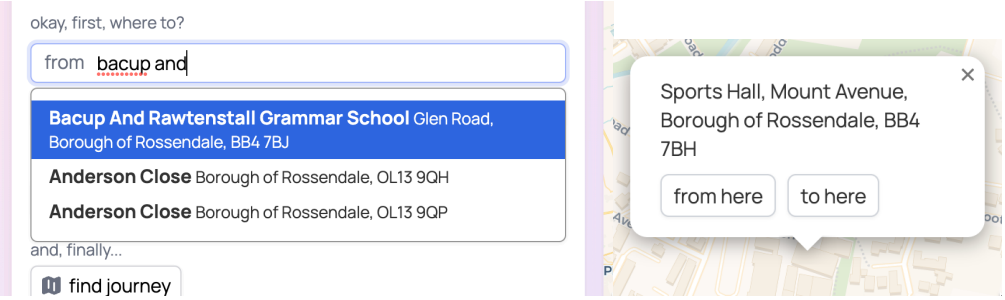


Figure 19. Search with autocomplete and reverse geocoding.

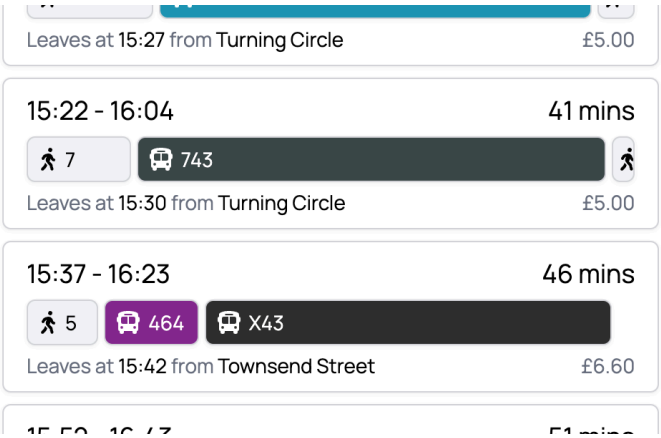


Figure 20. Total trip price in summary card.

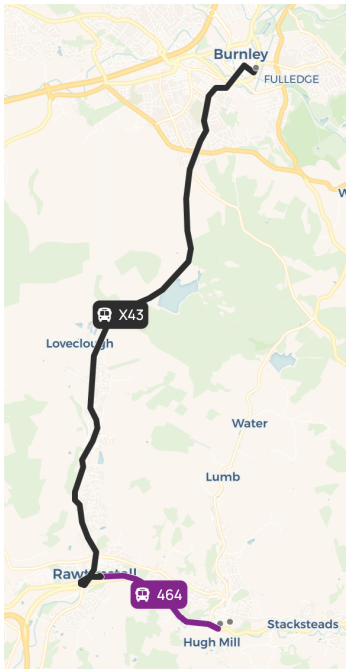


Figure 21. Transit legs labelled on map.

The final revision of the frontend includes several usability improvements. The biggest of these is the implementation of the Photon geocoding API, which allows for the from and to locations to be searched for easily, with human readable place names instead of just coordinates. This also works in reverse, i.e., when a point is selected on the map to set the location from there, the name can be requested and displayed. Additionally, the total price for each trip is displayed in the itinerary summary cards, to allow them to be compared more easily.

Managing state and data

The majority of the logic in the frontend involves managing the state of different components so that the correct information is displayed to the user at the right time, for example by switching to a different view, or adding data to the map. Data is passed around the frontend in variables, and Svelte automatically updates the page displayed to the user when their values change. Some data, such as the array storing the lines to be added to the map, uses the Svelte reactivity label, `$:`, which reruns the assignment or block of code it precedes when any of the values it references change. Because the variables for configuring the trip search are linked (bound) to the input fields, they update

automatically when the user interacts with the fields, and are then sent to the backend by the function `getPlan` and `getFares` when the find journey button is clicked.

`getPlan` requests a `TripPlan` from the `OpenTripPlanner` backend with the parameters specified by the user, and stores the JSON response in the variable tripPlan. The arrays selectedFares (holds the currently selected fare type for each journey leg) and `totalFares` (holds the total price for each journey) are cleared, and `tripPlanFares` is set by making a request to the fares backend with `getFares`. Then, for each journey in the plan, a placeholder is assigned in `selectedFares` and `totalFares` to be assigned later. For each leg in the journey, the fares are added from `tripPlanFares` to the `tripPlan` itself, and the default selected fare is stored by searching for the 'Trip@Single' preassigned fare product ID, which represents a standard adult single ticket. The function also iterates over the legs of the journeys, calling the `getLineColour` utility function for each if it is a bus leg. These calls are made asynchronously to each other and the fetching of fares to improve performance, especially when searching for multiple lines.

A reactive block is used to update the `totalFares` array when any of the `selectedFares` change, ensuring the totals displayed in the trip summary cards are correct.

Another block calls the `drawLine` function for the legs of the currently selected journey when it changes. `drawLine` takes an array of legs, and creates an array of line objects, using the decode method in the @googlemaps/polyline-codec library to convert the geometry of the line from the encoded 'polyline' given by OpenTripPlanner to an array of coordinates. These lines are stored in `mapLines`, which are added to and removed from the map programmatically using a Svelte `{#each}` block.

State variables are also used to decide what to display to the user. For example, the Boolean variables `editSearch` and `showItineraryDetail` decide whether the full search editor or just a summary of the search should be displayed, and whether a list of all itineraries or the detailed view of the currently selected itinerary should be shown. Svelte's built in transitions are used to transition smoothly between the states.

The map itself uses Leaflet, a popular JavaScript mapping library, and implements its standard functionality such as polylines and popups as Svelte components, allowing data to be changed in the map easily.

Frontend utilities

The file `utils.ts` provides a number of functions and constants which are used throughout the frontend.

Constants

The constant `otpBase` is pointed to the OpenTripPlanner instance, and is used in API requests.

Time management

The function `timestampToString` converts a UTC timestamp (i.e., milliseconds since 1st January 1970, such as those given by OpenTripPlanner) into a human readable string, with the time and then the date both formatted. This creates a JavaScript `Date` object from the timestamp, and uses the built-in `Date.toLocaleTimeString` and `Date.toLocaleDateString` functions for the conversion. The results of this are then returned as an object with `time` and `date` properties, so they can be accessed separately.

The function `secondsToDurationString` is used to convert the durations of journeys, which are provided in seconds by OpenTripPlanner, into human readable strings, such as “3 mins”, “< 1 min”, or “1 hr 27 mins”. It works by first creating a Date object from the duration in seconds provided by OpenTripPlanner. Then, if the duration is greater than an hour, the hours and minutes are returned, appending ‘s’ if needed for plurals. If there is less than one minute (e.g., a short walk between bus stops in a bus station), then “< 1 min” is returned. Finally, if neither of these conditions are met, the minutes are returned, formatted in a string, and pluralised as necessary.

Colour management

The function `getLineColour` takes an agency and a line, as strings, and returns a hex colour value. First, the function fetches the /api/getcolour endpoint from the backend,

with the agency and line as query parameters. The plain text response is then stored, and the function `lightenDarkenColour` is called with an amount value of 20. This lightens the colour, increasing the contrast between it and the text that will be placed on it, therefore increasing accessibility.

The `lightenDarkenColour` function takes a colour, `col`, as a hex string, and an amount, `amt`, to lighten or darken it by.

The pseudocode for this function is as follows:

```
useHash ← false
IF col[0] = "#" THEN
  col ← remove first char of col
  useHash ← true
ENDIF

num ← convert hex string to integer (col)

r ← (num >> 16) + amt

IF r > 255 THEN
  r ← 255
ELSE IF r < 0 THEN
  r ← 0
ENDIF

b ← ((num >> 8) & 0x00FF) + amt

IF b > 255 THEN
  b ← 255
ELSE IF b < 0 THEN
  b ← 0
ENDIF

g ← (num & 0x0000FF) + amt

IF g > 255 THEN
  g ← 255
ELSE IF g < 0 THEN
  g ← 0
ENDIF

IF useHash THEN
  prefix ← "#"
ELSE
  prefix ← ""
ENDIF

RETURN prefix + convert to hex string (g | (b << 8) | (r << 16))
```

Firstly, the hash character at the front of the string is removed if present, and will be re-added later. Next, the hexadecimal string is converted to an integer, so operations can be performed on it. The red, green, and blue components are lightened or darkened by the given amount. To split the components, the bitwise right shift operator, `>>`, is used, effectively extracting each colour by shifting it to the least significant byte. `&` is the bitwise `AND` operator, which is used to mask the part of the value that isn't part of the current component. Then, the amount specified is added to each component. If the adjusted components no longer fall within the allowed 0–255 range, they are adjusted to fit. Finally, the components are shifted left to their original positions with `<<`, and the bitwise `OR` operator, `|`, is used to combine the values into a single hexadecimal string. The hash is then prepended if it was present before, and the final string is returned.

The function `getContrastYIQ` uses a formula to determine whether text on a background of a given colour, `hexcol`, would be most readable if it was black or white. The formula converts colours into the YIQ colour space, which takes into account the sensitivity the human eye has to different colour spectrums, weighting the different parts of the colour according to their impact on our perception of the colour's brightness. Firstly, the hex colour is split into its three components; red, green, and blue, which are converted to integers between 0 and 255. These components are then converted to give a value in the YIQ colour space, where each is weighted differently: $((r * 299) + (g * 587) + (b * 114))/1000$. This value is then used to decide what colour the text should be. If it is greater than or equal to 128, then 'black' is returned. Otherwise, 'white' is.

Geocoding

The geocoding service I have used is Photon, an open-source project built on OpenStreetMap data. Its API represents places in JSON.

The function `reverseGeocode` takes a latitude and longitude as a parameter, and returns a place as a JSON object. A HTTP request to `https://photon.komoot.io/reverse` is made with the query parameters `lat` and `lon` both set. The reverse API only returns one place, so this is extracted from the results array and returned.

The function `placeToName` is used to create an easily-readable string to reference a place from the Photon API. It takes a place as a parameter, and joins together its properties based on which it has. If the place is a bus stop, only the name of the stop is returned. Otherwise, the place name, followed by the street, city, and postcode are joined together, if they are present in the data. The concatenated string is returned.

The function `searchPlaces` is used to provide an autocomplete function to the 'from' and 'to' fields. It takes a query as a required parameter, and optionally a latitude, longitude, and zoom. If these are set, they are used to weight the search results towards the map view that they represent, helping the user to find the right result more quickly. The function makes a HTTP request to `https://photon.komoot.io/api/` with the query parameters of the given query, a latitude and longitude bounding box for the UK, '-7.58, 49.96, 1.69, 58.64', to prevent international search results, and the map view parameters if they were supplied. If the length of the JSON results is not 0, then they are returned. If the length of the results is 0, then additional checks are made to see whether the user entered coordinates. Firstly, the query is tested against the following regular expression, `(-?\d{1,3}(?:\.\d+)?),\s*(-?\d{1,3}(?:\.\d+)?)`, which matches coordinates.

This expression has two capture groups, one for each coordinate. Each of these matches an optional negative, 1–3 digits, optionally followed by a decimal and one or more digits. These two capture groups must be joined by a comma, followed by any amount of whitespace.

If the query matches the regular expression, then `reverseGeocode` is called on the latitude and longitude given, and the result is returned as the search results array.

Technical solution

Index of files included in this section (naming reflects directory structure in project):

LICENSE 61
.env 61
.goreleaser.yaml 61
Dockerfile 63
create-tables.sql 63
go.mod 63
main.go 64
internal/agency/agency.go 67
internal/colours/colours.go 68
internal/fares/fares.go 70
internal/fares/dataset.go 76
internal/api/api.go 79
internal/router/router.go 80
frontend/embed.go 84
frontend/package.json 85
frontend/tsconfig.json 86
frontend/svelte.config.js 86
frontend/postcss.config.cjs 86
frontend/tailwind.config.cjs 86
frontend/vite.config.ts 87
frontend/src/app.d.ts 87
frontend/src/app.html 87
frontend/src/app.css 88
frontend/src/routes/+layout.ts 88
frontend/src/routes/+layout.svelte 88
frontend/src/routes/+page.svelte 88
frontend/src/lib/Tooltip.svelte 101
frontend/src/lib/Leaflet.svelte 102
frontend/src/lib/Polyline.svelte 105
frontend/src/lib/Button.svelte 106
frontend/src/lib/Popup.svelte 106
frontend/src/lib/ItinerarySummary.svelte 107
frontend/src/lib/ItineraryLegs.svelte 108
frontend/src/lib/ItineraryLeg.svelte 109
frontend/src/lib/utils.ts 110

LICENSE


```
MIT License

Copyright (c) 2023 Nathan Green

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```


.env

These are the environment variables used for development and testing. They may be configured differently for a deployment in production.

```
BODS_API_KEY=[REDACTED]
BODS_API_BASE=https://data.bus-data.dft.gov.uk/api/v1
GTFS_DIR=gtfs
COLOURS=data/colours.csv
DATABASE_URL=postgres://nathan@localhost:5432/postgres
NOCS=BPTR,HRGT,KDTR,LNUD,ROST,TPEN,YACT,YCST
```

.goreleaser.yaml

This is the configuration for GoReleaser, which is used by GitHub Actions (a continuous integration platform) to build the binaries and Docker images for each release of the program.

```
before:
  hooks:
    - go mod tidy
    - go generate ./...
builds:
  - env:
      - CGO_ENABLED=0
    goos:
      - linux
      - windows
```

```
      - darwin
dockers:
  - image_templates:
      - "ghcr.io/imnatgreen/busfares:latest-amd64"
      - "ghcr.io/imnatgreen/busfares:{{ .Tag }}-amd64"
      - "ghcr.io/imnatgreen/busfares:{{ .Major }}-amd64"
    use: buildx
    goos: linux
    goarch: amd64
    build_flag_templates:
      - "--platform=linux/amd64"
      - "--label=org.opencontainers.image.created={{ .Date }}"
      - "--label=org.opencontainers.image.title={{ .ProjectName }}"
      - "--label=org.opencontainers.image.revision={{ .FullCommit }}"
      - "--label=org.opencontainers.image.version={{ .Version }}"
    extra_files:
      - "create-tables.sql"
  - image_templates:
      - "ghcr.io/imnatgreen/busfares:latest-arm64v8"
      - "ghcr.io/imnatgreen/busfares:{{ .Tag }}-arm64v8"
      - "ghcr.io/imnatgreen/busfares:{{ .Major }}-arm64v8"
    use: buildx
    goos: linux
    goarch: arm64
    build_flag_templates:
      - "--platform=linux/arm64/v8"
      - "--label=org.opencontainers.image.created={{ .Date }}"
      - "--label=org.opencontainers.image.title={{ .ProjectName }}"
      - "--label=org.opencontainers.image.revision={{ .FullCommit }}"
      - "--label=org.opencontainers.image.version={{ .Version }}"
    extra_files:
      - "create-tables.sql"
docker_manifests:
  - name_template: "ghcr.io/imnatgreen/busfares:latest"
    image_templates:
      - "ghcr.io/imnatgreen/busfares:latest-amd64"
      - "ghcr.io/imnatgreen/busfares:latest-arm64v8"
  - name_template: "ghcr.io/imnatgreen/busfares:{{ .Major }}"
    image_templates:
      - "ghcr.io/imnatgreen/busfares:{{ .Major }}-amd64"
      - "ghcr.io/imnatgreen/busfares:{{ .Major }}-arm64v8"
  - name_template: "ghcr.io/imnatgreen/busfares:{{ .Tag }}"
    image_templates:
      - "ghcr.io/imnatgreen/busfares:{{ .Tag }}-amd64"
      - "ghcr.io/imnatgreen/busfares:{{ .Tag }}-arm64v8"
release:
  github:
checksum:
  name_template: 'checksums.txt'
snapshot:
  name_template: "{{ incpatch .Version }}-next"
changelog:
  sort: asc
  filters:
    exclude:
```

```
      - '^docs:'
      - '^test:'
```

Dockerfile

```
FROM scratch
EXPOSE 8080
ENTRYPOINT [ "/busfares" ]
COPY create-tables.sql /create-tables.sql
COPY busfares /
```

create-tables.sql

```
create extension if not exists "uuid-ossp";
drop table if exists fares;
create table fares (
  id uuid default uuid_generate_v4() primary key,
  fare_object jsonb not null
);
drop index if exists scheduled_stop_points_index;
drop index if exists lines_index;
create index if not exists lines_index on fares using gin ((fare_object-
>'Lines') jsonb_path_ops);
create index if not exists scheduled_stop_points_index on fares using gin
(
  (fare_object->'ScheduledStopPoints') jsonb_path_ops
);
```

go.mod

This file declares the project as a Go module, and details the required dependencies used.

