Abstract

This project entails creation of a journey transport application that is aimed at helping commuters in Cape Town in planning their trip on a variety of transport services that include MyCiTi, Metrorail and Golden Arrow. The program combines real-time information of such services to offer users the most effective paths, depending on their personal preferences, including the minimization of walking, the number of transfers, or the speed. The optimization of routes is made possible by the introduction of the algorithm of Dijkstra and RAPTOR algorithm which makes the routes very accurate and flexible in the process of planning the trips. The final output is a web application which has been developed using Django as the backend services and react as the front-end interface. This mixture will allow secure user management, route saving, and a responsive and intuitive user interface. The development was well structured with a first documentation approach whereby requirements and system specifications were developed to completion before being coded. During the development, a horizontal prototype of the user interface was developed to investigate the design options and to test the usability but was discarded and replaced with the final implementation. This brought out clarity, less ambiguity, and led to a solution that does not only satisfy the client but also improves the overall commuting experience to users of the public transport in Cape Town.

Introduction

Scope and Stakeholders

This project is meant to develop and design a public journey transport application that assists commuters to manoeuvre the transport system in the city in a convenient and easy manner. The application combines real time information on the major transport services accessible such as MyCiTi, Metrorail and Golden Arrow to give precise and updated journey plans. The user can not only find the quickest ways but also the ones that suit the user best, e.g. the ones that minimise the walking time, require minimum transfers, or are at convenient times. The application is reliable and flexible in planning journeys by incorporating the Dijkstra algorithm (optimal route) to achieve the shortest path with the RAPTOR algorithm (efficient routing of public transit). The result is a web-based platform, built using Django as the backend and React as the frontend. Django allows secure login, account management, and database management and React serves as an interactive and responsive user interface. When utilized together, these technologies ensure that customers can easily plan their journeys, register accounts, and store their most frequently used routes.

The stakeholders of the project are

Members of the team: Evian McKeown, Shaylen Naidoo, Benji Joss.

The clients: Professor Jan Buy

The target market:

1. Cape Town commuters that use transportation services regularly and require their routes to be planned faster and more efficiently.

2. New or infrequent users of the public transport that require a clear and easily accessible guide on how to manoeuvre through the system.
3. Tourists and visitors that might be strangers to the Cape Town public transport system.
4. A transport service provider would gain advantages as one of the beneficiaries of the better commuter planning tools.
5. City planners and researchers of urban mobility looking at commuter decision trends.

The possible influence and worth of such applications are not limited to making traveling easier. Since public transportation is now more accessible, it encourages more people to use it, which results in sustainability, less traffic, and eco-friendly transportation options. By providing users with an interactive and adaptable travel planner that best suits their requirements and tastes, the app makes commuting more effective, enjoyable, and useful.

Approach

The implementation of the project is a web application using the full-stack version, where the backend and frontend tasks are very separate. The reason why the Django framework was selected as the backend is due to its strength, security and the fact that it supports intricate interaction with the database. It deals with user accounts and the authentication as well as save route. The frontend was chosen as a React because of providing a dynamic and user-friendly experience that supports easy interaction and real-time responsiveness to user input. The routing capability is achieved by the Dijkstra and RAPTOR algorithm and by the utilization of real-time public transport data feeds.

This project was plan-based with documentation first, unlike agile development. All the documentation such as requirements, system specifications and design models were done prior to the actual coding. This helped keep the team on track and reduced and eradicated confusion and ambiguity in implementation. On the way, a horizontal throw away prototype of the user interface was created. This prototype gave the team an opportunity to experiment with various layouts of the interface, confirm that the usability is correct, and receive early feedback, but it was not a component of the final system and was eventually discarded. The coding process was then linear, and modules implemented based on the requirements that were documented. This methodology offered great form, consistency to the original design and facilitated the provision of a consistent product.

Requirements Captured

Client meetings and the continuation of the communication channel were conducted to achieve the requirements of the application. These requirements were used as the basis of determining the main functions of the system which were converted into a set of use cases. The use cases have been based on key functionalities like journey planning, route optimization, user account control and the capability to save frequently used routes where the final design would cater to the expectations of the client and the needs of the end-users.

