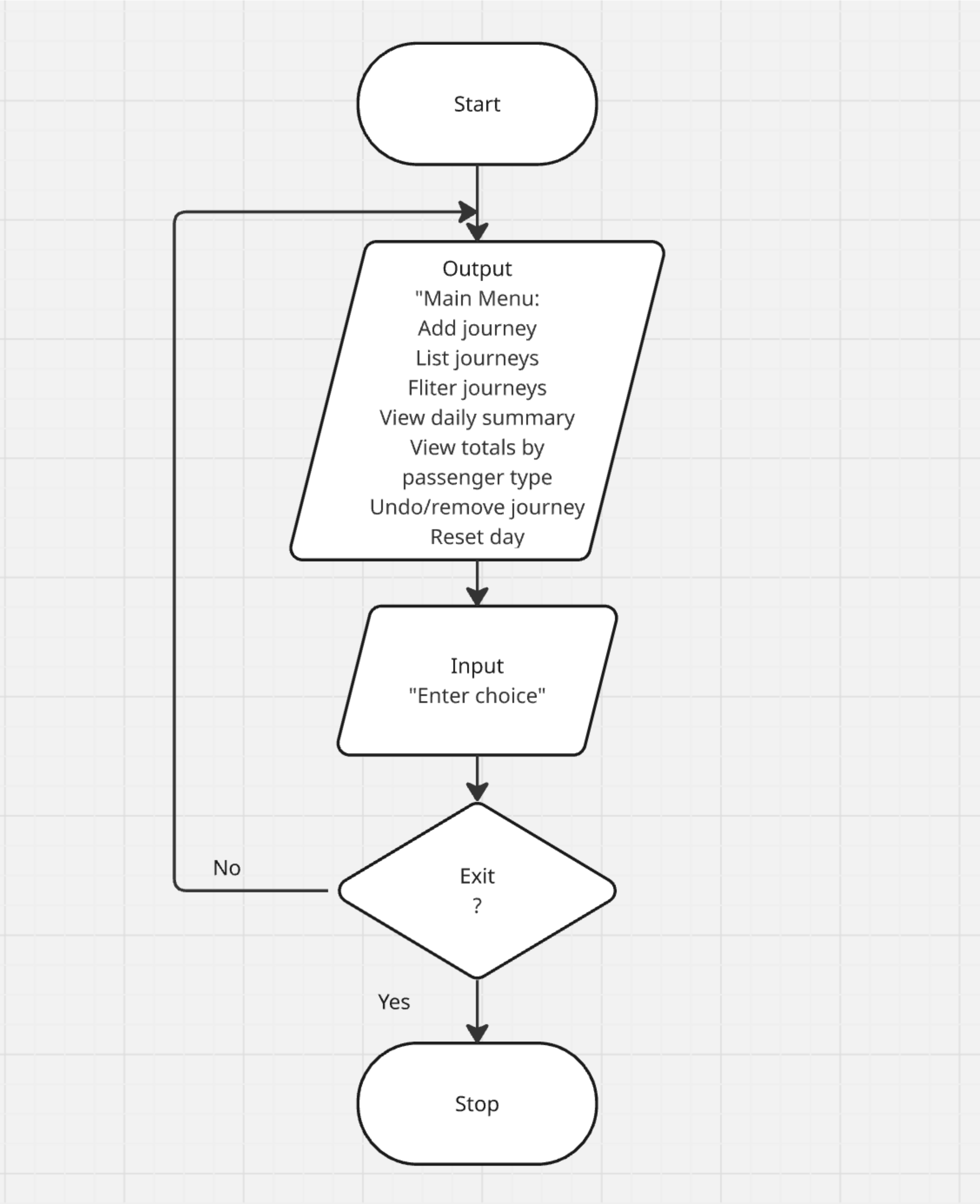
# IY4113 Milestone 2

Assessment Details	Please Complete All Details
Group	A
Module Title	Applied Software Engineering using Object Orientated Programming
Assessment Type	Java fundamentals
Module Tutor Name	Jonathan Shore
Student ID Number	P488018
Date of Submission	1/02/2025
Word Count	

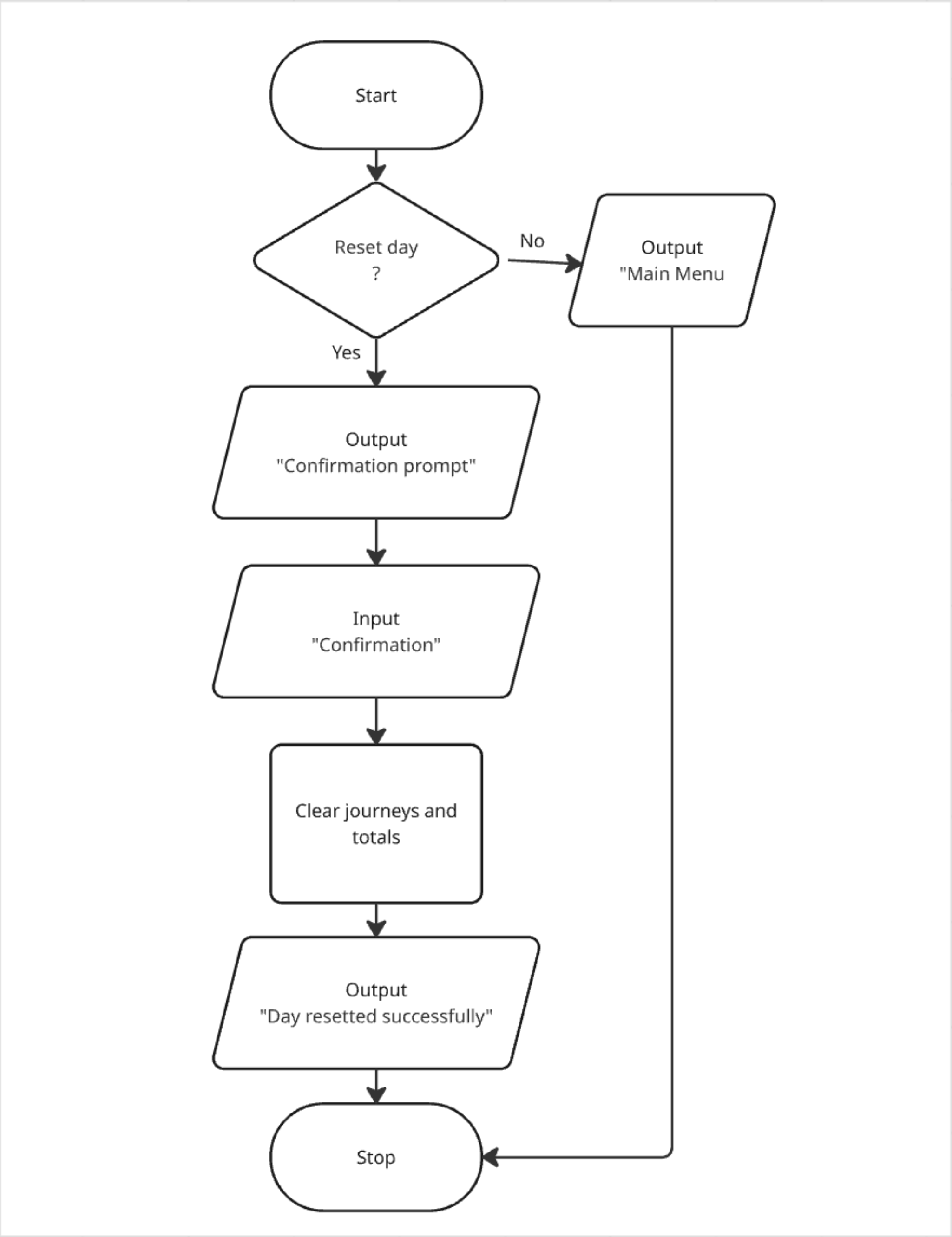
- ✓ I confirm that this assignment is my own work. Where I have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.*