```
module github.com/imnatgreen/busfares

go 1.19

require (
      github.com/bojanz/currency v1.0.4
      github.com/cavaliergopher/grab/v3 v3.0.1
      github.com/jackc/pgx/v5 v5.0.2
)

require (
      github.com/jackc/pgpassfile v1.0.0 // indirect
      github.com/jackc/pgservicefile v0.0.0-20200714003250-2b9c44734f2b //
indirect
      golang.org/x/crypto v0.0.0-20221010152910-d6f0a8c073c2 // indirect
      golang.org/x/text v0.3.8 // indirect
)

require (
      github.com/cockroachdb/apd/v3 v3.1.0 // indirect
      github.com/pkg/errors v0.9.1 // indirect
)
```

main.go

The main package imports all the other packages in the project and links them together, loading the configuration, connecting to the database, and starting the API.

```
package main

import (
  "archive/zip"
  "flag"
  "net/http"
  "time"

  "context"

  "io/fs"
  "log"
  "os"
  "path/filepath"

  "github.com/imnatgreen/busfares/frontend"
  "github.com/imnatgreen/busfares/internal/agency"
  "github.com/imnatgreen/busfares/internal/api"
  "github.com/imnatgreen/busfares/internal/colours"
  "github.com/imnatgreen/busfares/internal/fares"
  "github.com/jackc/pgx/v5"
)

func main() {
  var err error

  // load configuration from environment variables or command line flags
  var devMode bool
  var getDatasets bool
  var addDatasets bool
  var datasetDir string
  var dbUrl string
  var gtfsDir string
  var coloursPath string
  var nocs string
  var bodsApiKey string
  var bodsApiBase string
  flag.BoolVar(&devMode, "dev", false, "run in dev mode")
  flag.BoolVar(&getDatasets, "get-datasets", false, "download datasets
from BODS into dir")
  flag.BoolVar(&addDatasets, "add-datasets", false, "reset db, and then
add datasets from dir to database")
  flag.StringVar(&datasetDir, "datasets", os.Getenv("DATASET_DIR"),
"directory containing gtfs files")
  flag.StringVar(&dbUrl, "db", os.Getenv("DATABASE_URL"), "database url")
  flag.StringVar(&gtfsDir, "gtfs", os.Getenv("GTFS_DIR"), "directory
containing gtfs files")
  flag.StringVar(&coloursPath, "colours", os.Getenv("COLOURS"), "csv file
containing line colours")
```

```
  flag.StringVar(&nocs, "nocs", os.Getenv("NOCS"), "comma separated list
of NOCS")
  flag.StringVar(&bodsApiKey, "bods-key", os.Getenv("BODS_API_KEY"), "BODS
api key")
  flag.StringVar(&bodsApiBase, "bods-base", os.Getenv("BODS_API_BASE"),
"BODS api base url")
  flag.Parse()

  // connects to the database
  conn, err := pgx.Connect(context.Background(), dbUrl)
  if err ≠ nil {
    log.Fatalf("unable to connect to database: %v", err)
  }
  defer conn.Close(context.Background())

  // download datasets if configuration requests it
  if getDatasets {
    err = fares.GetDatasets(datasetDir, nocs)
    if err ≠ nil {
      log.Fatalf("failed to get datasets: %v", err)
    }
  }

  // imports datasets to the database if configuration requests it
  if addDatasets {
    sql, err := os.ReadFile("create-tables.sql")
    if err ≠ nil {
      log.Fatalf("failed to read create-tables.sql: %v", err)
    }
    _, err = conn.Exec(context.Background(), string(sql))
    if err ≠ nil {
      log.Fatalf("failed to create tables: %v", err)
    }
    err = fares.AddDir(conn, datasetDir)
    if err ≠ nil {
      log.Fatalf("failed to add datasets to db: %v", err)
    }
  }

  // load agencies from GTFS files
  start := time.Now()
  agencies, _ := loadAgencies(gtfsDir)
  log.Printf("loaded %d agencies from disk in %s", len(agencies),
time.Since(start))

  // load line colours from configured CSV
  cols, _ := loadColours(coloursPath)

  // initiate the HTTP server for the API
  mux := http.NewServeMux()

  mux.Handle("/api/", api.Handler("/api", conn, &agencies, &cols))

  mux.Handle("/", frontend.SvelteKitHandler("/"))
```

```
  var handler http.Handler = mux

  if devMode {
    handler = devCors(handler)
    log.Println("server running in dev mode")
  }

  // serve the API, and log and exit on any unhandled errors
  log.Fatal(http.ListenAndServe(":8080", handler))
}

// creates an Agencies object from the agencies in the configured GTFS
directory
func loadAgencies(gtfsDir string) (agencies agency.Agencies, err error) {
  agencies = make(agency.Agencies)

  var gtfsFiles []string
  err = filepath.WalkDir(gtfsDir, func(path string, d fs.DirEntry, err
error) error {
    if err ≠ nil {
      return err
    }
    if d.IsDir() {
      return nil
    }
    if filepath.Ext(d.Name()) == ".zip" {
      gtfsFiles = append(gtfsFiles, path)
    }
    return nil
  })
  if err ≠ nil {
    return agencies, err
  }

  for _, gtfsFile := range gtfsFiles {
    log.Printf("importing from %s", gtfsFile)
    openGtfsFile, err := zip.OpenReader(gtfsFile)
    if err ≠ nil {
      log.Fatalf("failed to open gtfs file %s", gtfsFile)
      return agencies, err
    }
    defer openGtfsFile.Close()
    for _, file := range openGtfsFile.File {
      if file.Name == "agency.txt" {
        openFile, err := file.Open()
        if err ≠ nil {
          log.Fatalf("failed to open file %s in %s", file.Name, gtfsFile)
          return agencies, err
        }
        agencies.AddFromCsv(openFile)
      }
    }
  }
  return agencies, nil
}
```

```
// creates a Colours object from the configured CSV path.
func loadColours(coloursPath string) (cols colours.Colours, err error) {
  cols = make(colours.Colours)
  coloursFile, err := os.Open(coloursPath)
  if err ≠ nil {
    return cols, err
  }
  defer coloursFile.Close()
  if err ≠ nil {
    return cols, err
  }
  err = cols.AddFromCsv(coloursFile)
  if err ≠ nil {
    return cols, err
  }
  return cols, err
}

// set the CORS headers required for API interaction during development
func devCors(handler http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    w.Header().Add("Access-Control-Allow-Origin", "*")
    handler.ServeHTTP(w, r)
  })
}
```

internal/agency/agency.go

```
package agency

import (
  "encoding/csv"
  "io"
)

const (
  ErrNocNotFound = AgencyErr("could not find NOC for given agency_id")
  ErrNocExists   = AgencyErr("NOC already exists, skipping")
)

type AgencyErr string

func (e AgencyErr) Error() string {
  return string(e)
}

type AgencyId string
type Noc string

type Agencies map[AgencyId]Noc

func (a Agencies) GetNoc(id AgencyId) (Noc, error) {
  definition, ok := a[id]
  if !ok {
```

```
    return "", ErrNocNotFound
  }
  return definition, nil
}

func (a Agencies) Add(id AgencyId, noc Noc) error {
  _, err := a.GetNoc(id)

  switch err {
  case ErrNocNotFound:
    a[id] = noc
  case nil:
    return ErrNocExists
  default:
    return err
  }

  return nil
}

func (a Agencies) AddFromCsv(file io.ReadCloser) error {
  defer file.Close()

  csvReader := csv.NewReader(file)
  csvReader.Read() // skip first line of column headers
  for {
    record, err := csvReader.Read()

    if err == io.EOF {
      break
    }
    if err ≠ nil {
      return err
    }

    a.Add(AgencyId(record[0]), Noc(record[6]))
  }
  return nil
}
```

internal/colours/colours.go

```
package colours

import (
  "encoding/csv"
  "io"
)

const (
  ErrNocNotFound  = ColoursErr("could not find NOC")
  ErrLineNotFound = ColoursErr("could not find line")
  ErrNocExists    = ColoursErr("NOC already exists, skipping")
  ErrLineExists   = ColoursErr("line already exists, skipping")
)
```

```
type ColoursErr string

func (e ColoursErr) Error() string {
  return string(e)
}

type Noc string
type Line string
type Colour string

type Colours map[Noc]map[Line]Colour

func (c Colours) Get(n Noc, l Line) (Colour, error) {
  _, ok := c[n]
  if !ok {
    return "", ErrNocNotFound
  }
  col, ok := c[n][l]
  if !ok {
    return "", ErrLineNotFound
  }
  return col, nil
}

func (c Colours) Add(n Noc, l Line, col Colour) error {
  _, err := c.Get(n, l)

  switch err {
  case ErrNocNotFound:
    c[n] = map[Line]Colour{}
    c[n][l] = col
  case ErrLineNotFound:
    c[n][l] = col
  case nil:
    return ErrNocExists
  default:
    return err
  }

  return nil
}

func (c Colours) AddFromCsv(file io.ReadCloser) error {
  defer file.Close()

  csvReader := csv.NewReader(file)
  csvReader.Read() // skip first line of column headers
  for {
    record, err := csvReader.Read()

    if err == io.EOF {
      break
    }
    if err ≠ nil {
```

```
      return err
    }

    c.Add(Noc(record[0]), Line(record[1]), Colour(record[2]))
  }
  return nil
}
```

internal/fares/fares.go

```
package fares

import (
  "encoding/xml"
  "errors"
  "time"

  "github.com/bojanz/currency"
  "github.com/imnatgreen/busfares/internal/agency"
)

type Noc = agency.Noc
type Naptan string

// XmlTime allows timestamps in FareXChange XML to be unmarshalled into
time.Time objects.
type XmlTime struct {
  time.Time `json:"time"`
}

func (x *XmlTime) UnmarshalXML(d *xml.Decoder, start xml.StartElement)
error {
  var v string
  error := d.DecodeElement(&v, &start)
  if error ≠ nil {
    return error
  }
  parse, error := time.Parse("2006-01-02T15:04:05", v)
  *x = XmlTime{parse}
  return error
}

type FareObject struct {
  XMLName             xml.Name                 `xml:"PublicationDelivery"`
  References          FareObjectReferences
`xml:"PublicationRequest>topics>NetworkFrameTopic>NetworkFilterByValue>obj
ectReferences"`
  Currency            string
`xml:"dataObjects>CompositeFrame>FrameDefaults>DefaultCurrency"`
  Operators           []Operator
`xml:"dataObjects>CompositeFrame>frames>ResourceFrame>organisations>Operat
or"`
  Lines               []Line
`xml:"dataObjects>CompositeFrame>frames>ServiceFrame>lines>Line"`
```

```
  ScheduledStopPoints []ScheduledStopPoint
`xml:"dataObjects>CompositeFrame>frames>ServiceFrame>scheduledStopPoints>S
cheduledStopPoint"`
  FareZones           []FareZone
`xml:"dataObjects>CompositeFrame>frames>FareFrame>fareZones>FareZone"`
  Tariffs             []Tariff
`xml:"dataObjects>CompositeFrame>frames>FareFrame>tariffs>Tariff"`
  UserProfiles        []UserProfile
`xml:"dataObjects>CompositeFrame>frames>FareFrame>usageParameters>UserProf
ile"`
  FareProducts        []PreassignedFareProduct
`xml:"dataObjects>CompositeFrame>frames>FareFrame>fareProducts>Preassigned
FareProduct"`
  SalesOfferPackages  []SalesOfferPackage
`xml:"dataObjects>CompositeFrame>frames>FareFrame>salesOfferPackages>Sales
OfferPackage"`
  PriceGroups         []PriceGroup
`xml:"dataObjects>CompositeFrame>frames>FareFrame>priceGroups>PriceGroup"`
  FareTables          []FareTable
`xml:"dataObjects>CompositeFrame>frames>FareFrame>fareTables>FareTable"`
}

type Operator struct {
  Id   string `xml:"id,attr"`
  Name string `xml:"Name"`
}
type FareObjectReferences struct {
  Operator AttrRef `xml:"OperatorRef"`
  Line     AttrRef `xml:"LineRef"`
}

type AttrRef struct {
  Ref string `xml:"ref,attr"`
}

type Tariff struct {
  Id                    string                 `xml:"id,attr"`
  Name                  string                 `xml:"Name"`
  OperatorRef           AttrRef                `xml:"OperatorRef"`
  LineRef               AttrRef                `xml:"LineRef"`
  TariffBasis           string                 `xml:"TariffBasis"`
  DistanceMatrixElements []DistanceMatrixElement
`xml:"fareStructureElements>FareStructureElement>distanceMatrixElements>Di
stanceMatrixElement"`
  TripType              string
`xml:"fareStructureElements>FareStructureElement>GenericParameterAssignmen
t>limitations>RoundTrip>TripType"`
  ValidBetween          ValidBetween
`xml:"validityConditions>ValidBetween"`
}

type DistanceMatrixElement struct {
  Id                string  `xml:"id,attr"`
  StartTariffZoneRef AttrRef `xml:"StartTariffZoneRef"`
  EndTariffZoneRef   AttrRef `xml:"EndTariffZoneRef"`
```

```
}

type ValidBetween struct {
  FromDate XmlTime `xml:"FromDate" json:"fromDate"`
  ToDate   XmlTime `xml:"ToDate" json:"toDate"`
}

type UserProfile struct {
  Id         string `xml:"id,attr" json:"id"`
  Name       string `xml:"Name" json:"name"`
  UserType   string `xml:"UserType" json:"userType"`
  MinimumAge string `xml:"MinimumAge" json:"minimumAge"`
  MaximumAge string `xml:"MaximumAge" json:"maximumAge"`
}

type PreassignedFareProduct struct {
  Id             string `xml:"id,attr" json:"id"`
  Name           string `xml:"Name" json:"name"`
  ChargingMoment string `xml:"ChargingMomentType" json:"chargingMoment"`
  TariffBasis    string `xml:"ConditionSummary>TariffBasis"
json:"tariffBasis"`
  ProductType    string `xml:"ProductType" json:"productType"`
}

type SalesOfferPackage struct {
  Id          string `xml:"id,attr" json:"id"`
  Name        string `xml:"Name" json:"name"`
  Description string `xml:"Description" json:"description"`
}

type PriceGroup struct {
  Id                        string                      `xml:"id,attr"`
  GeographicalIntervalPrice []GeographicalIntervalPrice
`xml:"members>GeographicalIntervalPrice"`
}

type GeographicalIntervalPrice struct {
  Id     string `xml:"id,attr"`
  Amount string `xml:"Amount"`
}

type FareTable struct {
  TariffRef                 AttrRef              `xml:"usedIn>TariffRef"`
  UserProfileRef            AttrRef
`xml:"pricesFor>UserProfileRef"`
  SalesOfferPackageRef      AttrRef
`xml:"pricesFor>SalesOfferPackageRef"`
  PreassignedFareProductRef AttrRef
`xml:"pricesFor>PreassignedFareProductRef"`
  Columns                   []FareTableRowColumn
`xml:"columns>FareTableColumn"`
  Rows                      []FareTableRowColumn `xml:"rows>FareTableRow"`
  Cells                     []FareTableCell
`xml:"includes>FareTable>cells>Cell"`
}
```