Functional Requirements

1) The system will enable users to key in an origin and destination and show available public transport routes within 2 seconds of user input.
2) The users will be able to input a travel time and / or a date; and the system will provide time specific route choices in less than 3 seconds.
3) The system will create route suggestions based on at least two routing algorithms (Dijkstra and RAPTOR) and compare those algorithms to each other in each query.
4) The suggested routes will introduce a minimum of three modes of transportation (bus, train, minibus taxi) and present it uniformly in all the generated results.
5) The user will be able to see a map with segments of the routes drawn on it in relation to each of the suggested paths and within a time less than 2 seconds, the map will be rendered.
6) The user will be able to alternate between various suggested routes and see the relative information (ETA, distance, cost, and transfers) in real-time (less than 1 second delay).
7) The users will be able to filter the route suggestions based on the shortest route or the minimal number of transfers and on their own preferences, and the result will be recalculated within 3 seconds.
8) The system will show a complete itinerary with the name of the stops/stations, departure and arrival times and transfer points, within 1 minute of the GTFS schedule data.
9) Users will be able to store at least 5 favourite routes or destinations and re-access them in future search, and they will be immediately available once the user is logged in.
10) Registered users will be capable of creating accounts, logging in in 5 seconds, and have customized functionality (last 20 searches saved preferences, trip history and route alerts).
11) The system will provide time-dependent routing on the backend by using dynamically scheduled edge weights dependent on schedule, which are recalculated at least once every 30 seconds during active session.

Non-Functional Requirements

1) The application would give back route results 3 seconds after a query submission.
2) It will be capable of supporting at least 3 transport providers when launching and enable new services to be supported with less than 1 day integration per provider.

3) Routes will be checked against GTFS data and refined at least once every day and consistency checked against live schedules.
4) Each of the user data (accounts, saved routes, preferences) will be encrypted with Django authentication and industry-best security practices in databases (hashed passwords, TLS) and no unencrypted data will be stored.
5) The frontend (React) will work on all popular browsers (Chrome, Edge, Firefox, Safari) and mobile devices, and performance testing will indicate that it takes 2 seconds to load on a 4G network.
6) The codebase will be composed of modules where Django will process the backend logic and React will render the frontend so that the bug fixes and add features can be delivered in 1 week.

Usability Requirements

1) The interface will be completely responsive (desktop and mobile) and all major functions (search, filter, switch routes) will be reachable in 3 clicks/taps.
2) The map (through Google Maps API) will show the routes, stops, and transfers with 95 percent accuracy with respect to GTFS coordinates.
3) The users will have the ability to view routes in a comparison format and switch between transport modes on/off with the UI being updated within 2 seconds.
4) Setting will enable units (km/miles) and priority of transport mode settings to be saved between sessions.
5) The features of accessibility will guarantee that users do not need to make long walks (more than 1km segments) due to mobility-friendly settings.
6) The system will list the previous 5 routes searched automatically on the homepage so one can access it easily.
7) The visual design will be done in such a way that the new user is able to successfully complete a trip search on his or her first attempt of using the app in 2 minutes.

Extended Requirements

1) There will be an administration panel where route data can be managed, the health of the monitoring system can be monitored and GTFS feeds can be refreshed in less than 10 minutes.
2) This system will give projected load/congestion graphs of the travel based on the previous usage, which should be updated after every hour or so.
3) Real time (less than 1 minute wait) alerts, disruptions and cancellations will be sent to registered users upon receipt of the data.

4) The service providers will be equipped with more sophisticated analytics tools and the reports will be generated within under 30 seconds on commuter queries.

Use Case Narratives

The following use cases, describing user interaction and program response(s) are implemented in the final

Use Case:	Backend route planner	ID:1	
Actors	Commuter		
Stakeholders and interests	Commuters need reliable, fast and cost-effective travel routes		
Brief description	A backend route planner receives information of start location, destination, and time supplied by the user after which it determines the most effective travel path based on the public transport schedule and data.		
Main Path	1)The user logs in, or starts the application, types in the start location (Claremont Station) and destination (Cape Town CBD). 2)The backend accesses schedules of buses, trains and taxis in the database. 3)The planner computes the quickest route based on existing traffic, and schedules. 4)The results are delivered to the frontend to be shown		
Alternative Path	1)In case one mode of transport being unavailable (e.g. train strike), the planner resorts to plan B on how to use alternative methods by bus or taxi. 2)In case of absent real-time data, it takes the previous schedule and marks it as unreliable. 3)In case the user inputs a vague location, the system requests clearance to plan.		
Preconditions	Backend operational, schedules available		
Postconditions	Fastest and most cost-effective routes shown to user		
Related Use cases	Route filters and preferences, Time specific routing		