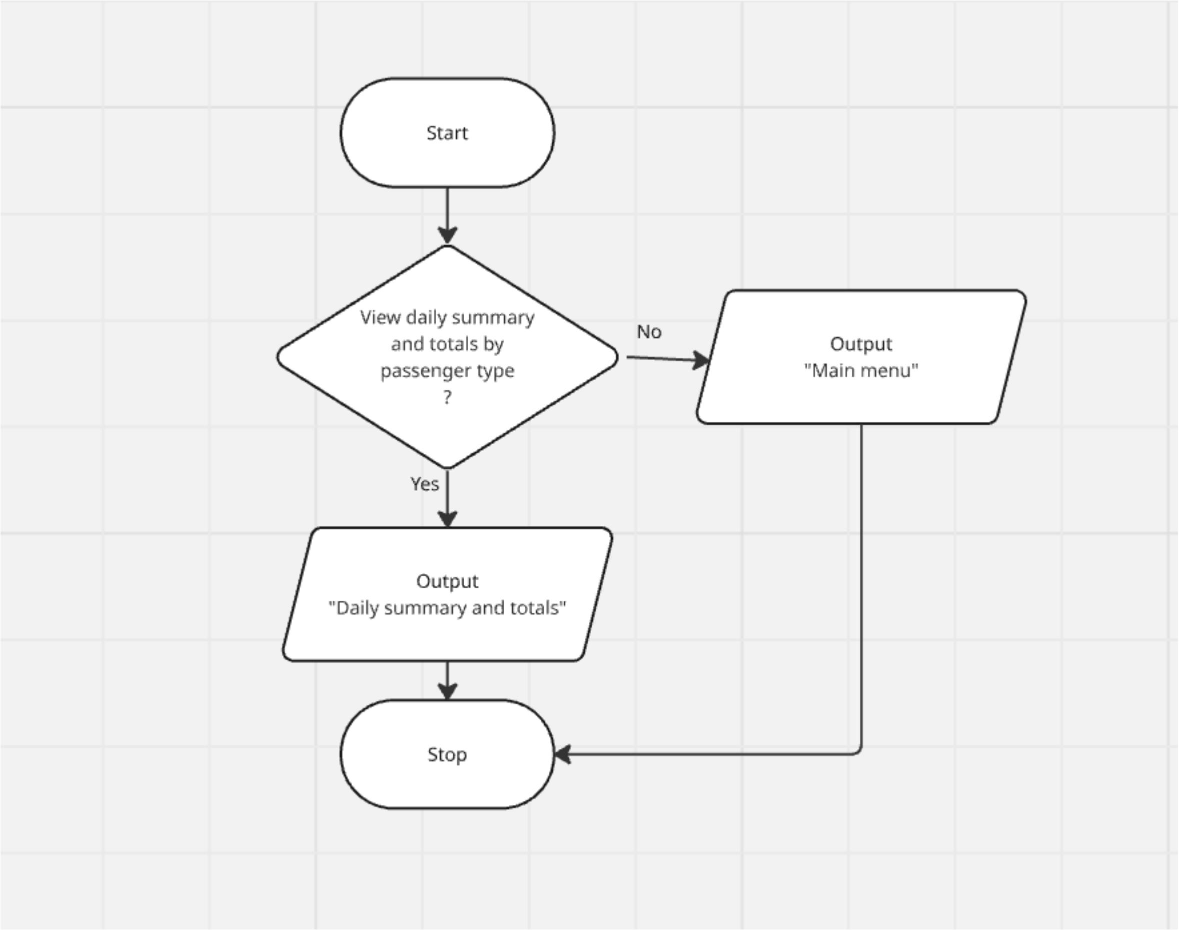
- ✓ Where I have used AI, I have cited and referenced appropriately.

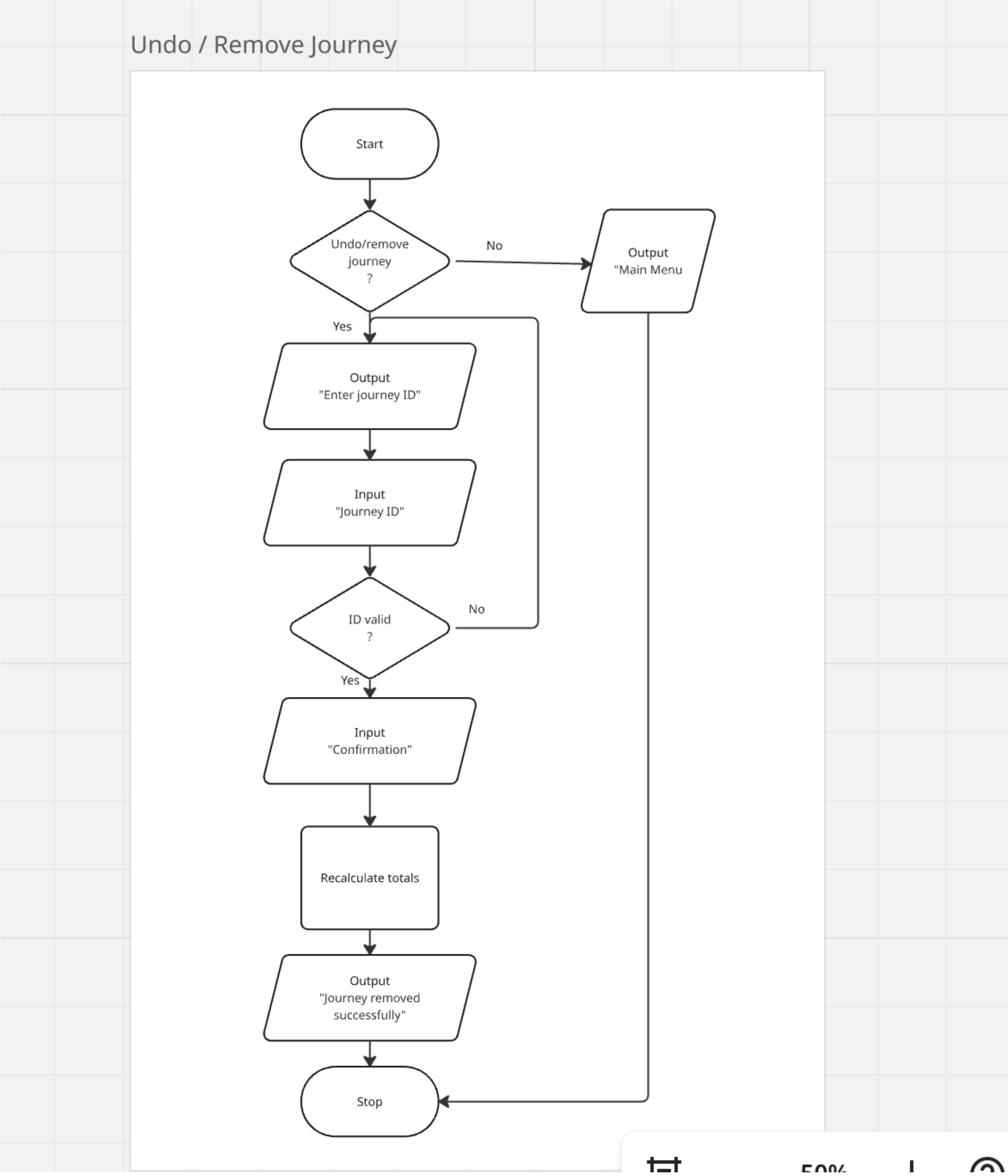
Algorithm Design


Start
Output
"Main Menu:
Add journey
List journeys
Fliter journeys
View daily summary
View totals by
passenger type
Undo/remove journey
Reset day
Input
"Enter choice"
Exit
?
No
Yes
Stop