```
type FareTableRowColumn struct {
  Id           string    `xml:"id,attr"`
  Order        string    `xml:"order,attr"`
  Name         string    `xml:"Name"`
  FareZoneRefs []AttrRef `xml:"representing>FareZoneRef"` // only in
columns
}

type FareTableCell struct {
  Id                         string                    `xml:"id,attr"`
  DistanceMatrixElementPrice DistanceMatrixElementPrice
`xml:"DistanceMatrixElementPrice"`
  ColumnRef                  AttrRef                  `xml:"ColumnRef"`
  RowRef                     AttrRef                  `xml:"RowRef"`
}

type DistanceMatrixElementPrice struct {
  Id                           string  `xml:"id,attr"`
  GeographicalIntervalPriceRef AttrRef
`xml:"GeographicalIntervalPriceRef"`
  DistanceMatrixElementRef     AttrRef `xml:"DistanceMatrixElementRef"`
}

type Line struct {
  LineRef     string  `xml:"id,attr"`
  PublicCode  string  `xml:"PublicCode"`
  PrivateCode string  `xml:"PrivateCode"`
  OperatorRef AttrRef `xml:"OperatorRef"`
}

type ScheduledStopPoint struct {
  ScheduledStopPointRef string               `xml:"id,attr"`
  Name                  string               `xml:"Name"`
  TopographicPlaceView  TopographicPlaceView `xml:"TopographicPlaceView"`
}

type TopographicPlaceView struct {
  TopographicPlaceViewRef AttrRef `xml:"TopographicPlaceRef"`
  Name                    string  `xml:"Name"`
}

type FareZone struct {
  Id      string                  `xml:"id,attr"`
  Name    string                  `xml:"Name"`
  Members []ScheduledStopPointRef `xml:"members>ScheduledStopPointRef"`
}

type ScheduledStopPointRef struct {
  Ref  string `xml:"ref,attr"`
  Name string `xml:",chardata"`
}

// Type Fare is used when finding fares from an imported FareObject
type Fare struct {
```

```
  Amount                 currency.Amount        `json:"amount"`
  ValidBetween           ValidBetween           `json:"validBetween"`
  UserProfile            UserProfile            `json:"userProfile"`
  SalesOfferPackage      SalesOfferPackage      `json:"salesOfferPackage"`
  PreassignedFareProduct PreassignedFareProduct
`json:"preassignedFareProduct"`
}

func ParseXml(data []byte) (FareObject, error) {
  var obj FareObject
  err := xml.Unmarshal(data, &obj)
  return obj, err
}

var ErrFareNotInTable = errors.New("fare not in table")

func (f *FareObject) GetFare(from, to Naptan) (fare Fare, err error) {
  // get fare zones from stops
  fromRef := string("atco:" + from)
  toRef := string("atco:" + to)
  var fromZone, toZone string
  for _, fareZone := range f.FareZones {
    for _, member := range fareZone.Members {
      if member.Ref == fromRef {
        fromZone = fareZone.Id
        break
      }
      if member.Ref == toRef {
        toZone = fareZone.Id
        break
      }
    }
  }

  // get distance matrix element id from fare zones
  // for some reason, some fare files have the ids the wrong way round, so
we need to check with fare zones reversed too
  var distanceMatrixElementIds []string

  for _, d := range f.Tariffs[0].DistanceMatrixElements {
    if (d.StartTariffZoneRef.Ref == fromZone && d.EndTariffZoneRef.Ref ==
toZone) ||
      (d.StartTariffZoneRef.Ref == toZone && d.EndTariffZoneRef.Ref ==
fromZone) {
      distanceMatrixElementIds = append(distanceMatrixElementIds, d.Id)
      break
    }
  }

  // get price group id from distance matrix element id
  // also get fare related refs if price group found
  var geographicalIntervalPriceRef string
  var userProfileRef string
  var salesOfferPackageRef string
  var preassignedFareProductRef string
```

```
  var tariffRef string
  for _, d := range distanceMatrixElementIds {
    for _, t := range f.FareTables {
      for _, c := range t.Cells {
        if c.DistanceMatrixElementPrice.DistanceMatrixElementRef.Ref == d
{
          geographicalIntervalPriceRef =
c.DistanceMatrixElementPrice.GeographicalIntervalPriceRef.Ref
          userProfileRef = t.UserProfileRef.Ref
          salesOfferPackageRef = t.SalesOfferPackageRef.Ref
          preassignedFareProductRef = t.PreassignedFareProductRef.Ref
          tariffRef = t.TariffRef.Ref
          break
        }
      }
    }
  }
  // check if fare found, if not return empty fare
  if geographicalIntervalPriceRef == "" {
    return fare, ErrFareNotInTable
  }

  // get price from price group id
  var fareAmount currency.Amount
  for _, p := range f.PriceGroups {
    if p.GeographicalIntervalPrice[0].Id == geographicalIntervalPriceRef {
      fareAmount, err =
currency.NewAmount(p.GeographicalIntervalPrice[0].Amount, f.Currency)
      break
    }
  }

  // get user profile
  var userProfile UserProfile
  for _, u := range f.UserProfiles {
    if u.Id == userProfileRef {
      userProfile = u
      break
    }
  }

  // get sales offer package
  var salesOfferPackage SalesOfferPackage
  for _, s := range f.SalesOfferPackages {
    if s.Id == salesOfferPackageRef {
      salesOfferPackage = s
      break
    }
  }

  // get preassigned fare product
  var preassignedFareProduct PreassignedFareProduct
  for _, p := range f.FareProducts {
    if p.Id == preassignedFareProductRef {
      preassignedFareProduct = p
```

```
      break
    }
  }

  // get ValidBetween
  var validBetween ValidBetween
  for _, t := range f.Tariffs {
    if t.Id == tariffRef {
      validBetween = t.ValidBetween
      break
    }
  }

  fare = Fare{
    Amount:                fareAmount,
    ValidBetween:          validBetween,
    UserProfile:           userProfile,
    SalesOfferPackage:     salesOfferPackage,
    PreassignedFareProduct: preassignedFareProduct,
  }

  return fare, err
}
```

internal/fares/dataset.go

```
package fares

import (
  "archive/zip"
  "context"
  "encoding/json"
  "encoding/xml"
  "io"
  "io/fs"
  "log"
  "net/http"
  "os"
  "path/filepath"
  "time"

  "github.com/cavaliergopher/grab/v3"
  "github.com/jackc/pgx/v5"
)

type FareObjects struct {
  Objects []FareObject
}

func AddFareObject(c *pgx.Conn, f *FareObject) (err error) {
  fareObjectJson, err := json.Marshal(f)
  if err != nil {
    return err
  }
```

```
  _, err = c.Exec(context.Background(), "insert into fares (fare_object)
values ($1)", fareObjectJson)
  if err ≠ nil {
    log.Printf("failed to insert fare object: %s", err)
    return err
  }
  return err
}

func AddXml(c *pgx.Conn, file io.ReadCloser) (err error) {
  defer file.Close()
  var obj FareObject
  err = xml.NewDecoder(file).Decode(&obj)
  if err ≠ nil {
    return err
  }
  AddFareObject(c, &obj)
  return err
}

func AddZip(c *pgx.Conn, path string) (err error) {
  openZipFile, err := zip.OpenReader(path)
  if err ≠ nil {
    log.Fatalf("failed to open zip file %s", path)
    return err
  }
  defer openZipFile.Close()
  for _, file := range openZipFile.File {
    if filepath.Ext(file.Name) == ".xml" {
      openFile, err := file.Open()
      if err ≠ nil {
        log.Fatalf("failed to open file %s in %s", file.Name, path)
        return err
      }
      AddXml(c, openFile)
    }
  }
  return err
}

func AddDir(c *pgx.Conn, dir string) (err error) {
  err = filepath.WalkDir(dir, func(path string, d fs.DirEntry, err error)
error {
    if err ≠ nil {
      return err
    }
    if filepath.Ext(d.Name()) == ".zip" {
      log.Printf("adding zip %s", d.Name())
      AddZip(c, path)
    }
    if filepath.Ext(d.Name()) == ".xml" {
      xml, err := os.Open(path)
      if err ≠ nil {
        return err
      }
```

```
      defer xml.Close()
      log.Printf("adding xml %s", d.Name())
      AddXml(c, xml)
    }
    return nil
  })
  return err
}

type ApiResponse struct {
  Results []ApiResult `json:"results"`
}

type ApiResult struct {
  Id  int    `json:"id"`
  Url string `json:"url"`
}

func GetDatasets(dir string, nocs string) (err error) {
  // create directory if it doesn't exist
  err = os.MkdirAll(dir, os.ModePerm)
  if err ≠ nil && !os.IsExist(err) {
    return err
  }
  url := os.Getenv("BODS_API_BASE") + "/fares/dataset/?noc=" + nocs +
"&status=published&api_key=" + os.Getenv("BODS_API_KEY")
  resp, err := http.Get(url)
  if err ≠ nil {
    return err
  }
  var apiResponse ApiResponse
  defer resp.Body.Close()
  err = json.NewDecoder(resp.Body).Decode(&apiResponse)
  if err ≠ nil {
    return err
  }

  var reqs []*grab.Request
  for _, result := range apiResponse.Results {
    req, err := grab.NewRequest(dir, result.Url)
    if err ≠ nil {
      return err
    }
    reqs = append(reqs, req)
  }

  client := grab.NewClient()
  respch := client.DoBatch(4, reqs...)

  t := time.NewTicker(time.Second)
  defer t.Stop()

  for resp := range respch {
    for {
      select {
```

```
      case <-t.C:
        log.Printf("downloading %v (%.2f%%)", resp.Request.URL(),
100*resp.Progress())
      case <-resp.Done:
        if err := resp.Err(); err != nil {
          log.Printf("download failed: %v", err)
        } else {
          log.Printf("downloaded %v", resp.Filename)
        }
        return err
      }
    }
  }

  return err
}
```

internal/api/api.go

```
package api

import (
  "encoding/json"
  "fmt"
  "log"
  "net/http"

  "github.com/imnatgreen/busfares/internal/agency"
  "github.com/imnatgreen/busfares/internal/colours"
  "github.com/imnatgreen/busfares/internal/router"
  "github.com/jackc/pgx/v5"
)

func HandleRequests(c *pgx.Conn, a *agency.Agencies) {
  http.HandleFunc("/api/", homepage)
  http.HandleFunc("/api/getfares", getFares(c, a))
  log.Fatal(http.ListenAndServe(":8080", nil))
}

func Handler(prefix string, c *pgx.Conn, a *agency.Agencies, col
*colours.Colours) http.Handler {
  mux := http.NewServeMux()
  mux.HandleFunc(prefix+"/", homepage)
  mux.HandleFunc(prefix+"/getfares", getFares(c, a))
  mux.HandleFunc(prefix+"/getcolour", getColour(col, a))
  return mux
}

func homepage(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, "busfares api")
}

func getFares(c *pgx.Conn, a *agency.Agencies) http.HandlerFunc {
  return func(w http.ResponseWriter, r *http.Request) {
    defer r.Body.Close()
```

```
    tripPlan := router.TripPlannerResponse{}
    err := json.NewDecoder(r.Body).Decode(&tripPlan)
    if err ≠ nil {
      fmt.Fprintln(w, ":/ an error occured. check logs for details.")
      log.Print(err)
    }
    fares, err := tripPlan.GetAllFares(c, a)
    if err ≠ nil {
      fmt.Fprintln(w, ":/ an error occured. check logs for details.")
      log.Print(err)
    }
    err = json.NewEncoder(w).Encode(fares)
    if err ≠ nil {
      fmt.Fprintln(w, ":/ an error occured. check logs for details.")
      log.Print(err)
    }
  }
}

func getColour(c *colours.Colours, a *agency.Agencies) http.HandlerFunc {
  return func(w http.ResponseWriter, r *http.Request) {
    defer r.Body.Close()
    ag := r.URL.Query().Get("agency")
    var noc colours.Noc
    if ag ≠ "" {
      aNoc, err := a.GetNoc(agency.AgencyId(ag))
      if err ≠ nil {
        noc = colours.Noc("")
      }
      noc = colours.Noc(aNoc)
    } else {
      noc = colours.Noc(r.URL.Query().Get("noc"))
    }
    colour, err := c.Get(noc, colours.Line(r.URL.Query().Get("line")))
    if err == nil {
      fmt.Fprint(w, colour)
    }
  }
}
```

internal/router/router.go

```
package router

import (
  "context"
  "encoding/json"
  "fmt"
  "log"
  "regexp"
  "sort"
  "time"

  "github.com/imnatgreen/busfares/internal/agency"
  "github.com/imnatgreen/busfares/internal/fares"
```

```
  "github.com/jackc/pgx/v5"
)

type TripPlannerResponse struct {
  Plan TripPlan `json:"plan"`
}

type TripPlan struct {
  Itineraries []Itinerary `json:"itineraries"`
}

type Itinerary struct {
  Legs []Leg `json:"legs"`
}

type Leg struct {
  StartTime            int64        `json:"startTime"`
  EndTime              int64        `json:"endTime"`
  Mode                 string       `json:"mode"`
  TransitLeg           bool         `json:"transitLeg"`
  AgencyTimeZoneOffset int          `json:"agencyTimeZoneOffset"`
  AgencyId             string       `json:"agencyId"`
  ServiceDate          string       `json:"serviceDate"`
  From                 LegVertex    `json:"from"`
  To                   LegVertex    `json:"to"`
  LegGeometry          LegGeometry  `json:"legGeometry"`
  RouteShortName       string       `json:"routeShortName"`
  Fares                []fares.Fare `json:"fares"`
}

type LegVertex struct {
  Name       string `json:"name"`
  StopId     string `json:"stopId"`
  VertexType string `json:"vertexType"`
}

type LegGeometry struct {
  Points string `json:"points"`
  Length int    `json:"length"`
}

func ParseJson(data []byte) (TripPlannerResponse, error) {
  var response TripPlannerResponse
  err := json.Unmarshal(data, &response)
  return response, err
}

// GetAllFares returns a list of fares for each leg of the
TripPlannerResponse, following the same structure
func (r *TripPlannerResponse) GetAllFares(c *pgx.Conn, a *agency.Agencies)
([][][]fares.Fare, error) {
  var err error

  // check how many legs are in each itinerary
  itineraryLengths := []int{}
```

```
  for _, itinerary := range r.Plan.Itineraries {
    itineraryLengths = append(itineraryLengths, len(itinerary.Legs))
  }

  itineraryFares := make([][][]fares.Fare, len(r.Plan.Itineraries))

  for i, itinerary := range r.Plan.Itineraries {
    // create slice to add fares to
    itineraryFares[i] = make([][]fares.Fare, itineraryLengths[i])

    for l, leg := range itinerary.Legs {
      if leg.TransitLeg {
        agencyId := agency.AgencyId(TrimId(leg.AgencyId))
        noc, err := a.GetNoc(agencyId)
        if err ≠ nil {
          return nil, err
        }
        legFares, err := leg.GetFares(c, noc)
        if err ≠ nil {
          return nil, err
        }
        itineraryFares[i][l] = legFares
      }
    }
  }
  return itineraryFares, err
}

// GetFares finds the possible fares for the given transit leg and returns
them
func (l *Leg) GetFares(c *pgx.Conn, n agency.Noc) (legFares []fares.Fare,
err error) {
  // find all possible fares
  from := fares.Naptan(TrimId(l.From.StopId))
  to := fares.Naptan(TrimId(l.To.StopId))
  fareSlice := []fares.Fare{}

  start := time.Now()

  query := fmt.Sprintf(`select *
  from fares
  where fare_object→'Lines' @> '[{"PublicCode": "%s", "OperatorRef":
{"Ref": "noc:%s"}}]'
    and fare_object→'ScheduledStopPoints' @> '[{"ScheduledStopPointRef":
"atco:%s"}, {"ScheduledStopPointRef": "atco:%s"}]';`, l.RouteShortName, n,
from, to)
  rows, err := c.Query(context.Background(), query)
  log.Printf("query took %s", time.Since(start))
  if err ≠ nil {
    return legFares, err
  }

  start = time.Now()
  var uuid []byte
  var obj fares.FareObject
```

```
  pgx.ForEachRow(rows, []any{&uuid, &obj}, func() error {
    fare, err := obj.GetFare(from, to)
    if err ≠ nil && err ≠ fares.ErrFareNotInTable {
      return err
    }
    if err ≠ fares.ErrFareNotInTable {
      fareSlice = append(fareSlice, fare)
    }
    return nil
  })
  log.Printf("processed query in %s", time.Since(start))