Use Case:	Multiple routing algorithms	ID:2	
Actors	Commuter		
Stakeholders and interests	A tourist wants a variety of route options		
Brief description	This use case allows the backend to adopt various routing algorithms with regard to the needs of the users and the availability of data. The system is able to adopt different approaches rather than the one size fits all route search		
Main Path	1)The user chooses a travel preference (e.g. fastest, cheapest, fewest transfers). 2)Backend selects the right routing algorithm. 3)Available data is processed by algorithm to obtain an optimal route. 4)Backend routes to a display in frontend.		
Alternative Path	1. In the case of choosing an algorithm that fails because of missing data, system reverts to some simpler algorithm. 2. When two or more algorithms obtain close results, backend sorts them out so that a user may select them. 3. In case real-time data evolves in the course of calculation, re-run of an algorithm with newer data.		
Preconditions	Algorithms (Fastest and most cost effective) implemented and selectable, location and time data entered		
Postconditions	User can view a variety of routes.		
Related Use cases	Route Filters and Preferences, Map overlays		

Use Case:	Frontend Journey loader	ID:3	
Actors	Commuter		
Stakeholders and interests	Frequent users require fast access to saved trips		
Brief description	In case the user wants to take a saved journey, the system recalls the journey information and checks the present schedules against them then updates the information to be		

	current. In case of disruption, the system offers to change by providing alternatives. This makes repeating journeys (e.g. a weekly commute or frequent family visits) far easier and removes the necessity of manually designing the same route again and again.
Main Path	1) User logs in to his/her PTJP account. 2)User opens the application and goes to ” Saved Journeys“. 3) User takes one of the saved journeys out of the list. 4) Frontend request get changed travel times at the backend. 5) Backend gets thus the latest schedule. 6) New route presented to the user will have visible time of the departure, transfer stations, and the expected arrival.
Alternative Path	**Journey Not Available:** In case the original route is not available system automatically asks alternate routes to the backend and present them to the user. **Multiple Choices:** In case there are a number of equally good equivalents available, then the system shows them in the order of fastest, cheapest, or least amount of changes depending on the preferences of the user.
Preconditions	Saved Journeys stored in the user profile, schedules must be retrievable
Postconditions	Saved journey with most recent times or alternatives
Related Use cases	Account and Saved Routes, Error handling

Use Case:	Route Updating	ID:4	
Actors	Admin		

Stakeholders and interests	Users: Rely on the right timetables to get journey planning correct. System admin: Wants reliable data.
Brief description	Updating bus schedules is done by an admin at the PTJP dashboard. The system can automatically refresh results and flag broken route links within a process by the time live users have to be impacted by them.
Main Path	1. Admin logs in to the dashboard of PTJP. 2. Admin modifies route details and is able to make changes. 3. Refreshes of the results of system are done publicly. 4. Modifications manifest themselves to all those using it.
Alternative Path	1. New routes that break are flagged in the system so that the admin can fix it before it goes live.
Preconditions	It has a valid user and pass that is logged in to PTJP dashboard. The API to load map and route data is working. Admin is allowed to edit and save schedule. Available routes and schedules got stored in the system.
Postconditions	Utilization of the updated route data is saved successfully in the system.

	The new data replaces the public-facing results. In case of existent broken route links, such information is marked, and admins are able to review them. All active users by searching will encounter new routes.
Related Use cases	Admin Panel

Use Case:	Route filters and preferences	ID:5	
Actors	Commuter		
Stakeholders and interests	Commuters want to tailor their travel route to best suit their needs (i.e. minimum walking, minimum cost)		
Brief description	This use case would enable the user the option to customize the planning of the journey using the filters like not taking specific mode of transport, restricting walking distance, or making cost/time the priority. The system eliminates the unsuitable routes and dynamically sifts out the unsuitable ones based on these preferences providing the user with the personalized selection with a backup selection provided as available.		
Main Path	1. Preferences are set by user. 2. Filters are provided in the system. 3. The user chooses route.		
Alternative Path	Display closest available at any cost and explain.		
Preconditions	User is logged in/has access to preferences; Routes are open; Filtering is turned on.		
Postconditions	Display the appropriate routes based on the filters		
Related Use cases	Backend Route planner, Multiple routing algorithm		