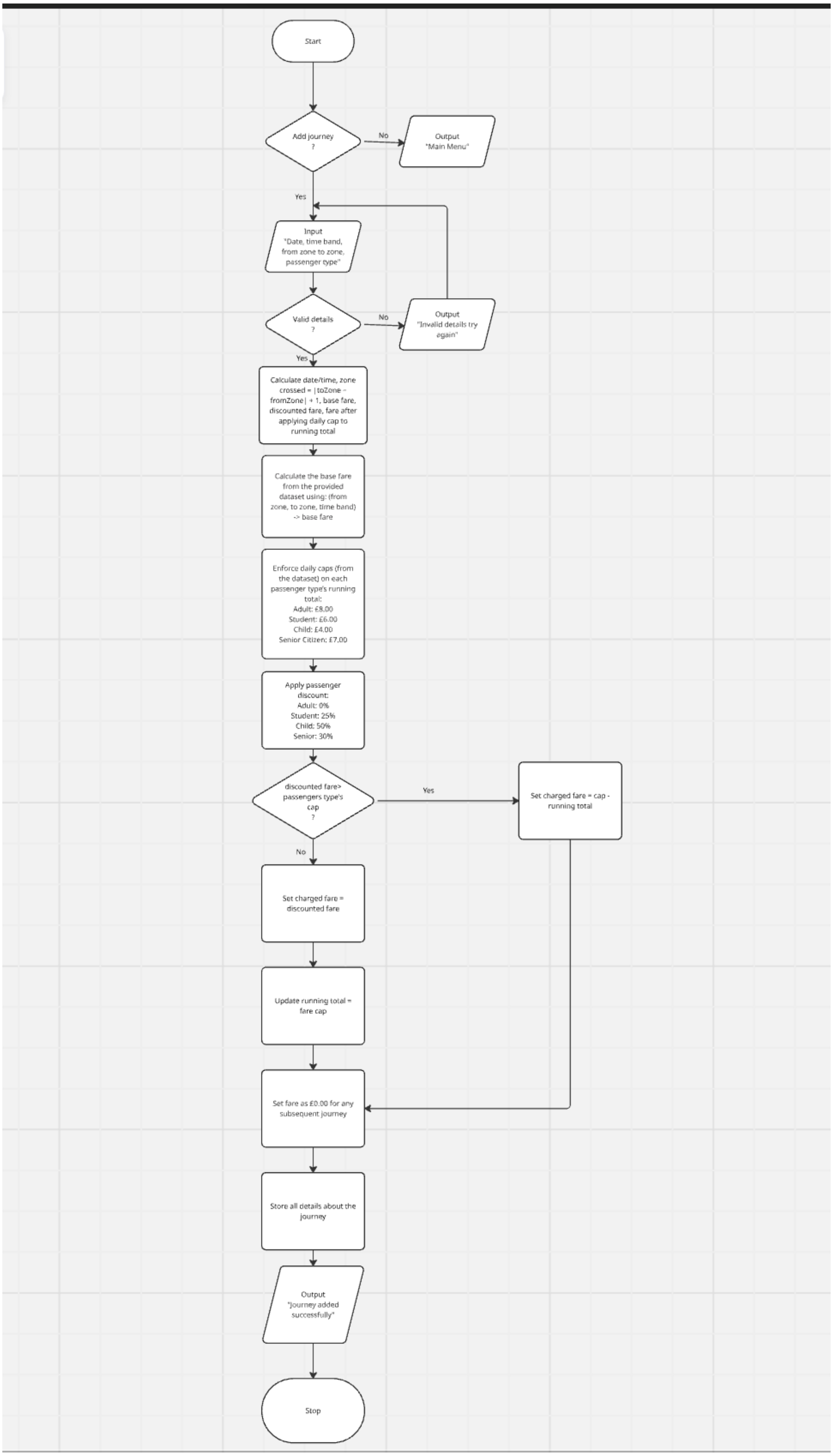
Reset day

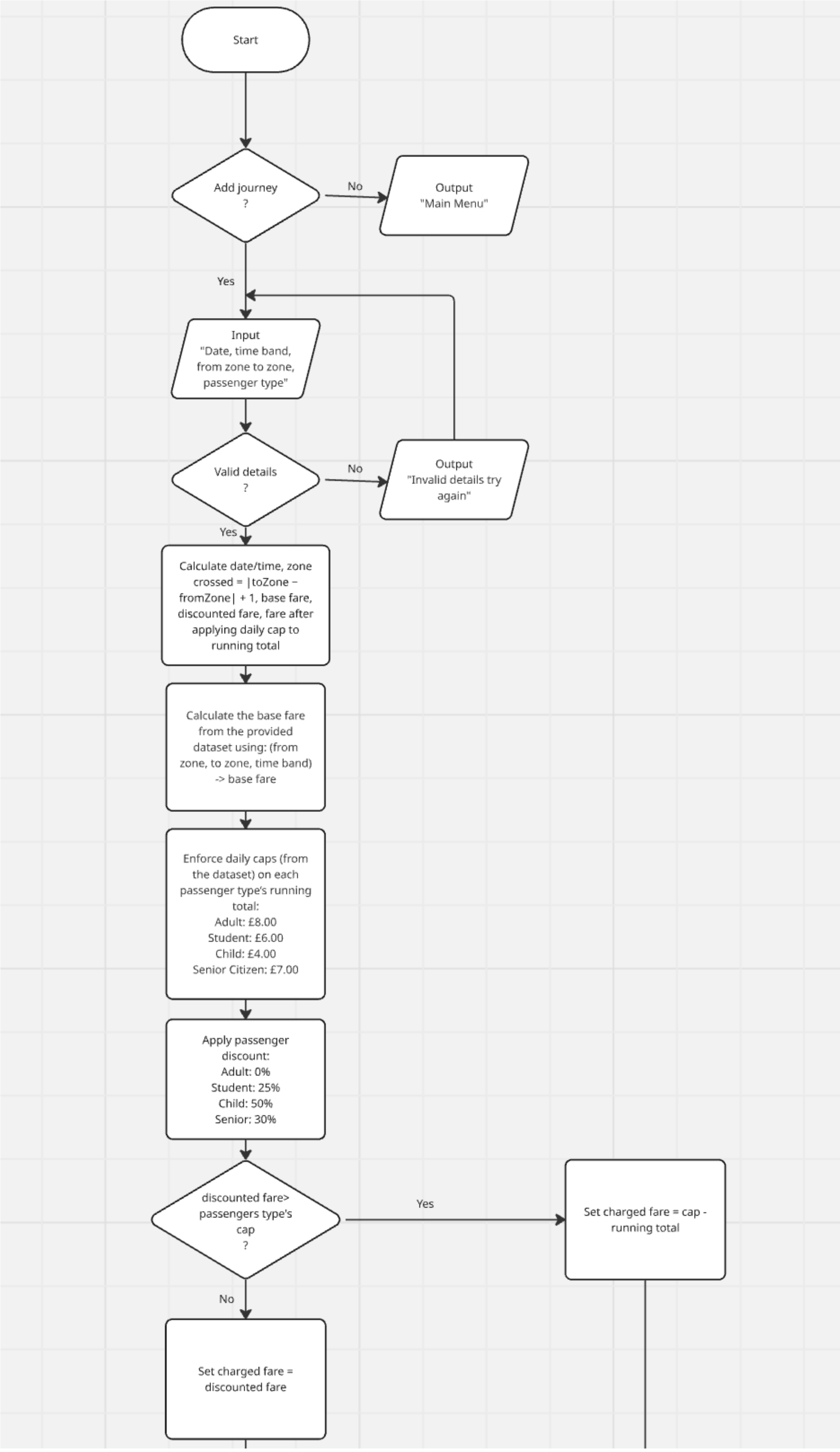

Start
Reset day ?
No
Output "Main Menu
Yes
Output "Confirmation prompt"
Input "Confirmation"
Clear journeys and totals
Output "Day resetted successfully"
Stop