  // sort fares by type
  sort.Slice(fareSlice, func(i, j int) bool {
    return fareSlice[i].PreassignedFareProduct.Id <
fareSlice[j].PreassignedFareProduct.Id
  })

  // group fares by type into separate slices
  filteredFareSlices := make([][]fares.Fare, len(fareSlice))
  prevId := ""
  currentId := ""
  j := 0
  for i, fare := range fareSlice {
    if i == 0 {
      prevId = fare.PreassignedFareProduct.Id
    } else {
      prevId = currentId
    }
    currentId = fare.PreassignedFareProduct.Id
    if prevId ≠ currentId {
      j++
    }
    filteredFareSlices[j] = append(filteredFareSlices[j], fare)
  }

  // remove empty slices
  if len(filteredFareSlices) > 0 {
    filteredFareSlices = filteredFareSlices[:j+1]
  }

  // only keep newest fare for each type
  for _, fSlice := range filteredFareSlices {
    newestFare := fSlice[0]
    for _, fare := range fSlice {
      if
fare.ValidBetween.FromDate.Time.After(newestFare.ValidBetween.FromDate.Tim
e) {
        newestFare = fare
      }
    }
    legFares = append(legFares, newestFare)
  }
```

```
  log.Printf("found %d fares for leg %s, filtered to %d.", len(fareSlice),
l.RouteShortName, len(legFares))
  return legFares, err
}

// Removes the "X:" prefix present on IDs from the OTP API (where X is the
router ID)
func TrimId(id string) string {
  re := regexp.MustCompile(`^[0-9]+:`)
  trimmed := re.ReplaceAllString(id, "")
  return trimmed
}

func FilterTransitLegs(response TripPlannerResponse) []Leg {
  var transitLegs []Leg
  for _, itinerary := range response.Plan.Itineraries {
    for _, leg := range itinerary.Legs {
      if leg.TransitLeg {
        transitLegs = append(transitLegs, leg)
      }
    }
  }
  return transitLegs
}
```

frontend/embed.go

```
package frontend

import (
  "embed"
  "errors"
  "fmt"
  "io/fs"
  "log"
  "net/http"
  "os"
  "strings"
)

//go:generate npm i
//go:generate npm run build
//go:embed all:build
var files embed.FS

func SvelteKitHandler(path string) http.Handler {
  fsys, err := fs.Sub(files, "build")
  if err ≠ nil {
    log.Fatal(err)
  }
  filesystem := http.FS(fsys)

  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    path := strings.TrimPrefix(r.URL.Path, path)
```

```
    // try if file exists at path, if not append .html (SvelteKit adapter-
static specific)
    _, err := filesystem.Open(path)
    if errors.Is(err, os.ErrNotExist) {
      path = fmt.Sprintf("%s.html", path)
    }
    r.URL.Path = path
    http.FileServer(filesystem).ServeHTTP(w, r)
  })
}
```

frontend/package.json

```
{
  "name": "frontend",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "dev": "VITE_API_URL=http://localhost:8080 vite dev",
    "build": "vite build",
    "preview": "vite preview",
    "check": "svelte-kit sync && svelte-check --tsconfig ./tsconfig.json",
    "check:watch": "svelte-kit sync && svelte-check --tsconfig
./tsconfig.json --watch",
    "lint": "prettier --plugin-search-dir . --check . && eslint .",
    "format": "prettier --plugin-search-dir . --write ."
  },
  "devDependencies": {
    "@sveltejs/adapter-auto": "next",
    "@sveltejs/adapter-static": "^1.0.0-next.46",
    "@sveltejs/kit": "next",
    "@tailwindcss/forms": "^0.5.3",
    "@types/leaflet": "^1.9.0",
    "@typescript-eslint/eslint-plugin": "^5.27.0",
    "@typescript-eslint/parser": "^5.27.0",
    "autoprefixer": "^10.4.13",
    "eslint": "^8.16.0",
    "eslint-config-prettier": "^8.3.0",
    "eslint-plugin-svelte3": "^4.0.0",
    "postcss": "^8.4.19",
    "prettier": "^2.6.2",
    "prettier-plugin-svelte": "^2.7.0",
    "simple-svelte-autocomplete": "^2.5.1",
    "svelte": "^3.44.0",
    "svelte-check": "^2.7.1",
    "svelte-preprocess": "^4.10.7",
    "tailwindcss": "^3.2.4",
    "tslib": "^2.3.1",
    "typescript": "^4.7.4",
    "vite": "^3.1.0",
    "vite-plugin-iso-import": "^0.1.3"
  },
  "type": "module",
  "dependencies": {
    "@googlemaps/polyline-codec": "^1.0.28",
```

```
    "currency.js": "^2.0.4",
    "leaflet": "^1.9.2",
    "leaflet.vectorgrid": "^1.3.0"
  }
}
```

frontend/tsconfig.json

```
{
  "extends": "./.svelte-kit/tsconfig.json",
  "compilerOptions": {
    "allowJs": true,
    "checkJs": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "plugins": [{ "name": "vite-plugin-iso-import" }],
    "resolveJsonModule": true,
    "skipLibCheck": true,
    "sourceMap": true,
    "strict": false
  }
}
```

frontend/svelte.config.js

```
import adapter from '@sveltejs/adapter-static';
import preprocess from 'svelte-preprocess';

/** @type {import('@sveltejs/kit').Config} */
const config = {
  preprocess: preprocess({
    postcss: true
  }),

  kit: {
    adapter: adapter()
  }
};

export default config;
```

frontend/postcss.config.cjs

```
module.exports = {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
}
```

frontend/tailwind.config.cjs

```
/** @type {import('tailwindcss').Config} */

const defaultTheme = require('tailwindcss/defaultTheme')
```

```
module.exports = {
  content: ['./src/**/*.{html,js,svelte,ts}'],
  theme: {
    extend: {
      fontFamily: {
        'sans': ['Manrope', defaultTheme.fontFamily.sans],
      },
      animation: {
        'spin-fast': 'spin .75s linear infinite',
      }
    },
  },
  plugins: [
    require('@tailwindcss/forms'),
  ],
}
```

frontend/vite.config.ts

```
import { sveltekit } from '@sveltejs/kit/vite';
import { isoImport } from 'vite-plugin-iso-import'
import type { UserConfig } from 'vite';

const config: UserConfig = {
  plugins: [sveltekit(), isoImport()],
};

export default config;
```

frontend/src/app.d.ts

```
// See https://kit.svelte.dev/docs/types#app
// for information about these interfaces
// and what to do when importing types
declare namespace App {
  // interface Locals {}
  // interface PageData {}
  // interface Error {}
  // interface Platform {}
}

interface ImportMetaEnv {
  VITE_API_URL: string;
}

declare module 'leaflet?client' {
  import all from 'leaflet';
  export = all;
}
```

frontend/src/app.html

```
<!DOCTYPE html>
<html lang="en">
```

```
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%sveltekit.assets%/favicon.png" />
    <meta name="viewport" content="width=device-width" />
    %sveltekit.head%
  </head>
  <body>
    <div>%sveltekit.body%</div>
  </body>
</html>
```

frontend/src/app.css

```
@import
url('https://fonts.googleapis.com/css2?family=Manrope:wght@400;700&display
=swap');
@tailwind base;
@tailwind components;
@tailwind utilities;

@supports (-webkit-hyphens:none) {
  input::-webkit-datetime-edit {
    line-height: 1;
    padding: 0;
    margin-bottom: -2px;
  }
}

input[type="time"]::-webkit-calendar-picker-indicator,
input[type="date"]::-webkit-calendar-picker-indicator {
  background: none;
  display: none;
  -webkit-appearance: none;
}
```

frontend/src/routes/+layout.ts

```
export const prerender = true;
```

frontend/src/routes/+layout.svelte

```
<script>
  import "../app.css";
</script>

<slot />
```

frontend/src/routes/+page.svelte

```
<script lang="ts">
  import { onMount } from 'svelte';
  import { fade, fly } from 'svelte/transition'
  import { otpBase, apiUrl, getLineColour, latLngString, searchPlaces,
placeToName, reverseGeocode } from '$lib/utils';
```

```
  import L from 'leaflet?client';
  import 'leaflet/dist/leaflet.css';

  import currency from 'currency.js';
  import { decode } from '@googlemaps/polyline-codec';
  import Leaflet from '$lib/Leaflet.svelte';
  import Polyline from '$lib/Polyline.svelte';
  import Popup from '$lib/Popup.svelte';
  import ItineraritySummary from '$lib/ItineraritySummary.svelte';
  import Button from '$lib/Button.svelte';
  import ItineraryLeg from '$lib/ItineraryLeg.svelte';
  import Tooltip from '$lib/Tooltip.svelte';

  import AutoComplete from 'simple-svelte-autocomplete';

  let map;
  let leaflet;

  let gettingPlan = false;

  const initialView = [53.71,-2.24];
  function resizeMap() {
    if(map) { map.invalidateSize(); }
  }

  function resetMapView() {
    map.setView(initialView, 10);
  }

  // declare search parameters
  let from;
  let to;
  let now = new Date();
  let datePicker;
  let timePicker;
  let date = now.toISOString().split('T')[0];
  let time = now.getHours().toString().padStart(2, '0') + ':' +
now.getMinutes().toString().padStart(2, '0');
  let arriveBy = false;
  let editSearch = true;

  // switches states when the edit button is selected
  const openEditSearch = () => {
    editSearch = true;
    showItineraryDetail = false;
  }

  let leftRightTransition = false;

  const getFares = async (tripPlan: object) => {
    const res = await fetch(apiUrl('/api/getfares'), {
      method: 'POST',
      body: JSON.stringify(tripPlan),
    });
    return await res.json();
```

```
  };

  let tripPlan;
  let tripPlanFares;
  $: mapLines = [];
  let currentItinerary = 0;
  let showItineraryDetail = false;
  let selectedFares = [];
  let totalFares = [];

  const getPlan = async () => {
    tripPlan = undefined;
    tripPlanFares = undefined;
    gettingPlan = true;
    editSearch = false;
    showItineraryDetail = false;
    const res = await fetch(otpBase+'/routers/default/plan?'+ new
URLSearchParams({
      fromPlace:
from.properties.name+'::'+from.geometry.coordinates[1]+','+from.geometry.c
oordinates[0],
      toPlace:
to.properties.name+'::'+to.geometry.coordinates[1]+','+to.geometry.coordin
ates[0],
      date: date,
      time: time,
      mode: 'TRANSIT,WALK',
      maxWalkDistance: '2500',
      arriveBy: arriveBy ? 'true' : 'false',
      wheelchair: 'false',
      showIntermediateStops: 'true',
      locale: 'en'
    }));
    tripPlan = await res.json();
    gettingPlan = false;
    selectedFares.length = 0;
    totalFares.length = 0;
    tripPlanFares = getFares(tripPlan).then((res) => {
      res.forEach((legs, i) => {
        selectedFares = [...selectedFares, []];
        totalFares = [...totalFares, undefined];
        legs.forEach((fares, l) => {
          tripPlan.plan.itineraries[i].legs[l].fares = fares;
          if (tripPlan.plan.itineraries[i].legs[l].fares) {
            let defaultFare =
tripPlan.plan.itineraries[i].legs[l].fares.find(f =>
f.preassignedFareProduct.id == 'Trip@Single');
            selectedFares[i] = [...selectedFares[i], defaultFare];
          } else {
            selectedFares[i] = [...selectedFares[i], undefined];
          }
        })
      });
    });
```

```
    for (const [i, itinerary] of tripPlan.plan.itineraries.entries()) {
      for (const [l, leg] of itinerary.legs.entries()) {
        if (leg.mode === 'BUS') {
          getLineColour(leg.agencyId.substring(2),
leg.routeShortName).then((col) => {
            if (col !== '') {
              tripPlan.plan.itineraries[i].legs[l].colour = col;
            }
          });
        }
      }
    }
  }

  $: {
    if (totalFares) {
      selectedFares.forEach((itineraries ,i) => {
        let total = currency(0);
        itineraries.forEach((fare, f) => {
          if (fare) {
            total = total.add(fare.amount.number);
          }
        })
        totalFares[i] = total.value;
      })
    }
  }

  $: if(tripPlan) {
    if (tripPlan.plan.itineraries.length > 0) {
      drawLine(tripPlan.plan.itineraries[currentItinerary].legs);
    }
  }

  const drawLine = (legs) => {
    mapLines = [];
    legs.map((leg, i) => {
      if (leg.legGeometry.points) {
        let line = {
          id: i,
          latLngs: decode(leg.legGeometry.points),
          mode: leg.mode,
          colour: leg.colour ? leg.colour : '#6366F1',
          routeShortName: leg.routeShortName,
          duration: leg.duration
        }
        mapLines = [...mapLines, line];
      }
    });
    leaflet.fitMapToLines();
  }

  let mapClickPopupContent;
  let mapClickPopupEvent;
  let mapClickPopup;
```

```
  // this only runs when the DOM is created
  onMount(async () => {
    mapClickPopup = L.popup({minWidth: 200, maxWidth: 200});
  })

  let popupPlace;
  let popupPlaceLabel;

  const mapClick = async(e) => {
    popupPlace = null;
    mapClickPopupEvent = e.detail;
    mapClickPopup
      .setLatLng(mapClickPopupEvent.latlng)
      .setContent(mapClickPopupContent)
      .openOn(map);
    popupPlaceLabel = latLngString(mapClickPopupEvent.latlng);
    popupPlace = await reverseGeocode(mapClickPopupEvent.latlng.lat,
mapClickPopupEvent.latlng.lng);
    if (popupPlace) {
      popupPlaceLabel = placeToName(popupPlace);
    }
  }

  const setLocation = (location, latlng) => {
    openEditSearch();
    if (location == 'from') {
      from = popupPlace;
    } else if (location == 'to') {
      to = popupPlace;
    }
    if (mapClickPopup) {
      mapClickPopup.remove();
    }
  }

  // calls searchPlaces with the current map position set to prioritise
results within the map view
  const searchPlacesBias = async (query) => {
    let mapCenter = map.getCenter();
    let mapZoom = map.getZoom();
    return await searchPlaces(query, mapCenter.lat, mapCenter.lng,
mapZoom);
  }
</script>
<svelte:head>
  <title>bus fares.</title>
</svelte:head>
<svelte:window on:resize={resizeMap} />

<!— tailwind css classes used to easily define styles —>
<div class="flex flex-col md:flex-row w-full h-[calc(100vh-env(safe-area-
inset-bottom))] p-4 gap-4 bg-gradient-to-br from-pink-100 via-purple-100
to-indigo-200">
  <div class="h-1/2 md:h-full md:w-1/2 lg:w-2/5 overflow-hidden flex-none
rounded-lg shadow-lg bg-white">
```

```
    <div class="w-full h-full p-4 overflow-auto">
      <div class="grid grid-rows-[1fr] grid-cols-[1fr]">
        {#if editSearch}
          <div transition:fly="{{y:-100, duration:500}}" class="row-[1]
col-[1]">
            <h1 class="text-2xl">bus fares.</h1>
            <div class="text-gray-500 text-sm">
              <p>get bus, but cheap.</p>
              <div class="flex flex-row space-x-px items-center">
                <span>made with</span>
                <svg
                  xmlns="http://www.w3.org/2000/svg"
                  class="h-5 w-5 pb-px hover:text-red-400 transition"
                  viewBox="0 0 20 20"
                  fill="currentColor"
                >
                  <path
                    fill-rule="evenodd"
                    d="M3.172 5.172a4 4 0 015.656 0L10 6.343l1.172-1.171a4
4 0 115.656 5.656L10 17.657l-6.828-6.829a4 4 0 010-5.656z"
                    clip-rule="evenodd"
                  />
                </svg>
                <span>by nathan :)</span>
              </div>
            </div>
            <p class="mb-1 mt-4 text-gray-500 text-sm">okay, first, where
to?</p>
            <div class="relative">
              <div class="pointer-events-none z-10 absolute inset-y-0
left-0 flex items-center pl-3">
                <span class="text-gray-500">from</span>
              </div>
              <AutoComplete
                bind:selectedItem={from}
                searchFunction={searchPlacesBias}
                cleanUserText={false}
                delay={200}
                labelFunction={(place) => {return place.properties.name}}
                localFiltering={false}
                localSorting={false}
                placeholder="enter origin"
                hideArrow
                noInputStyles
                className="w-full mt-1"
                inputClassName="
                  px-2 py-1 pl-14 block w-full rounded-md  border-gray-300
shadow-sm
                  focus:border-indigo-300 focus:ring focus:ring-indigo-200
focus:ring-opacity-50
                "
                dropdownClassName="
                  mt-1 w-full rounded-md border-indigo-300 shadow-sm
                  focus:border-indigo-300 focus:ring focus:ring-indigo-200
focus:ring-opacity-50"
```

```
              >
                <div slot="item" let:item let:label>
                  <span>
                    <b>{@html label}</b>
                    <span class="text-
sm">{placeToName(item).replace(item.properties.name, '').replace(', ',
'')}</span>
                  </span>
                </div>
              </AutoComplete>
            </div>

            <div class="relative">
              <div class="pointer-events-none z-10 absolute inset-y-0
left-0 flex items-center pl-3">
                <span class="text-gray-500">to</span>
              </div>
              <AutoComplete
                bind:selectedItem={to}
                searchFunction={searchPlacesBias}
                cleanUserText={false}
                delay={200}
                labelFunction={(place) => {return place.properties.name}}
                localFiltering={false}
                localSorting={false}
                placeholder="enter destination"
                hideArrow
                noInputStyles
                className="w-full mt-1"
                inputClassName="
                  px-2 py-1 pl-14 block w-full rounded-md  border-gray-300
shadow-sm
                  focus:border-indigo-300 focus:ring focus:ring-indigo-200
focus:ring-opacity-50
                "
                dropdownClassName="
                  mt-1 w-full rounded-md border-indigo-300 shadow-sm
                  focus:border-indigo-300 focus:ring focus:ring-indigo-200
focus:ring-opacity-50"
              >
                <div slot="item" let:item let:label>
                  <span>
                    <b>{@html label}</b>
                    <span class="text-
sm">{placeToName(item).replace(item.properties.name, '').replace(', ',
'')}</span>
                  </span>
                </div>
              </AutoComplete>
            </div>

            <p class="mb-1 mt-4 text-gray-500 text-sm">great! now,
when?</p>
            <div class="mt-1 flex flex-row gap-1 flex-wrap">
              <div class="relative">
```

```
                <div class="pointer-events-none absolute inset-y-0 left-0
flex items-center pl-2 text-gray-500">
                  <!— change the icon based on state —>
                  {#if arriveBy}
                    <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0
20 20" fill="currentColor" class="w-5 h-5">
                      <path fill-rule="evenodd" d="M3 4.25A2.25 2.25 0
015.25 2h5.5A2.25 2.25 0 0113 4.25v2a.75.75 0 01-1.5 0v-2a.75.75 0 00-.75-
.75h-5.5a.75.75 0 00-.75.75v11.5c0 .414.336.75.75.75h5.5a.75.75 0 00.75-
.75v-2a.75.75 0 011.5 0v2A2.25 2.25 0 0110.75 18h-5.5A2.25 2.25 0 013
15.75V4.25z" clip-rule="evenodd" />
                      <path fill-rule="evenodd" d="M19 10a.75.75 0 00-.75-
.75H8.704l1.048-.943a.75.75 0 10-1.004-1.114l-2.5 2.25a.75.75 0 000
1.114l2.5 2.25a.75.75 0 101.004-1.114l-1.048-.943h9.546A.75.75 0 0019 10z"
clip-rule="evenodd" />
                    </svg>
                  {:else}
                    <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0
20 20" fill="currentColor" class="w-5 h-5">
                      <path fill-rule="evenodd" d="M3 4.25A2.25 2.25 0
015.25 2h5.5A2.25 2.25 0 0113 4.25v2a.75.75 0 01-1.5 0v-2a.75.75 0 00-.75-
.75h-5.5a.75.75 0 00-.75.75v11.5c0 .414.336.75.75.75h5.5a.75.75 0 00.75-
.75v-2a.75.75 0 011.5 0v2A2.25 2.25 0 0110.75 18h-5.5A2.25 2.25 0 013
15.75V4.25z" clip-rule="evenodd" />
                      <path fill-rule="evenodd" d="M6 10a.75.75 0 01.75-
.75h9.546l-1.048-.943a.75.75 0 111.004-1.114l2.5 2.25a.75.75 0 010 1.114l-
2.5 2.25a.75.75 0 11-1.004-1.114l1.048-.943H6.75A.75.75 0 016 10z" clip-
rule="evenodd" />
                    </svg>
                  {/if}
                </div>
                <select
                  aria-label="Leave at time or arrive by time?"
                  bind:value={arriveBy} label="arrive by"
                  class="
                    px-2 pl-7 {arriveBy ? 'pr-7' : 'pr-6'} py-1 block
rounded-md  border-gray-300 shadow-sm
                    focus:border-indigo-300 focus:ring focus:ring-indigo-
200 focus:ring-opacity-50
                  "
                >
                  <option value={false}>leave at</option>
                  <option value={true}>arrive by</option>
                </select>
              </div>
              <div class="relative">
                <button aria-label="Open date picker" on:click={() =>
datePicker.showPicker()} class="absolute inset-y-0 left-0 flex items-
center pl-2 text-gray-500">
                  <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20
20" fill="currentColor" class="w-5 h-5">
                    <path d="M5.25 12a.75.75 0 01.75-.75h.01a.75.75 0
01.75.75v.01a.75.75 0 01-.75.75H6a.75.75 0 01-.75-.75V12zM6 13.25a.75.75 0
00-.75.75v.01c0 .414.336.75.75.75h.01a.75.75 0 00.75-.75V14a.75.75 0 00-
.75-.75H6zM7.25 12a.75.75 0 01.75-.75h.01a.75.75 0 01.75.75v.01a.75.75 0
```

```
01-.75.75H8a.75.75 0 01-.75-.75V12zM8 13.25a.75.75 0 00-.75.75v.01c0
.414.336.75.75.75h.01a.75.75 0 00.75-.75V14a.75.75 0 00-.75-.75H8zM9.25
10a.75.75 0 01.75-.75h.01a.75.75 0 01.75.75v.01a.75.75 0 01-
.75.75H10a.75.75 0 01-.75-.75V10zM10 11.25a.75.75 0 00-.75.75v.01c0
.414.336.75.75.75h.01a.75.75 0 00.75-.75V12a.75.75 0 00-.75-.75H10zM9.25
14a.75.75 0 01.75-.75h.01a.75.75 0 01.75.75v.01a.75.75 0 01-
.75.75H10a.75.75 0 01-.75-.75V14zM12 9.25a.75.75 0 00-.75.75v.01c0
.414.336.75.75.75h.01a.75.75 0 00.75-.75V10a.75.75 0 00-.75-.75H12zM11.25
12a.75.75 0 01.75-.75h.01a.75.75 0 01.75.75v.01a.75.75 0 01-
.75.75H12a.75.75 0 01-.75-.75V12zM12 13.25a.75.75 0 00-.75.75v.01c0
.414.336.75.75.75h.01a.75.75 0 00.75-.75V14a.75.75 0 00-.75-.75H12zM13.25
10a.75.75 0 01.75-.75h.01a.75.75 0 01.75.75v.01a.75.75 0 01-
.75.75H14a.75.75 0 01-.75-.75V10zM14 11.25a.75.75 0 00-.75.75v.01c0
.414.336.75.75.75h.01a.75.75 0 00.75-.75V12a.75.75 0 00-.75-.75H14z" />
                    <path fill-rule="evenodd" d="M5.75 2a.75.75 0
01.75.75V4h7V2.75a.75.75 0 011.5 0V4h.25A2.75 2.75 0 0118 6.75v8.5A2.75
2.75 0 0115.25 18H4.75A2.75 2.75 0 012 15.25v-8.5A2.75 2.75 0 014.75
4H5V2.75A.75.75 0 015.75 2zm-1 5.5c-.69 0-1.25.56-1.25 1.25v6.5c0 .69.56
1.25 1.25 1.25h10.5c.69 0 1.25-.56 1.25-1.25v-6.5c0-.69-.56-1.25-1.25-
1.25H4.75z" clip-rule="evenodd" />
                  </svg>
                </button>
                <input
                  aria-label="Date picker"
                  bind:this={datePicker}
                  type="date" bind:value={date} label="date"
                  class="
                    px-2 pl-8 py-1 block rounded-md  border-gray-300
shadow-sm
                    focus:border-indigo-300 focus:ring focus:ring-indigo-
200 focus:ring-opacity-50
                  "
                />
              </div>
              <div class="relative">
                <button aria-label="Open time picker" on:click={() ⇒
timePicker.showPicker()} class="absolute inset-y-0 left-0 flex items-
center pl-2 text-gray-500">
                  <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20
20" fill="currentColor" class="w-5 h-5">
                    <path fill-rule="evenodd" d="M10 18a8 8 0 100-16 8 8 0
000 16zm.75-13a.75.75 0 00-1.5 0v5c0 .414.336.75.75.75h4a.75.75 0 000-
1.5h-3.25V5z" clip-rule="evenodd" />
                  </svg>
                </button>
                <input
                  aria-label="Time picker"
                  bind:this={timePicker}
                  type="time" bind:value={time} label="time"
                  class="
                    px-2 pl-8 py-1 block rounded-md  border-gray-300
shadow-sm
                    focus:border-indigo-300 focus:ring focus:ring-indigo-
200 focus:ring-opacity-50
                  "
```

```
                />
              </div>
            </div>

            <p class="mb-1 mt-4 text-gray-500 text-sm">and, finally...</p>
            <Button wrapClasses="mt-1" on:click={getPlan}>
              <svg slot="icon" xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 20 20" fill="currentColor" class="w-5 h-5">
                <path fill-rule="evenodd" d="M8.157 2.175a1.5 1.5 0 00-
1.147 0l-4.084 1.69A1.5 1.5 0 002 5.251v10.877a1.5 1.5 0 002.074
1.386l3.51-1.453 4.26 1.763a1.5 1.5 0 001.146 0l4.083-1.69A1.5 1.5 0 0018
14.748V3.873a1.5 1.5 0 00-2.073-1.386l-3.51 1.452-4.26-1.763zM7.58
5a.75.75 0 01.75.75v6.5a.75.75 0 01-1.5 0v-6.5A.75.75 0 017.58 5zm5.59
2.75a.75.75 0 00-1.5 0v6.5a.75.75 0 001.5 0v-6.5z" clip-rule="evenodd" />
              </svg>
              <span>find journey</span>
            </Button>
            <hr class="my-2">
          </div>
        {:else}
          <div transition:fly="{{y:100, duration:500}}" class="row-[1]
col-[1] text-gray-500 text-sm">
            <div>journey from <span class="text-
black">{from.properties.name}</span> to <span class="text-
black">{to.properties.name}</span>, {arriveBy ? 'arriving by' : 'leaving
at'} {date} {time}</div>
            <Button on:click={openEditSearch} wrapClasses="mt-1"
classes="text-sm pl-7">
              <svg slot="icon" xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 20 20" fill="currentColor" class="w-4 h-4">
                <path d="M2.695 14.763l-1.262 3.154a.5.5 0 00.65.65l3.155-
1.262a4 4 0 001.343-.885L17.5 5.5a2.121 2.121 0 00-3-3L3.58 13.42a4 4 0
00-.885 1.343z" />
              </svg>
              <span>edit</span>
            </Button>
            <hr class="my-2">
          </div>
        {/if}
      </div>
      <!— key is used to redraw the results when the state of editSearch
changes —>
      {#key editSearch}
        <div in:fly="{
            editSearch?{y:-100,duration:500, delay:0, opacity:1}
            :{y:25,duration:500,delay:500}
          }"
          on:introstart="{() => leftRightTransition = false}"
          on:introend="{() => leftRightTransition = true}"
          class="{editSearch ? 'opacity-100':''} transition-opacity
duration-500 -z-10" >
          {#if tripPlan}
            <div in:fly="{
              editSearch?{y:-100,duration:500, delay:0, opacity:1}
              :{y:25,duration:500,delay:500}
```

```
            }">
              {#if tripPlan.plan.itineraries.length > 0}
                <div class="grid grid-rows-[1fr] grid-cols-[1fr]">
                  <!— shown when a summary card is clicked —>
                  {#if showItineraryDetail}
                    <div class="row-[1] col-[1]"
                      in:fly|local="{{x:100,duration:500, delay:0}}"
                      out:fly|local="{{x:100,duration:500, delay:0}}"
                    >
                      <Button on:click={() => showItineraryDetail = false}
classes="text-sm pl-7" wrapClasses="mb-2">
                        <svg slot="icon"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20" fill="currentColor"
class="w-5 h-5">
                          <path fill-rule="evenodd" d="M17 10a.75.75 0 01-
.75.75H5.612l4.158 3.96a.75.75 0 11-1.04 1.08l-5.5-5.25a.75.75 0 010-
1.08l5.5-5.25a.75.75 0 111.04 1.08L5.612 9.25H16.25A.75.75 0 0117 10z"
clip-rule="evenodd" />
                        </svg>
                        <span>back</span>
                      </Button>
                      {#each
tripPlan.plan.itineraries[currentItinerary].legs as leg, l}
                        <p>{new Date(leg.startTime).toLocaleString()}:
{leg.mode} {leg.mode=='BUS'? '('+leg.routeShortName+' towards
'+leg.headsign+')' : ''} from {leg.from.name} to {leg.to.name}</p>
                        {#if leg.fares}
                          <div class="relative mt-1">
                            <div class="pointer-events-none absolute
inset-y-0 left-0 flex items-center pl-2 text-gray-500">
                              <svg xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 20 20" fill="currentColor" class="w-4 h-4">
                                <path fill-rule="evenodd" d="M1 4a1 1 0
011-1h16a1 1 0 011 1v8a1 1 0 01-1 1H2a1 1 0 01-1-1V4zm12 4a3 3 0 11-6 0 3
3 0 016 0zM4 9a1 1 0 100-2 1 1 0 000 2zm13-1a1 1 0 11-2 0 1 1 0 012
0zM1.75 14.5a.75.75 0 000 1.5c4.417 0 8.693.603 12.749 1.73 1.111.309
2.251-.512 2.251-1.696v-.784a.75.75 0 00-1.5 0v.784a.272.272 0 01-
.35.25A49.043 49.043 0 001.75 14.5z" clip-rule="evenodd" />
                              </svg>
                            </div>
                            <select
                              aria-label="Selected fare"

bind:value={selectedFares[currentItinerary][l]}
                              class="
                                px-2 pl-7 pr-7 py-1 block rounded-md
border-gray-300 shadow-sm text-sm
                                focus:border-indigo-300 focus:ring
focus:ring-indigo-200 focus:ring-opacity-50
                              "
                            >
                              {#each leg.fares as fare}
                                <option
                                  value={fare}
```