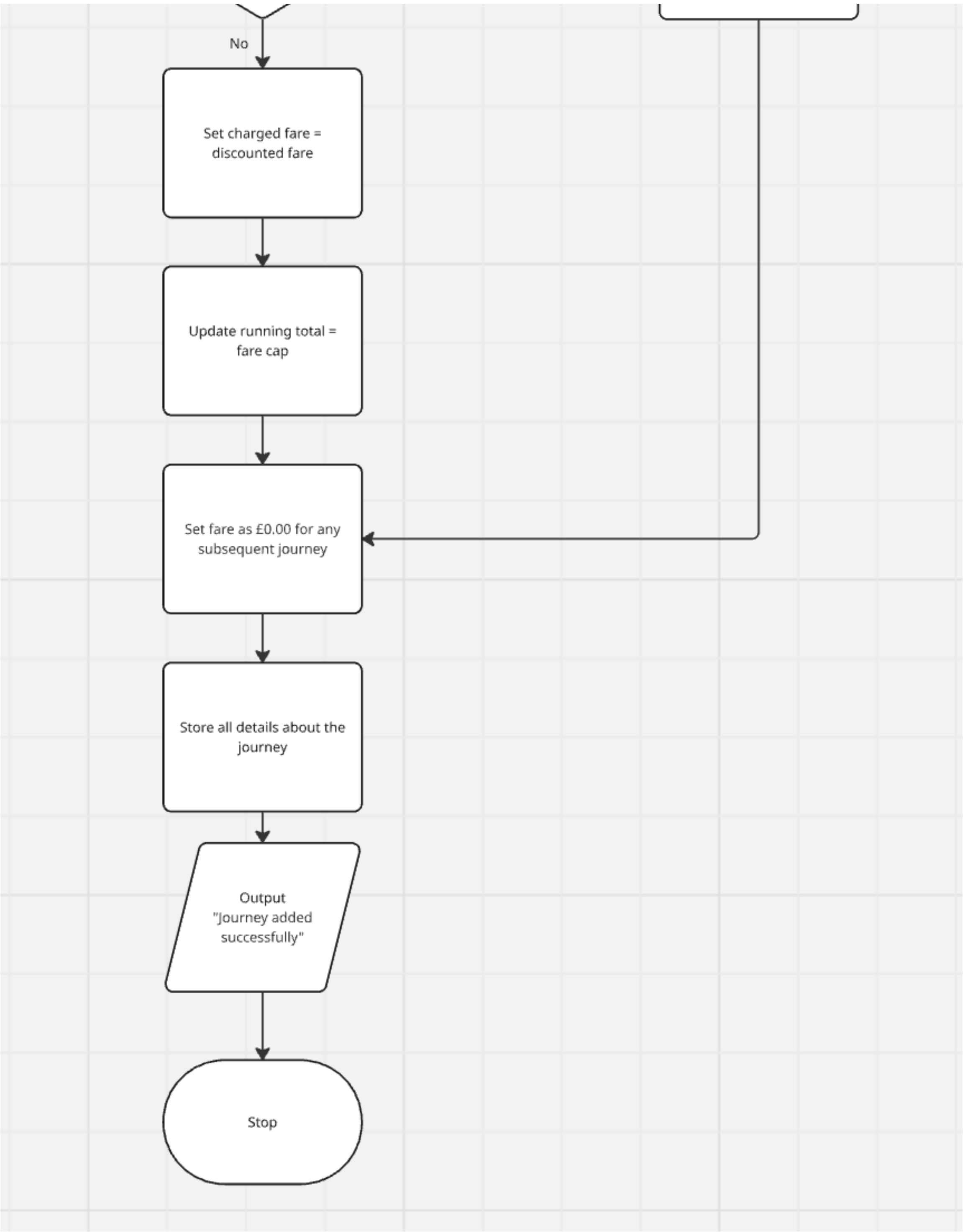

Start
View daily summary and totals by passenger type ?
No
Output "Main menu"
Yes
Output "Daily summary and totals"
Stop

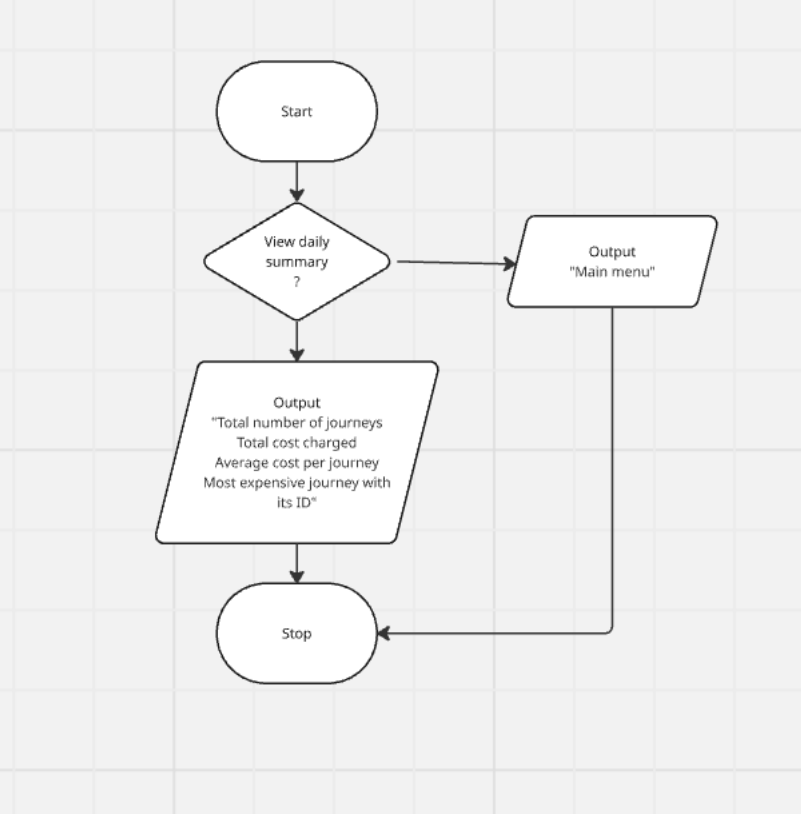


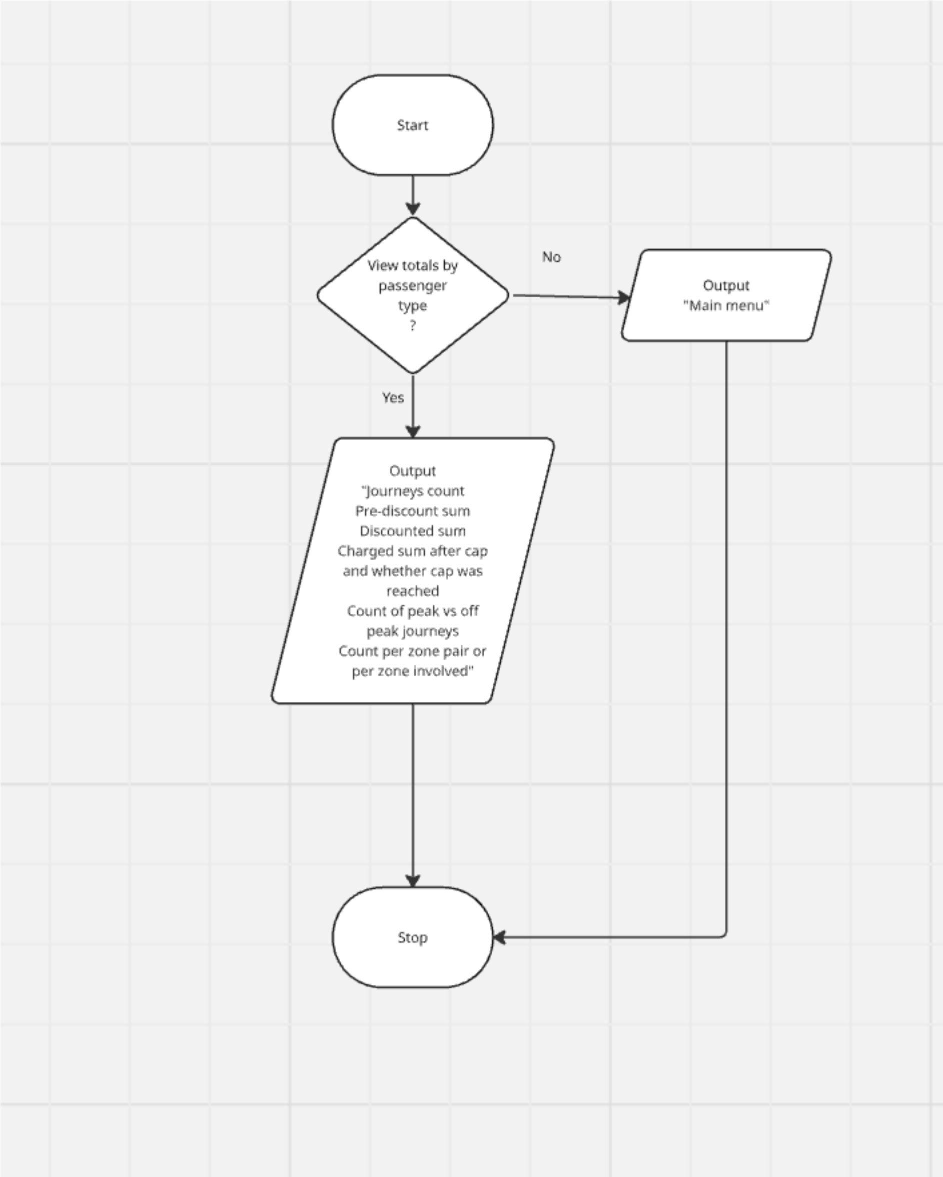
Undo / Remove Journey


Start
Add journey ?
No
Output "Main Menu"
Yes
Input "Date, time band, from zone to zone, passenger type"
Valid details ?
No
Output "Invalid details try again"
Yes
Calculate date/time, zone crossed = |toZone − fromZone| + 1, base fare, discounted fare, fare after applying daily cap to running total
Calculate the base fare from the provided dataset using: (from zone, to zone, time band) -> base fare
Enforce daily caps (from the dataset) on each passenger type's running total:
Adult: £8.00
Student: £6.00
Child: £4.00
Senior Citizen: £7.00
Apply passenger discount:
Adult: 0%
Student: 25%
Child: 50%
Senior: 30%
discounted fare> passengers type's cap ?
Yes
Set charged fare = cap - running total
No
Set charged fare = discounted fare
Update running total = fare cap
Set fare as £0.00 for any subsequent journey
Store all details about the journey
Output "Journey added successfully"
Stop


Start
Add journey ?
No
Output "Main Menu"
Yes
Input "Date, time band, from zone to zone, passenger type"
Valid details ?
No
Output "Invalid details try again"
Yes
Calculate date/time, zone crossed = |toZone − fromZone| + 1, base fare, discounted fare, fare after applying daily cap to running total
Calculate the base fare from the provided dataset using: (from zone, to zone, time band) -> base fare
Enforce daily caps (from the dataset) on each passenger type's running total:
Adult: £8.00
Student: £6.00
Child: £4.00
Senior Citizen: £7.00
Apply passenger discount:
Adult: 0%
Student: 25%
Child: 50%
Senior: 30%
discounted fare> passengers type's cap ?
Yes
Set charged fare = cap - running total
No
Set charged fare = discounted fare


No
Set charged fare = discounted fare
Update running total = fare cap
Set fare as £0.00 for any subsequent journey
Store all details about the journey
Output "Journey added successfully"
Stop


Start
View daily summary ?
Output "Main menu"
Output "Total number of journeys Total cost charged Average cost per journey Most expensive journey with its ID"
Stop


Start
View totals by passenger type ?
No
Output "Main menu"
Yes
Output "Journeys count Pre-discount sum Discounted sum Charged sum after cap and whether cap was reached Count of peak vs off peak journeys Count per zone pair or per zone involved"
Stop

class diagram

Research (minimum of 1 required, preferably 2)

Research existing programs that solve a similar problem. The program does not have to be written in java or object orientated in nature - just solve a similar type of problem.

Use the strucutre below to capture your evidence:

Name of program: Farebot

Reference (link): https://play.google.com/store/apps/details?id=com.codebutler.farebot

What it does well (2-3 features that work effectively):

The program is able to calculate and display the total amount of money spent on travel over a certain period of time which helps users see the overall travel costs

The program is able to show the user their journey history providing dates, routes and total costs

What it does poorly (at least 1 feature):

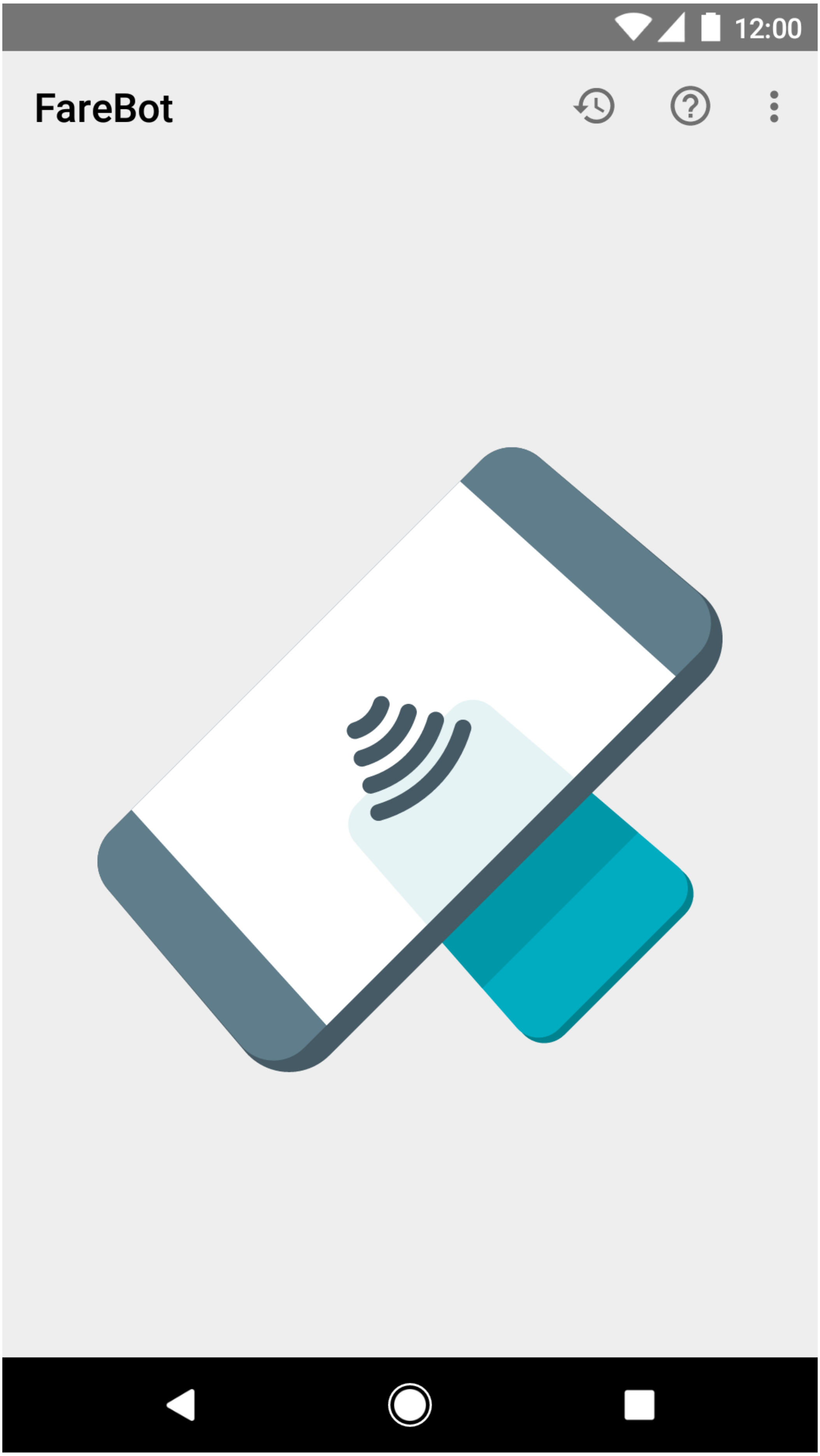
The program is quite outdated and not that easy to use as compared to modern transport apps