```
                                  selected={fare.preassignedFareProduct.id
== 'Trip@Single'}
                                >
                                  {fare.preassignedFareProduct.name}: {new
Intl.NumberFormat('en-GB',{ style: 'currency', currency:
fare.amount.currency }).format(currency(fare.amount.number).value)}
                                </option>
                              {/each}
                            </select>
                          </div>
                        {/if}
                        <hr class="my-2">
                      {/each}
                    </div>
                  {:else}
                    <div class="row-[1] col-[1]"
                      in:fly|local="{{x:-100,duration:500, delay:0}}"
                      out:fly|local="{{x:-100,duration:500, delay:0}}"
                    >
                      <!— shows a summary card for each itinerary —>
                      {#each tripPlan.plan.itineraries as itinerary, i}
                        <button on:click={() => {
                          currentItinerary = i;
                          showItineraryDetail = true;
                        }}
                          class="mb-2 w-full"
                        >
                          <ItinerarySummary
                            itinerary={itinerary}
                            totalFare={totalFares[i]}
                            on:click={() => currentItinerary = i}
                          />
                        </button>
                      {/each}
                    </div>
                  {/if}
                </div>
              {:else}
                <p>sorry, no journeys found</p>
              {/if}
            </div>
          {:else}
            {#if gettingPlan}
              <svg out:fade="{{duration:250}}" class="animate-spin-fast w-
8 h-8" viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg">
                <path
d="M12,1A11,11,0,1,0,23,12,11,11,0,0,0,12,1Zm0,19a8,8,0,1,1,8-
8A8,8,0,0,1,12,20Z" opacity=".25"/>
                <path d="M10.14,1.16a11,11,0,0,0-
9,8.92A1.59,1.59,0,0,0,2.46,12,1.52,1.52,0,0,0,4.11,10.7a8,8,0,0,1,6.66-
6.61A1.42,1.42,0,0,0,12,2.69h0A1.57,1.57,0,0,0,10.14,1.16Z"/>
              </svg>
            {/if}
          {/if}
        </div>
```

```
      {/key}
    </div>
  </div>
  <div class="h-1/2 md:h-full w-full md:w-1/2 lg:w-3/5 md:pr-4 flex-none">
    <!— the map instance is bound to the variable leaflet so it can be
accessed in the main code —>
    <Leaflet classes="h-[calc(50vh-2rem)] md:h-[calc(100vh-2rem)] w-full
rounded-lg overflow-hidden z-10 shadow-lg" bind:map bind:this={leaflet}
view={initialView} zoom={10} on:click={mapClick}>
      {#each mapLines as line}
        <Polyline latLngs={line.latLngs} options={{
          color: '#ffffff',
          dashArray: line.mode=='WALK' ? '.1 11' : null,
          weight: 6,
        }}/>
        <Polyline latLngs={line.latLngs} options={{
          color: line.mode=='WALK' ? 'grey' : line.colour,
          dashArray: line.mode=='WALK' ? '.1 11' : null,
          weight: 5,
        }}>
          <!— each bus line has its route number overlaid —>
          {#if line.mode=='BUS'}
            {#key currentItinerary}
              <Tooltip permanent interactive opacity={1}
direction="center">
                <ItineraryLeg leg={line} classes='font-sans' small
border={false}/>
              </Tooltip>
            {/key}
          {/if}
        </Polyline>
      {/each}
    </Leaflet>
  </div>
</div>

<!— the popup content is hidden and bound to mapClickPopupContent; its
current content is shown on the map when the popup is opened —>
<div class="hidden">
  <div bind:this={mapClickPopupContent}>
    {#if mapClickPopupEvent}
      <div class="flex flex-col gap-2 font-sans text-sm">
        {popupPlaceLabel}
        <div class="flex gap-2">
          <Button on:click={() => setLocation('from',
mapClickPopupEvent.latlng)} classes="text-sm">from here</Button>
          <Button on:click={() => setLocation('to',
mapClickPopupEvent.latlng)} classes="text-sm">to here</Button>
        </div>
      </div>
    {/if}
  </div>
</div>
```

frontend/src/lib/Tooltip.svelte

```
<script lang="ts">
  import L from 'leaflet?client';
  import { getContext } from 'svelte';

  let classNames: string | undefined = undefined;
  export { classNames as class };

  export let permanent = false;
  export let sticky = false;
  export let interactive = false;
  export let direction: L.Direction = 'auto';
  export let opacity = 0.9;

  export let tooltip: L.Tooltip | undefined = undefined;

  let showContents = permanent;
  let tooltipOpen = permanent;

  const layer = getContext<() => L.Layer>('layer')();
  function createPopup(tooltipElement: HTMLElement) {
    tooltip = L.tooltip({permanent, sticky, interactive, direction,
opacity}).setContent(tooltipElement);
    layer.bindTooltip(tooltip);

    layer.on('tooltipopen', () => {
      tooltipOpen = true;
      showContents = true;
    });

    layer.on('tooltipclose', () => {
      tooltipOpen = false;
      // wait for the tooltip to completely fade out before destroying it,
otherwise the contents disappear too early.
      setTimeout(() => {
        if (!tooltipOpen) {
          showContents = false;
        }
      }, 500);
    });

    return {
      destroy() {
        if (tooltip) {
          layer.unbindTooltip();
          tooltip.remove();
          tooltip = undefined;
        }
      },
    };
  }
</script>

<div class="hidden">
```

```
  <div use:createPopup class={classNames}>
    {#if showContents}
      <slot />
    {/if}
  </div>
</div>
```

frontend/src/lib/Leaflet.svelte

```
<script>
  import { createEventDispatcher, setContext } from 'svelte';
  import L from 'leaflet?client';
  import 'leaflet/dist/leaflet.css';
  export let classes = '';

  // Must set either bounds, or view and zoom.
  export let bounds = undefined;
  export let view = undefined;
  export let zoom = undefined;
  let mapProp = undefined;
  export { mapProp as map };

  export const invalidateSize = () => map?.invalidateSize();

  const dispatch = createEventDispatcher();

  let map;
  let mapFeatureGroup;
  $: mapProp = map;

  export const getMap = () => map;
  export const getMapFeatureGroup = () => mapFeatureGroup;
  setContext('layerGroup', getMap);
  setContext('featureGroup', getMapFeatureGroup);
  setContext('layer', getMap);
  setContext('map', getMap);

  function createLeaflet(node) {
    map = L.map(node, {zoomControl:false})
      .addControl(L.control.scale({metric: true, imperial: false,
position:'bottomright'}))
      .addControl(L.control.zoom({position:'bottomright'}))
      .on('zoom', (e) => dispatch('zoom', e))
      .on('click', (e) => dispatch('click', e));
    if(bounds) {
      map.fitBounds(bounds)
    } else {
      map.setView(view, zoom);
    }


L.tileLayer('https://{s}.basemaps.cartocdn.com/rastertiles/voyager/{z}/{x}
/{y}' + (L.Browser.retina ? '@2x.png' : '.png'), {
```

```
      attribution:'&copy; <a
href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>, &copy; <a
href="https://carto.com/attributions">CARTO</a>',
      subdomains: 'abcd',
      maxZoom: 20,
      minZoom: 0
    }).addTo(map);

    mapFeatureGroup = L.featureGroup().addTo(map);

    return {
      destroy() {
        map.remove();
        map = undefined;
      },
    };
  }

  export function fitMapToLines() {
    if(mapFeatureGroup) {
      if(mapFeatureGroup.getLayers().length > 0) {
        map.flyToBounds(mapFeatureGroup.getBounds().pad(0.15), {duration:
0.2});
      }
    }
  }

  $: if(map) {
    if(bounds) {
      map.fitBounds(bounds)
    } else {
      map.setView(view, zoom);
    }
  }
</script>

<div class="{classes}" use:createLeaflet>
  {#if map}
    <slot {map} />
  {/if}
</div>

<style>
  :global(.leaflet-control-container) {
    @apply static;
    @apply font-sans;
  }
  :global(.leaflet-container .leaflet-control-attribution) {
    height: 22px;
    opacity: 0.7;
    border-radius: 11px;
    background-color: #fff;
    margin: 10px;
    position: relative;
    top: -5px;
```

```
    font-size: 11px;
    font-weight: normal;
    font-stretch: normal;
    font-style: normal;
    line-height: normal;
    letter-spacing: -0.7px;
    text-align: right;
    padding-top:3px;
    color: #666;
    text-decoration: none;
  }
  :global(.leaflet-container .leaflet-popup-close-button) {
    @apply text-indigo-500;
    font-size: 25px;
    font-weight: 100;
    width: 32px;
    height: 32px;
    top: 12px;
    right: 4px;
    padding: 0;
  }

  :global(.leaflet-tooltip) {
    padding: 0!important;
    border: 0!important;
    background: 0!important;
    box-shadow: none!important;
  }

  :global(div.leaflet-bottom.leaflet-right div.leaflet-control-zoom) {
    width: 36px;
    height: 73px;
    border-radius: 5px;
    background-color: #fff;
    margin-right: 15px;
    margin-bottom: 12px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
  }
  :global(div.leaflet-bottom.leaflet-right div.leaflet-control-
zoom.leaflet-bar) {
    border: none;
  }
  :global(div.leaflet-bottom.leaflet-right div.leaflet-control-zoom .icon)
{
    font-size: 18px;
  }
  :global(div.leaflet-bottom.leaflet-right div.leaflet-control-zoom a) {
    display: flex;
    align-items: center;
    justify-content: space-around;
    width: 36px;
    height: 36px;
    line-height: 18px;
    font-size: 18px;
    @apply text-indigo-400;
```

```
  }
  /* fix to default leaflet behavior */
  :global(.leaflet-map-pane svg) {
    position: relative;
  }
  :global(.leaflet-map-pane svg.icon-badge) {
    transform: translate(-0.5em, -2.5em);
    border-radius: 50%;
  }
  :global(.leaflet-map-pane svg.icon-badge > .badge-circle) {
    stroke-width: 14%;
  }
  :global(div.leaflet-control-scale.leaflet-control) {
    margin-right: 15px;
    margin-bottom: 10px;
    cursor: grab;
  }
  :global(div.leaflet-control-scale-line) {
    text-align: right;
    cursor: grab;
    background: rgba(255, 255, 255, 0);
    border: none;
  }
  :global(.leaflet-control-scale::after) {
    content: '';
    display: block;
    border-bottom: 1px solid #888;
    border-left: 1px solid #888;
    border-right: 1px solid #888;
    height: 4px;
    background: none;
    cursor: grab;
  }
</style>
```

frontend/src/lib/Polyline.svelte

```
<script lang="ts">
  import L from 'leaflet?client';
  import {createEventDispatcher, getContext, setContext, onDestroy} from
'svelte';

  export let latLngs;
  export let options;

  const dispatch = createEventDispatcher();

  let featureGroup = getContext<() => L.FeatureGroup>('featureGroup')();

  export let line: L.Polyline = new L.Polyline(latLngs, options)
    .on('click', (e) => dispatch('click', e))
    .on('mouseover', (e) => dispatch('mouseover', e))
    .on('mouseout', (e) => dispatch('mouseout', e))
    .addTo(featureGroup);
```

```
  setContext('layer', () => line);

  onDestroy(() => {
    line.remove();
    line = undefined;
  });

  $: line.setStyle(options);

  $: {
    line.setLatLngs(latLngs);
    line.redraw();
  }
</script>

<slot />
```

frontend/src/lib/Button.svelte

```
<script lang="ts">
  import { createEventDispatcher } from "svelte";

  export let classes = '';
  export let wrapClasses = '';

  const dispatch = createEventDispatcher();

  function handleClick() {
    dispatch('click');
  }
</script>

<div class="relative {wrapClasses}">
  {#if $$slots.icon}
    <div class="pointer-events-none absolute inset-y-0 left-0 flex items-
center pl-2 text-gray-500">
      <slot name="icon"></slot>
    </div>
  {/if}
  <button on:click={handleClick} class="form-input px-2 py-1 {$$slots.icon
&& !classes.includes('pl-') ? 'pl-8' : ''} block rounded-md border-gray-
300 shadow-sm
  focus:border-indigo-300 focus:ring focus:ring-indigo-200 focus:ring-
opacity-50 hover:bg-indigo-50 transition-all {classes}">
    <slot></slot>
  </button>
</div>
```

frontend/src/lib/Popup.svelte

```
<script lang="ts">
  import L from 'leaflet?client';
  import { getContext } from 'svelte';

  let classNames: string | undefined = undefined;
```

```
  export { classNames as class };

  export let popup: L.Popup | undefined = undefined;

  let showContents = false;
  let popupOpen = false;

  const layer = getContext<() => L.Layer>('layer')();
  function createPopup(popupElement: HTMLElement) {
    popup = L.popup().setContent(popupElement);
    layer.bindPopup(popup);

    layer.on('popupopen', () => {
      popupOpen = true;
      showContents = true;
    });

    layer.on('popupclose', () => {
      popupOpen = false;
      // wait for the tooltip to completely fade out before destroying it,
otherwise the contents disappear too early.
      setTimeout(() => {
        if (!popupOpen) {
          showContents = false;
        }
      }, 500);
    });

    return {
      destroy() {
        if (popup) {
          layer.unbindPopup();
          popup.remove();
          popup = undefined;
        }
      },
    };
  }
</script>

<div class="hidden">
  <div use:createPopup class={classNames}>
    {#if showContents}
      <slot />
    {/if}
  </div>
</div>
```

frontend/src/lib/ItinerarySummary.svelte

```
<script lang="ts">
  import ItineraryLegs from '$lib/ItineraryLegs.svelte';
  import {timestampToString, secondsToDurationString} from '$lib/utils';

  import { fly } from 'svelte/transition'
```

```
  export let itinerary;
  export let totalFare;

  let firstTransitLegIndex = itinerary.legs.findIndex(leg ⇒
leg.transitLeg);
</script>

<div class="flex flex-col px-4 py-2 w-full rounded-md  border-gray-300
hover:bg-indigo-50 hover:bg-opacity-30 border shadow-sm hover:shadow-md
transition-all">
  <div class="flex flex-row text-lg justify-between">
    <span>{timestampToString(itinerary.startTime).time} -
{timestampToString(itinerary.endTime).time}</span>
    <span>{secondsToDurationString(itinerary.duration)}</span>
  </div>
  <div class="w-full">
    <ItineraryLegs itinerary={itinerary}/>
  </div>
  {#if firstTransitLegIndex ≠≡ -1}
  <div class="flex flex-row text-lg justify-between align-middle">
    <span class="text-sm text-gray-500 mt-1 text-left">
      Leaves at <span class="text-
black">{timestampToString(itinerary.legs[firstTransitLegIndex].from.depart
ure).time}</span> from <span class="text-
black">{itinerary.legs[firstTransitLegIndex].from.name}</span>
    </span>
    <div class="text-sm text-gray-500 mt-1 ml-2 text-left">
      {#if totalFare}
        <span transition:fly="{{x:25}}">{new Intl.NumberFormat('en-GB',{
style: 'currency', currency: 'GBP' }).format(totalFare)}</span>
      {/if}
      </div>
  </div>
  {/if}
</div>
```

frontend/src/lib/ItineraryLegs.svelte

```
<script lang="ts">
  import ItineraryLeg from '$lib/ItineraryLeg.svelte';

  export let itinerary;

  <!— calculates the width for each ‘bar’ based on its relative time,
whilst ensuring all bars have a width of at least 60 —>
  let totalWidth = itinerary.duration - itinerary.waitingTime;
  if (itinerary.legs[0].duration <= 60) {
    totalWidth -= 60;
  }

  let widths = itinerary.legs.map((leg) ⇒ {
    if (leg.duration <= 90) {
      return 0;
    } else {
```

```
      return (leg.duration / totalWidth) * 100;
    }
  });
</script>

<div class="flex flex-row flex-none gap-1 mt-1">
  {#each itinerary.legs as leg, i}
    {#if widths[i] > 0}
      <ItineraryLeg leg={leg} width={widths[i]} />
    {/if}
  {/each}
</div>
```

frontend/src/lib/ItineraryLeg.svelte

```
<script lang="ts">
  import {getContrastYIQ} from '$lib/utils';

  export let classes = '';
  export let width: number = 0;
  export let leg: {
    mode: string;
    colour: string;
    routeShortName: string;
    duration: number;
  }
  export let border = true;
  export let small = false;
</script>