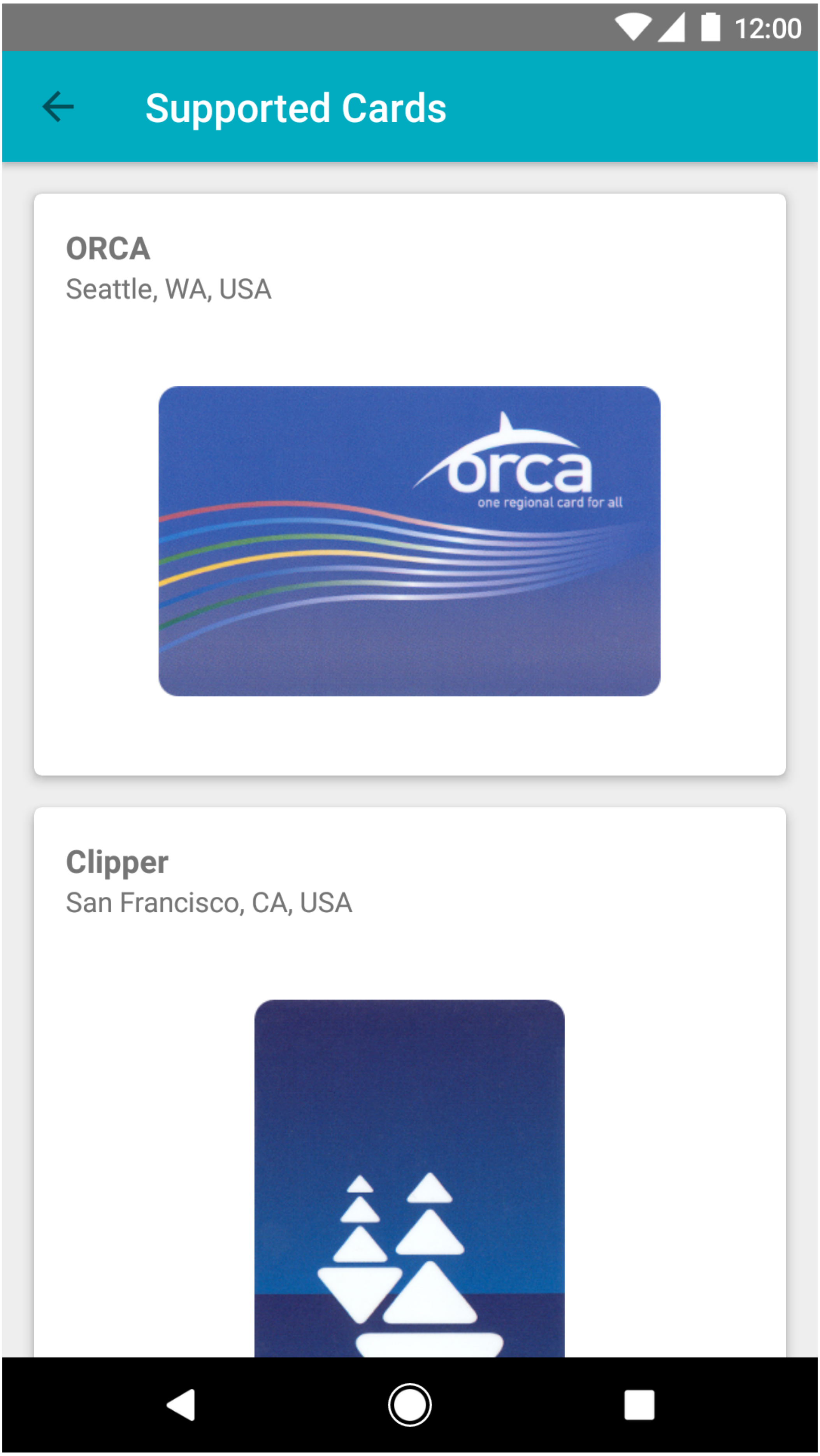
Key design ideas you could reuse (e.g., layout, navigation, input/output, program structure):

Storing journeys in a list with timestamps and costs which is similar to the journey list in the CityRide App

The program provides a summary of daily or session based travel expenses

Screenshot (showing the interface/output):


12:00
FareBot



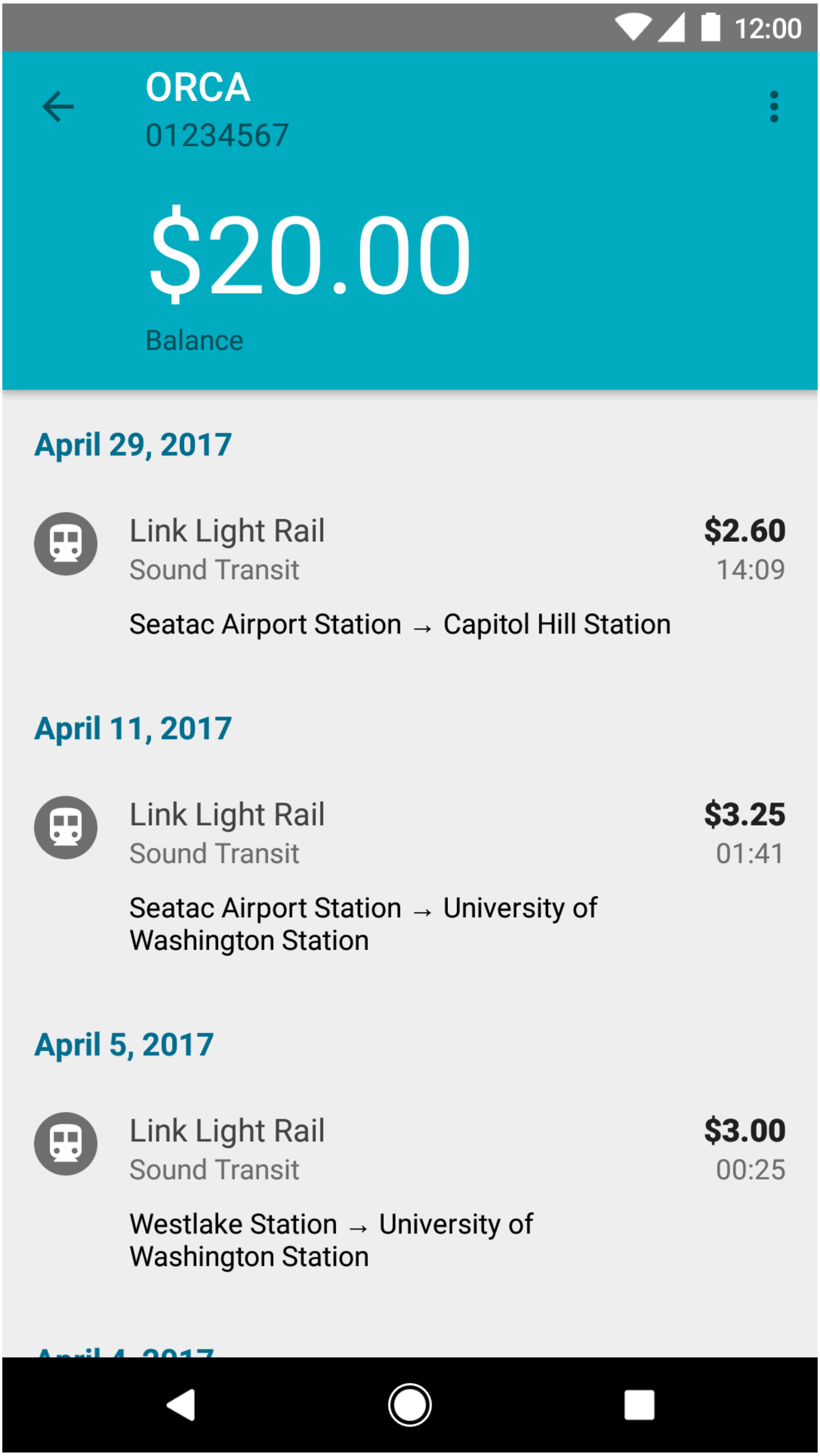
Supported Cards

ORCA

Seattle, WA, USA

Clipper

San Francisco, CA, USA



12:00

ORCA

01234567

$20.00

Balance

April 29, 2017

Link Light Rail — $2.60

Sound Transit — 14:09

Seatac Airport Station → Capitol Hill Station

April 11, 2017

Link Light Rail — $3.25

Sound Transit — 01:41

Seatac Airport Station → University of Washington Station

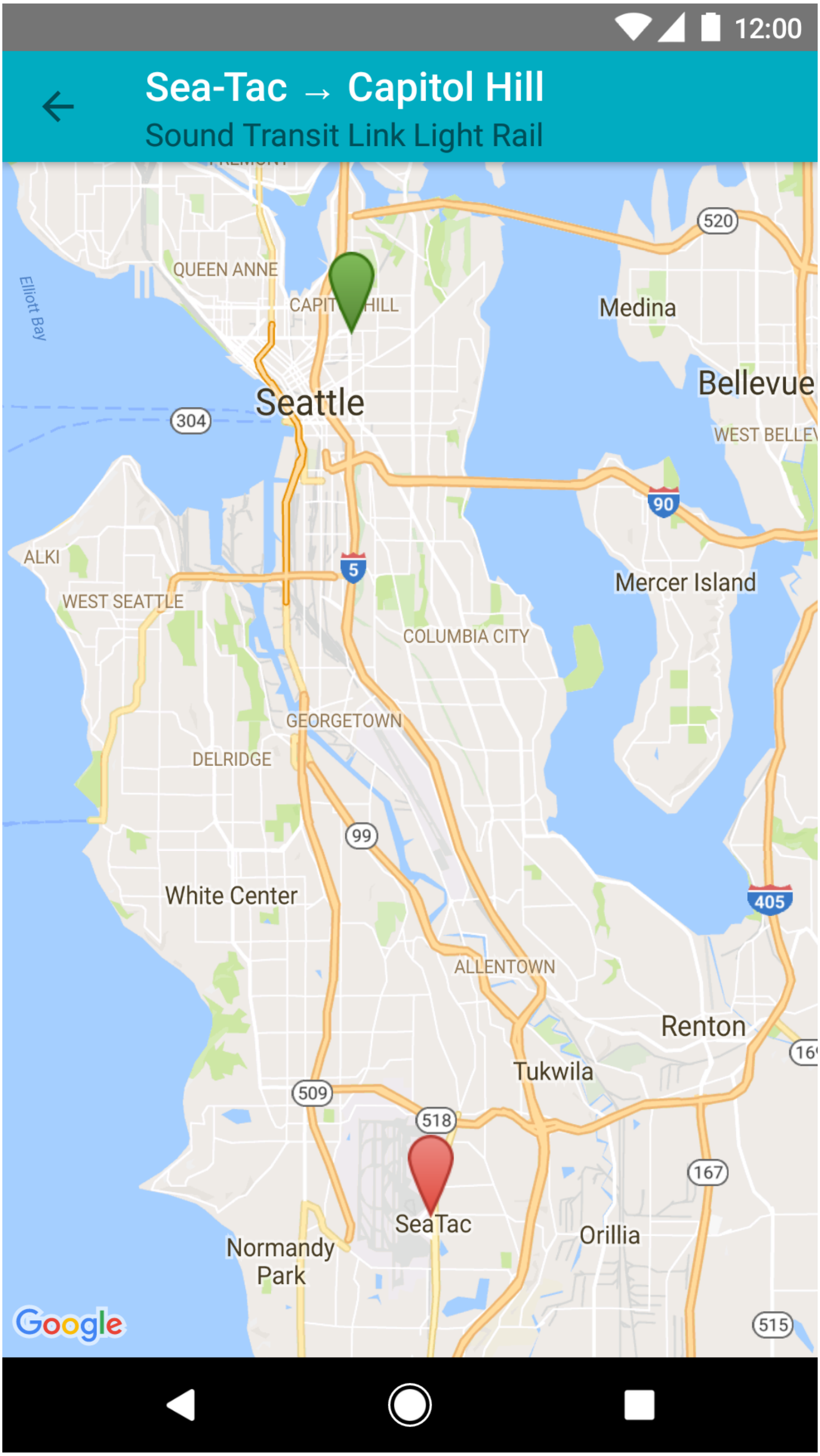
April 5, 2017

Link Light Rail — $3.00

Sound Transit — 00:25

Westlake Station → University of Washington Station

April 4, 2017



12:00

Sea-Tac → Capitol Hill

Sound Transit Link Light Rail

Name of program: Transit App

Reference (link): [Transit - the best app for buses and trains](https://transitapp.com/?utm_source=chatgpt.com)

What it does well (2-3 features that work effectively):

The program is able to display real time arrivals for both buses and trains

The program helps users to plan specific journeys with step by step directions until the destination

The program has a simple interface which provides quick and easy journey decisions

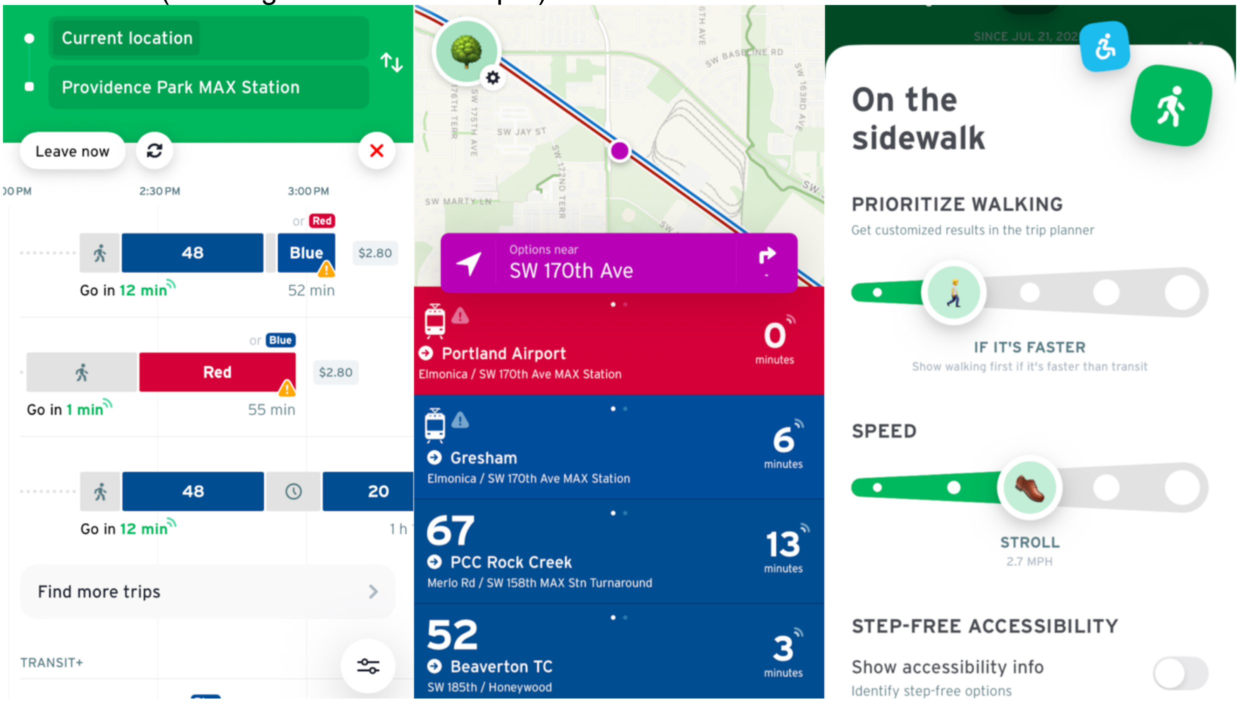
What it does poorly (at least 1 feature):