<div class="{classes} rounded-md {small?'text-xs h-6 py-px':'text-sm h-8
py-1'} {width ≠ 0 ? (width < 7.5 ? 'px-1 place-items-center' : 'px-2') :
'px-1'} {border ? 'border-gray-300 border' : ''} shadow-sm transition-
colors duration-300
overflow-hidden whitespace-nowrap flex flex-row items-center flex-wrap
gap-[6px]
{leg.mode ≡ 'WALK' ? 'bg-gray-100' : 'bg-indigo-300'}
{leg.colour?(getContrastYIQ(leg.colour)=='black'?'text-black':'text-
white'):''}"
style="{width ≠ 0 ? 'width:'+width+'%' : 'width:max-content'};{leg.colour
? 'background-color:'+leg.colour : ''}">
  <div class="{small?'h-4 pt-[2px]':'h-6 pt-[3px]'} ">
    {#if leg.mode == "BUS"}
      <svg class={small?"w-3 h-3":"w-4 h-4"} fill="currentColor"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 512 512"><path d="M256
0C390.4 0 480 35.2 480 80V96l0 32c17.7 0 32 14.3 32 32v64c0 17.7-14.3 32-
32 32l0 160c0 17.7-14.3 32-32 32v32c0 17.7-14.3 32-32 32H384c-17.7 0-32-
14.3-32-32V448H160v32c0 17.7-14.3 32-32 32H96c-17.7 0-32-14.3-32-32l0-32c-
17.7 0-32-14.3-32-32l0-160c-17.7 0-32-14.3-32-32V160c0-17.7 14.3-32 32-
32h0V96h0V80C32 35.2 121.6 0 256 0zM96 160v96c0 17.7 14.3 32 32
32H240V128H128c-17.7 0-32 14.3-32 32zM272 288H384c17.7 0 32-14.3 32-
32V160c0-17.7-14.3-32-32-32H272V288zM112 400c17.7 0 32-14.3 32-32s-14.3-
32-32-32s-32 14.3-32 32s14.3 32 32 32zm288 0c17.7 0 32-14.3 32-32s-14.3-
32-32-32s-32 14.3-32 32s14.3 32 32 32zM352 80c0-8.8-7.2-16-16-16H176c-8.8
0-16 7.2-16 16s7.2 16 16 16H336c8.8 0 16-7.2 16-16z"/></svg>
```

```
    {:else if leg.mode == "WALK"}
      <svg class="w-3 h-4" fill="currentColor"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 320 512"><path d="M256
48c0 26.5-21.5 48-48 48s-48-21.5-48-48s21.5-48 48-48s48 21.5 48 48zM126.5
199.3c-1 .4-1.9 .8-2.9 1.2l-8 3.5c-16.4 7.3-29 21.2-34.7 38.2l-2.6 7.8c-
5.6 16.8-23.7 25.8-40.5 20.2s-25.8-23.7-20.2-40.5l2.6-7.8c11.4-34.1 36.6-
61.9 69.4-76.5l8-3.5c20.8-9.2 43.3-14 66.1-14c44.6 0 84.8 26.8 101.9
67.9L281 232.7l21.4 10.7c15.8 7.9 22.2 27.1 14.3 42.9s-27.1 22.2-42.9
14.3L247 287.3c-10.3-5.2-18.4-13.8-22.8-24.5l-9.6-23-19.3 65.5 49.5 54c5.4
5.9 9.2 13 11.2 20.8l23 92.1c4.3 17.1-6.1 34.5-23.3 38.8s-34.5-6.1-38.8-
23.3l-22-88.1-70.7-77.1c-14.8-16.1-20.3-38.6-14.7-59.7l16.9-63.5zM68.7
398l25-62.4c2.1 3 4.5 5.8 7 8.6l40.7 44.4-14.5 36.2c-2.4 6-6 11.5-10.6
16.1L54.6 502.6c-12.5 12.5-32.8 12.5-45.3 0s-12.5-32.8 0-45.3L68.7
398z"/></svg>
    {/if}
  </div>
  <div class="{small?'h-4':'h-6 pt-px'}">
    {#if leg.mode == "BUS"}
      {leg.routeShortName}
    {:else}
      {Math.round(leg.duration/60)}
    {/if}
  </div>
</div>
```

frontend/src/lib/utils.ts

```
// generate a path-dependent if we have VITE_API_URL defined (dev mode) or
not
export const apiUrl = (path: string) => `${import.meta.env.VITE_API_URL ||
''}${path}`;
export const otpBase = 'https://otp.nat.omg.lol/otp';

export function timestampToString(timestamp: number) {
  const dt = new Date(timestamp);
  const time = dt.toLocaleTimeString('en-GB', {timeStyle: 'short'});
  const date = dt.toLocaleDateString('sv-SE', {dateStyle: 'short'});
  return {time, date};
};

// only works up to 24 hours
export function secondsToDurationString(seconds) {
  const duration = new Date(seconds*1000)
  if (duration.getUTCHours() + (duration.getUTCMinutes()/60) >= 1) {
    return `${
      duration.getUTCHours()
    } hr${duration.getUTCHours()>1?'s':''} ${duration.getUTCMinutes()}
min${duration.getUTCMinutes()>1?'s':''}`;
  }
  if (duration.getUTCMinutes() < 1) {
    return `< 1 min`;
  }

  return `${duration.getUTCMinutes()}
min${duration.getUTCMinutes()>1?'s':''}`;
```

```
}

export async function getLineColour(agency: string, line: string) {
  const res = await fetch(apiUrl('/api/getcolour?')+ new URLSearchParams({
    agency: agency,
    line: line
  }));
  let col = await res.text();
  col = lightenDarkenColour(col, 20);
  return col
}

export function lightenDarkenColour(col, amt) {
  let useHash = false;
  if ( col[0] == "#" ) {
      col = col.slice(1);
      useHash = true;
  }

  const num = parseInt(col,16);

  let r = (num >> 16) + amt;

  if ( r > 255 ) r = 255;
  else if  (r < 0) r = 0;

  let b = ((num >> 8) & 0x00FF) + amt;

  if ( b > 255 ) b = 255;
  else if  (b < 0) b = 0;

  let g = (num & 0x0000FF) + amt;

  if ( g > 255 ) g = 255;
  else if  ( g < 0 ) g = 0;

  return (useHash?"#":"") + (g | (b << 8) | (r << 16)).toString(16);
}

export function getContrastYIQ(hexcol) {
  hexcol = hexcol.replace("#", "");
  const r = parseInt(hexcol.substr(0,2),16);
  const g = parseInt(hexcol.substr(2,2),16);
  const b = parseInt(hexcol.substr(4,2),16);
  const yiq = ((r*299)+(g*587)+(b*114))/1000;
  return (yiq >= 128) ? 'black' : 'white';
}

export function latLngString(latLng) {
  return latLng.lat.toFixed(5) + ',' + latLng.lng.toFixed(5);
}

export async function reverseGeocode(lat, lon) {
  const res = await fetch('https://photon.komoot.io/reverse?'+ new
URLSearchParams({
```

```
    lat: lat,
    lon: lon
  }));
  const data = await res.json();
  return data.features[0];
}

export function placeToName(place) {
  const properties = place.properties;
  return properties.name + (properties.osm_value≠'bus_stop'?
                    (properties.street? ', ' + properties.street : '')
                  + (properties.city? ', ' + properties.city : '')
                  + (properties.postcode? ', ' + properties.postcode : '')
                      : '' )
}

export async function searchPlaces(query, lat = null, lon = null, zoom =
null) {
  const res = await fetch('https://photon.komoot.io/api/?'+ new
URLSearchParams({
    q: query,
    bbox: '-7.58,49.96,1.69,58.64', // UK bounding box,
    ...(lat && lon && {lat: lat, lon: lon}),
    ...(zoom && {zoom: zoom})
  }));
  const data = await res.json();
  if (data.features.length == 0) {
    const re = /(-?\d{1,3}(?:\.\d+)?),\s*(-?\d{1,3}(?:\.\d+)?)/;
    if (re.test(query)) {
      const match = query.match(re);
      const reverse = reverseGeocode(match[1], match[2]);
      return [reverse];
    }
  }
  return data.features;
}
```

Testing

	Objective/Test	*Expected outcome*	*Actual outcome*
1.	User can enter a ‘from’ location for the journey.	‘From’ text field present which allows text to be entered.	As expected.
2.	The ‘from’ field can autocomplete entered locations from a geocoding API.	When typing in from field, suggestions are displayed as the user types.	As expected.
3.	User can enter a ‘to’ location for the journey.	‘To’ text field present which allows text to be entered.	As expected.
4.	The ‘from’ field can autocomplete entered locations from a geocoding API.	When typing in to field, suggestions are displayed as the user types.	As expected.

5.	User can select a date for the journey.	Selecting the current date opens a system date picker.	As expected.
6.	User can select a time for the journey.	Selecting the current time opens a system time picker	As expected.
7.	User can select whether the journey should leave at, or arrive by, the time selected.	Selecting the 'leave at' drop down allows the user to choose 'arrive by' instead.	As expected.
8.	User can click a button to initiate the journey search.	User can select 'plan journey' button, and search begins.	As expected.
9.	A list of possible journeys will be displayed to the user in a list.	List of journeys displayed.	As expected.
10.	In the list, a summary of each journey will be presented.	Summary card shown for each journey.	As expected.
11.	The summary will include the start and end times of the journey, and its length.	Times and length for each journey shown in card.	As expected.
12.	The summary will show the legs of the journey as proportional bars.	Legs of journey shown as bars proportional to their length.	As expected.
13.	The summary will show the total fare for that journey, if available.	Total fare for journey shown on journeys that contain fares.	As expected.
14.	The summary will show the live arrival time of the first transit leg in the journey, if available.		Not implemented.
15.	The user will be able to select a journey to see a detailed view of the legs of that journey.	Selecting a journey opens a details screen.	As expected.
16.	A list of possible fares will be shown for each transit leg in the journey.	Drop down selector shown alongside each transit leg in the details screen, containing possible fares.	As expected.
17.	There is a map for the user to plan and visualise journeys.	Map present on page.	As expected.
18.	If the user selects a point on the map, a popup is presented.	Selecting a point on the map displays a popup.	As expected.
19.	The popup contains the name of the location that was pressed.	Popup contains the location name.	As expected.

20.	The popup contains ‘from here’ and ‘to here’ buttons that can set the ‘from’ and ‘to’ fields.	Popup contains ‘from here’ and ‘to here’ buttons. Selecting them sets the ‘from’ or ‘to’ field to the selected location.	As expected.
21.	The selected journey from the results is drawn onto the map.	All legs of journey present on map.	As expected.
22.	Each leg of the journey is drawn as a line on the map.	Separate lines visible on map.	As expected.
23.	Walking legs will be drawn as a dotted line.	Dotted lines drawn on map to and from bus stops.	As expected.
24.	Transit legs will be drawn in the line’s colour, with the name of the line overlaid on it.	Transit legs coloured, e.g., 464 in purple, with ‘464’ label.	As expected.
25.	The map will zoom to fit the whole journey when it is selected.	Map automatically zooms to fit the whole journey after the search and when a new journey is selected.	As expected.
26.	The frontend will send a request to an OpenTripPlanner instance for a journey plan with the parameters in objectives 1, 3, 5, 6, and 7, when they select the button in objective 8.	Check network tab of web inspector, HTTP request sent to OpenTripPlanner instance.	As expected.
27.	The frontend will display the received journey plan in the format described between objectives 9 and 25.	See tests for objectives 9 and 25.	N/A
28.	The frontend will send the journey plan to the backend, requesting fares.	Check network tab of web inspector, HTTP request sent to `/api/getfares` endpoint.	As expected.
29.	The frontend will add the fare data to the already shown journey data once it is received.	Set browser to simulate slow network speed, total fare is added to summary a few seconds after it is first shown.	As expected.
30.	The backend will be able to download fare data from the Bus Open Data Service.	Program downloads configured datasets when started with the `-get-datasets` flag.	As expected.
31.	The fare data will be stored in a database.	Datasets added to configured database when program started with the `-add-datasets` flag.	As expected.

32. The backend will expose an API that can handle an OpenTripPlanner journey sent by the frontend.	Requesting `/api` returns 'busfares api', showing the backend is listening for requests.	As expected.
33. The backend will query the database and then filter the results to find all the possible fares for each leg of the journey.	Observe the following log entries for the backend: `query took [x]ms` `processed query in [x]ms` `found [x] fares for leg [x], filtered to [x].`	As expected.
34. The API will return the fares to the frontend once they are found.	Check for JSON response to `/api/getfares` request in network inspector.	As expected.
35. The backend will also expose an API to which returns the colours of different bus routes.	Check the `/api/colours` endpoint by requesting data, e.g., `/api/getcolour?agency=OP291&line=464`, where the response should be `#72177A`.	As expected.
36. The backend will store and retrieve the colour data of bus routes.	Check data in test above matches data in `colours.csv`.	As expected.

The following screenshots provide evidence for each of the tests:

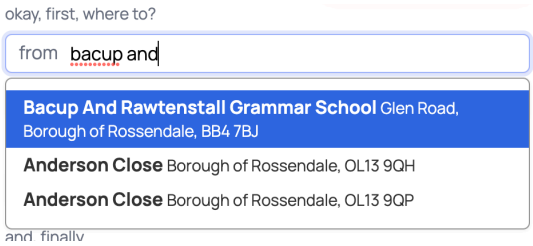


Figure 22. Tests 1 and 2.

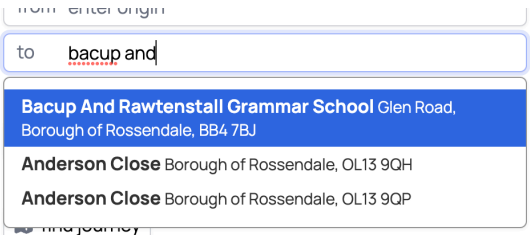


Figure 23. Tests 3 and 4.

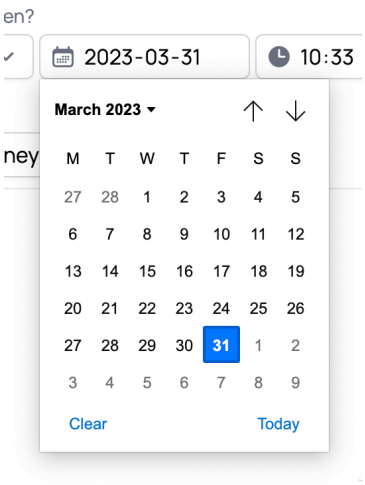


Figure 24. Test 5.

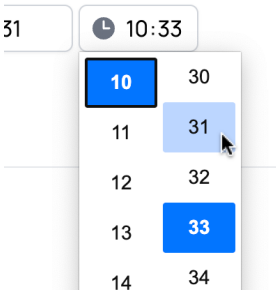


Figure 25. Test 6.

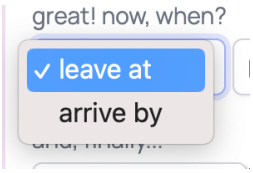


Figure 26. Test 7.

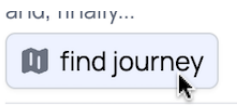


Figure 27. Test 8.

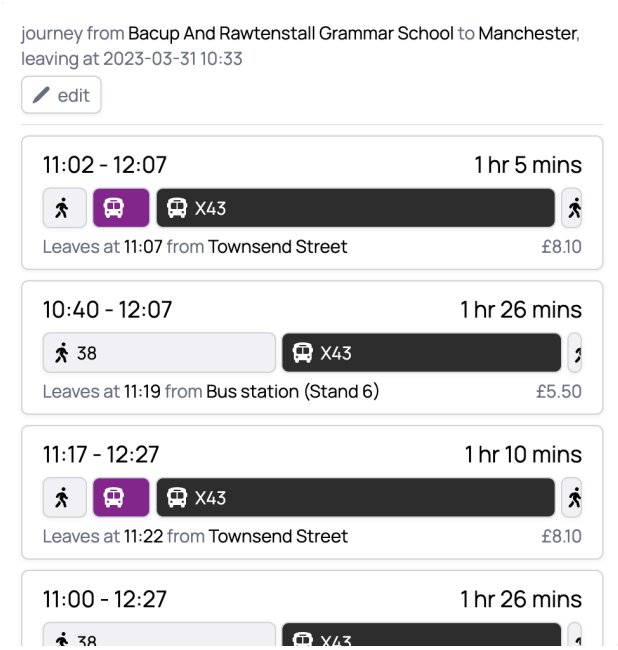


Figure 28. Tests 9–13.

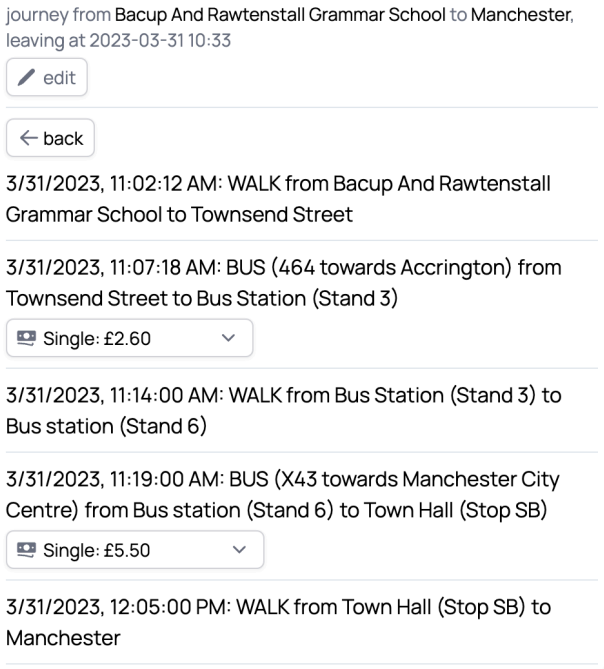


Figure 29. Test 15.

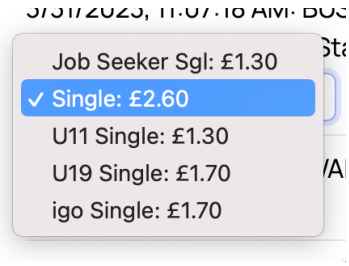


Figure 30. Test 16.

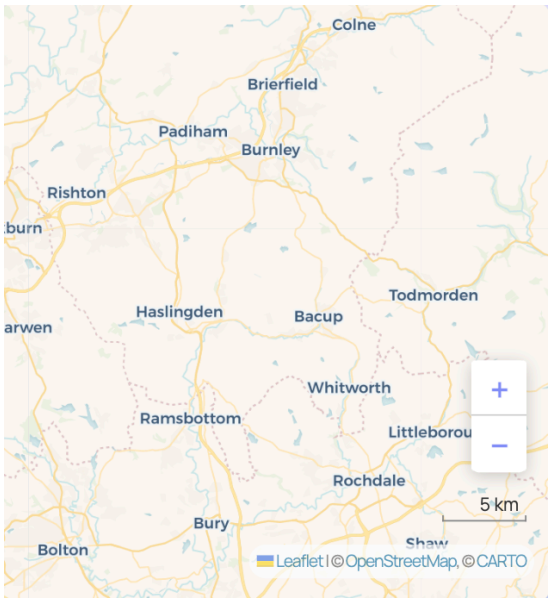


Figure 31. Test 17.

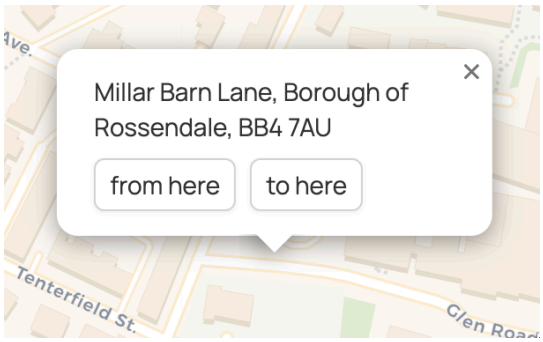


Figure 32. Tests 18–20.

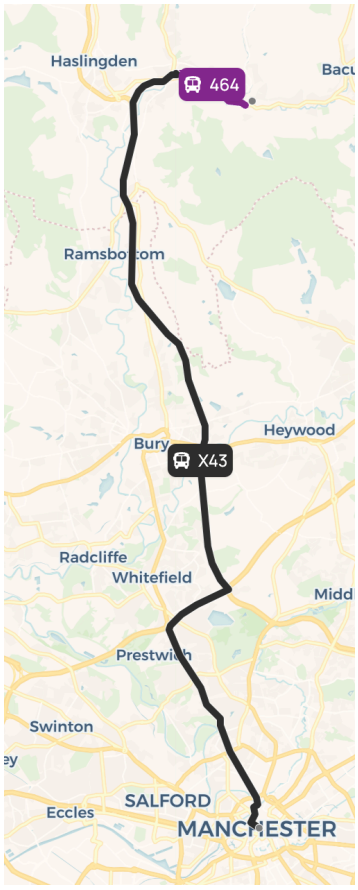


Figure 33. Test 21 and 25.

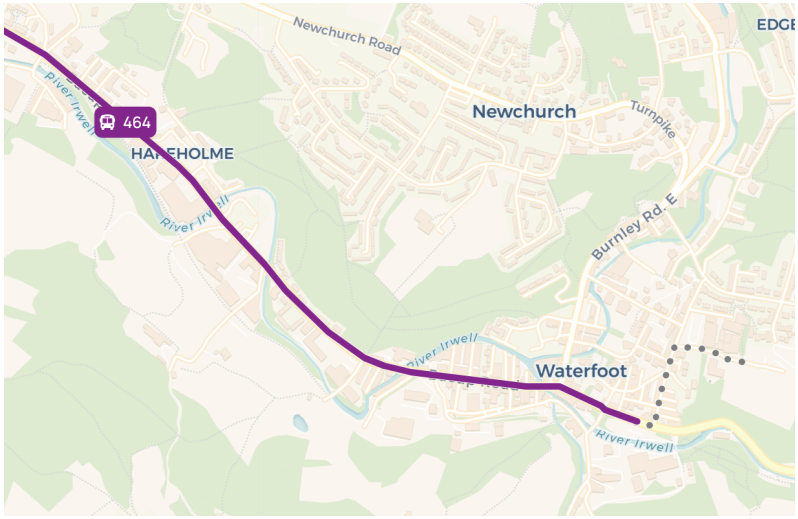


Figure 34. Tests 22–24.

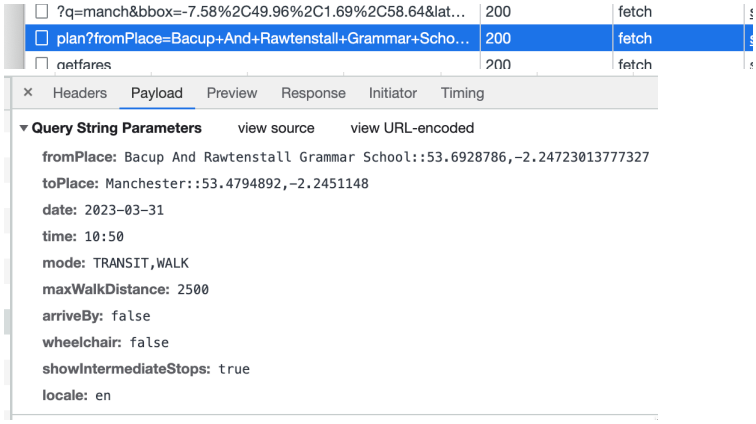


Figure 35. Test 26.

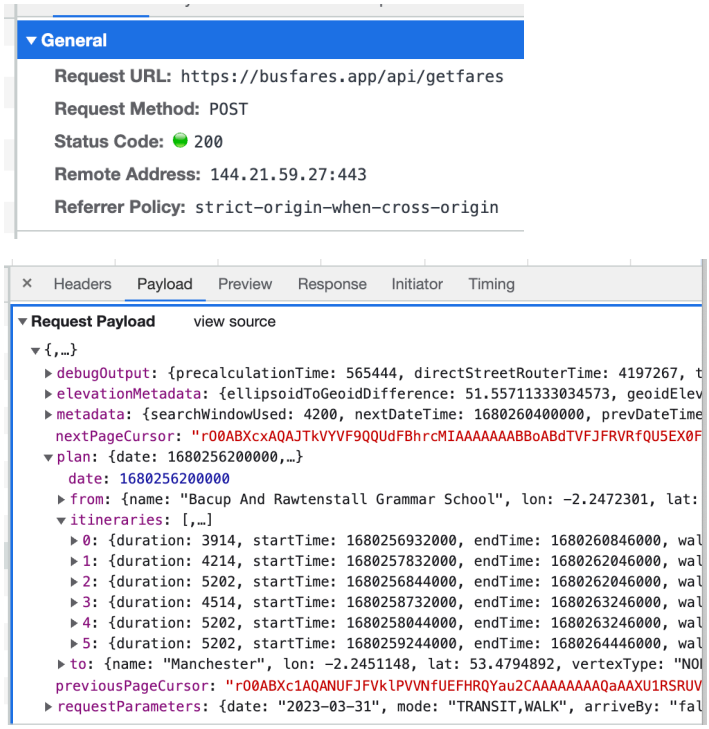


Figure 36. Test 28.

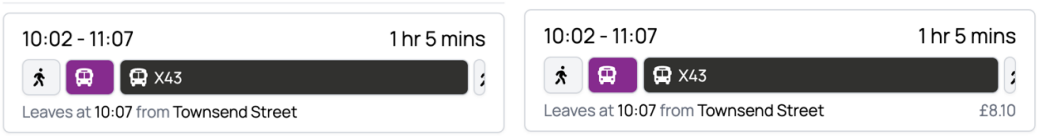


Figure 37. Test 29.

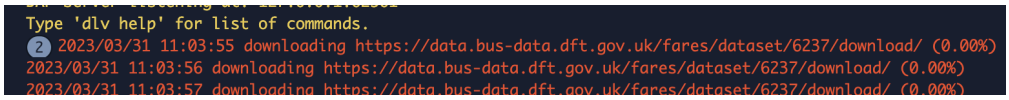


Figure 38. Test 30.

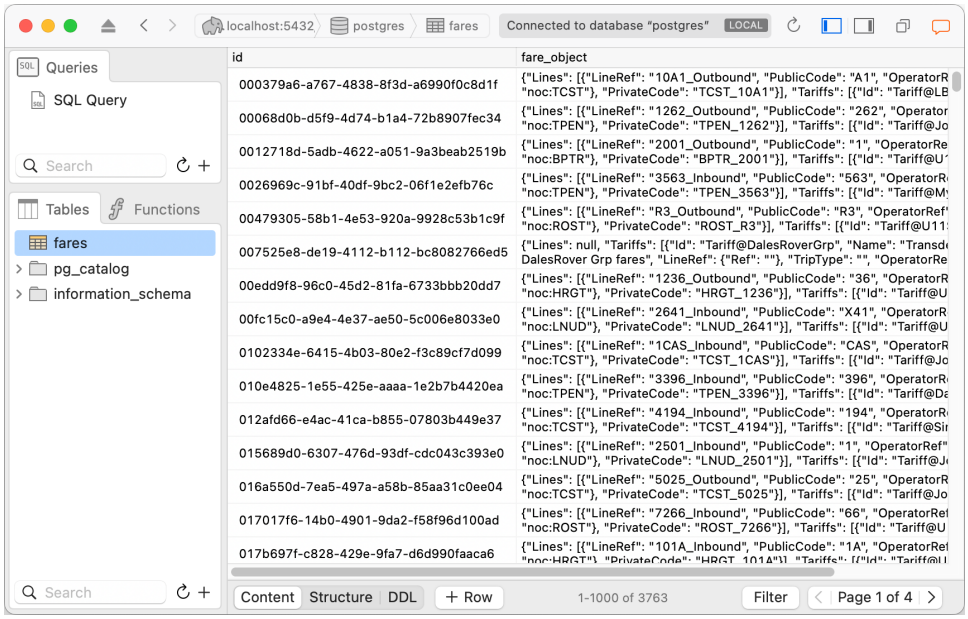


Figure 39. Test 31.

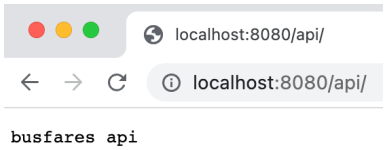


Figure 40. Test 32.

```
2023/03/16 14:49:41 query took 5.453474ms
2023/03/16 14:49:41 processed query in 88.780582ms
2023/03/16 14:49:41 found 9 fares for leg 464, filtered to 5.
2023/03/16 14:49:41 query took 8.460412ms
2023/03/16 14:49:41 processed query in 116.14187ms
2023/03/16 14:49:41 found 9 fares for leg 481, filtered to 5.
```

Figure 41. Test 33.

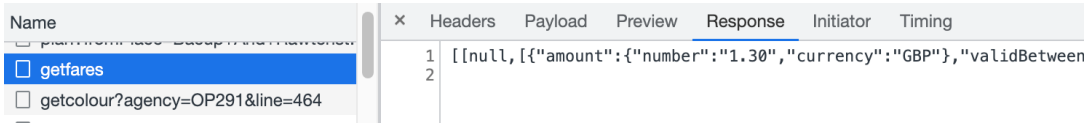


Figure 42. Test 34.

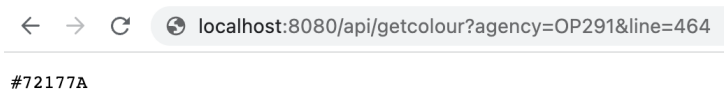


Figure 43. Test 35.

colours.csv

```
LNUD,153,#522583
LNUD,122,#2B3236
ROST,464,#72177A
ROST,X41,#C10E1A
ROST,467,#D3300F
```

Figure 44. Test 36.

Additionally, there is a video test, available here: https://youtube.com/watch?v=92bZ0xI9qww, which demonstrates the functionality of the system, and checks that the data provided is accurate in the real world. This video also covers the functionality described in tests 1, 2, 3, 6, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, and 29.

Conclusion

Ultimately, I believe that my solution has successfully met all but one of my original system objectives, and that I have built a system that follows the design well.

I conducted interviews with some of the respondents to my original survey, to gather user feedback on the final solution, and see how it could be improved further. All participants commented on the easy-to-understand design and overall ease of use of the system, so I believe that this reinforces that those system objectives were completed successfully. One participant particularly liked the ability to click on the map to choose a from and to location, finding it “very useful”. Participants liked the ability to see and choose between the possible fares for each journey, which wasn’t something they had seen before in a route planning system. This makes it clear that the system I have built does what it intended to; allowing users to plan journeys and be provided with useful fare information to help them make informed decisions about their travel.

Some participants suggested improvements that could further develop the system. One of these was to add a button that would switch the from and to locations. This would improve the usability of the system by allowing users to more easily plan journeys back to their starting point, and would be easy to implement. Another usability enhancement suggested was to improve the display of location search results on mobile devices, especially those with smaller screens, as they found it can be difficult to scroll through

multiple search results when there are enough to overflow the screen. Again, this would be a relatively simple improvement to add to the current system. A more complex idea that was suggested was to incentivise bus journeys by showing the user how much their carbon footprint/emissions would be reduced by, compared to taking the same journey by car. This would definitely be more of a 'nice to have' feature, as it doesn't directly affect the usability of the system, but I would definitely consider adding it in a future version of the system.

The one system objective that wasn't met was objective 14, which specified that the summary of a journey would show the live arrival time of the first transit leg in the journey, if it was available. I decided not to implement this, as after further research, I discovered that getting and processing the required live data to do it would be too difficult, as the data provided by the Bus Open Data Service only provides the position of vehicles, so these would need to be matched to journeys to calculate how long it would take them to reach the bus stop. I believe that this complexity puts the objective beyond the scope of this project, however it is still something that could potentially be added later in a future version of the system, alongside other improvements such as those suggested by users.

If I were to revisit the solution, I would also consider adding more ways to display fares, for example showing savings compared to different day or week passes, or adding the ability to passes that would be valid for trips that include multiple bus journeys. These features were beyond the scope of this project, but I would definitely consider building them in a future version, potentially conducting more user research to gauge demand for them.