The program does not have detailed fare calculations and summaries of the totals

Key design ideas you could reuse (e.g., layout, navigation, input/output, program structure):

The program provides feedback after each user input making it interactive which is needed in the CityRide app

Screenshot (showing the interface/output):



Current location

Providence Park MAX Station

PRIORITIZE WALKING

Get customized results in the trip planner

IF IT'S FASTER

Show walking first if it's faster than transit

SPEED

STROLL

2.7 MPH

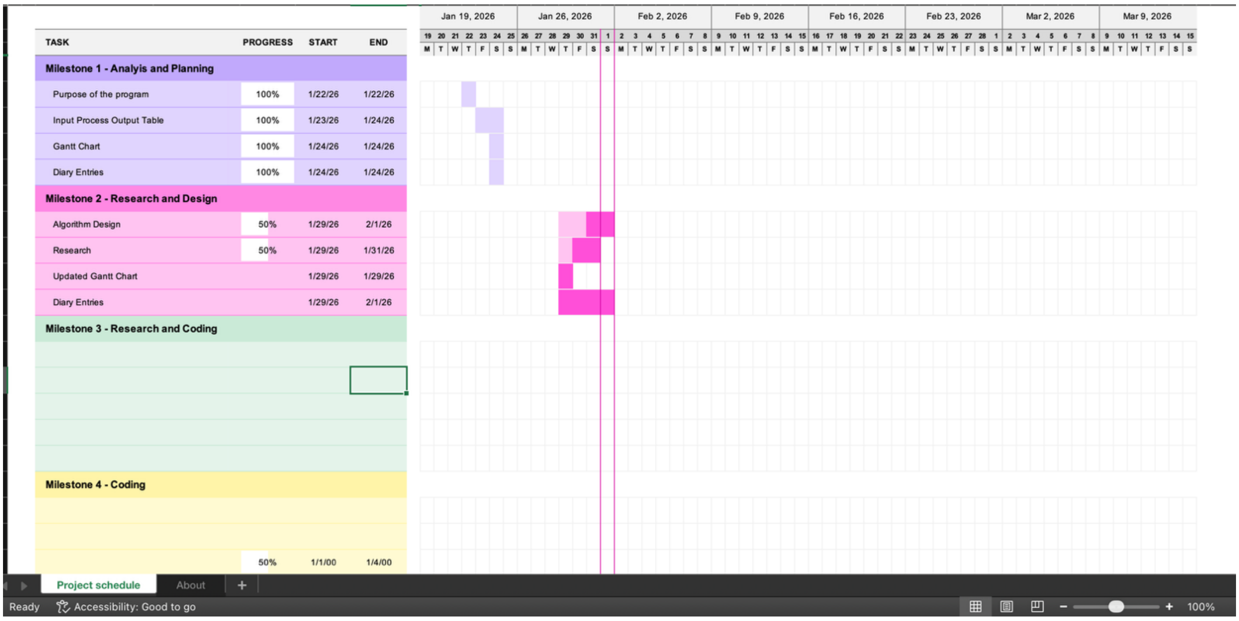
STEP-FREE ACCESSIBILITY

Show accessibility info

Identify step-free options

References
 Farebot Inc. (n.d.). Farebot – Public transport fare calculation and tracking app. Retrieved from: https:www.farebot.com/[Accessed 1 February 2026].
Transit App, Inc. (n.d). Transit – Real-time public transport navigation app. Retrieved from: https://transitapp.com/ [Accessed 1 February 2026].

Updated Gantt Chart



TASK	PROGRESS	START	END
Milestone 1 - Analyis and Planning			
Purpose of the program	100%	1/22/26	1/22/26
Input Process Output Table	100%	1/23/26	1/24/26
Gantt Chart	100%	1/24/26	1/24/26
Diary Entries	100%	1/24/26	1/24/26
Milestone 2 - Research and Design			
Algorithm Design	50%	1/29/26	2/1/26
Research	50%	1/29/26	1/31/26
Updated Gantt Chart		1/29/26	1/29/26
Diary Entries		1/29/26	2/1/26
Milestone 3 - Research and Coding			
Milestone 4 - Coding			
	50%	1/1/00	1/4/00

Diary Entries

Friday Researching Similar Programs

Date: Friday, 30 January 2026

What I did:
I researched existing programs that solve a similar problem to the CityRide app system. I focused on finding an application that tracks journeys and calculates travel costs. I chose Farebot and Transit App which i got of a google search on apps that help users access transport. This were my main examples and reviewed information from its Google Play Store page. I looked at how they both keep journey history, display routes and dates, and calculate total travel costs over a period of time.

Why I did it:
This research was carried out to understand how real world systems manage travel fare data and present it to users. By analysing Farebot and Transit App, I was able to see how journey information can be stored in a structured way and how summaries can help users understand their spending. This helped me identify useful features that could be adapted for my own project such as storing journeys in a list and showing daily or session based totals.

Problems encountered:

One problem was that Farebot is quite outdated, meaning its interface is not very modern. This made it harder to compare it directly with newer applications. However, this also helped me focus more on functionality rather than design, which is more relevant to my coursework.

On the other hand the Transit App was quite modern and is the better one in this case

---

Saturday Analysing Design Ideas and Updating Plans

Date:Saturday, 31 February 2026

What I did:
I looked at Farebot and Transit App in detail and found key design ideas that could be reused in the CityRide app. I focused on how journeys are stored with timestamps and costs, and how the app provides summaries of travel expenses. I then documented what they do well and what they do poorly linking these observations to how I plan to design the CityRide system. I also began reviewing my project plan and checked which tasks had been completed so far.

Why I did it:
This step was important to ensure that my project design is informed by existing solutions. By evaluating both the strengths and weaknesses of the apps, I could make decisions about what features to include and what to avoid. Reviewing my progress also helped me prepare to update my Gantt chart.

Problems encountered:
It was sometimes difficult to decide which features were relevant to my project and which were outside the scope of the coursework. I resolved this by focusing only on features related to journey tracking, fare calculation, and summary outputs, which directly match the CityRide requirements.

Sunday Updating the Gantt Chart and Reflecting on Progress

Date: Sunday, 1 February 2026

What I did:
I updated my Gantt chart to show the current status of the project. This included marking the research phase as complete, confirming that the algorithm design using flowcharts had been finished and adjusting future tasks such as implementation and testing. I reviewed the entire project time to ensure it has clearly showed progress over time.

Why I did it:
Updating the Gantt chart was necessary to demonstrate good project management and organisation. It helps show that enough time is given for each part. This also allowed me to look at how the research and planning stages support the later development of the program.

Problems encountered:
One issue was deciding how much time to allocate to each remaining stage of the project. I fixed this by checking the complexity of each task and allowing extra time for testing and debugging, which are often more time consuming than expected.