Use Case:	Visual Loading Feedback	ID:6	
Actors	Commuter		
Stakeholders and interests	Passengers feel more at ease when they can see visual confirmation of progression from the system.		
Brief description	This aspect guarantees that when a route query between Somerset West and Table View takes too long to load, say as a result of high volume of rendered data or short-lived network congestions, the user will get an artificially smooth progress animation and feedback message such as; “Finding the best routes…”. When the system overtakes more than 5 seconds, it provides the initiative of giving a request button to try again, or troubleshoot the internet connection. This keeps the aspect of responsiveness and provides users with a course of action to follow instead of sitting with a blank or jammed screen.		
Main Path	1. User enters trip query (i.e. Somerset West -> Table View) 2. Backend starts analyzing route data. 3. Frontend shows progress animation at immediately. 4. If result is available in <5sec, results are displayed straight away. 5. If processing time takes more than 5 seconds, show ‘Retry’ button and ‘check connection’ tip. 6. Results will be shown when backend is finish.		
Alternative Path	**There is a slow Network:** System admin shows a warning earlier (at ~3 sec) warning that things are going to be bad. **Connection Drop In The Middle Of A Search:** Animation halts and the message is to reconnect and try again.		

	Backend Timeout: When a search fails after a maximum time waiting (e.g. 15 seconds), the system aborts the search and asks the user to retry.
Preconditions	Requires backend processing route request
Postconditions	User provided with feedback about their progress or are prompted to try reload the page.
Related Use cases	Error Handling

Use Case:	Error Handling	ID:7	
Actors	User		
Stakeholders and interests	Reduce searches with no result and improve satisfaction for the user.		
Brief description	Users might be in a rush misspelling within user inputs. This feature will suggest corrections and recommendations based on the inputted user data.		
Main Path	1. User signs in. 2. Saves route. 3. Data is updated in systems.		
Alternative Path	Otherwise, route displayed with warning iteration of outdated.		
Preconditions	The journey planning screen is presented to the user. The location database of the system is loaded and available. A location has been typed in the search field (with possible errors). The input validation and suggestion nature of the system is in place.		

Postconditions	Once a legitimate place is identified, applicable routes are shown. In the situation when no definite match is produced, the system will suggest similar places. In case of no available suggestions, the system shows an error message with advice. The search process will result in a state where user is aware of what to do (chose a suggestion or re-enter input).
Related Use cases	Route filters and preferences, Admin Panel

Use Case:	Account and Saved Routes	ID:8	
Actors	Commuter (Logged in user with saved routes)		
Stakeholders and interests	Users prefer easy access to routes they travel frequently as this makes the journey more efficient		
Brief description	Account and Saved Routes feature provides users the possibility to create and manage account in which they can save their favorite routes. This implies that a commuter can save his daily commute to work and a student can save his weekend bus route to the campus without searching. The functionality has a secure authentication system with support to add, edit or delete any saved routes and device-synchronized basic.		
Main Path	1. User registers or logs in to his/her PTJP account. 2. User clicks through on the “Saved Routes” section. 3. User can create new journey: enter start and destination point.		

	4. User can save the route. 5. Saved routes are stored into the account normally the profile of a user so as to be retrieved in the future.
Alternative Path	**Already a Route:** In case the route is already stored the system will prompt the user to either overwrite or to retain the two routes. **Invalid Route:** In case the backend fails to identify an effective route, system alerts the user. **Duplicate Name:** The case already has particular name, then system requires a different name.
Preconditions	The user account must exist and have saved routes in the database.
Postconditions	Preferences along with saved routes are applied to the journey planning
Related Use cases	Frontend Journey loader, Route Filters and preferences

Use Case:	Admin Panel	ID:9	
Actors	Admin		
Stakeholders and interests	Admins can update route schedules as well as add new routes to ensure data is accurate and user journeys are planned correctly		
Brief description	The second use case deals with the capability of the admin to administrate the transport data via central dashboard. It has route editing, time table fixes, and comes with a verification of changes before publishing so that no data is corrupted and the system is reliable.		

Main Path	1. Admin enters with a log-in. 2. Updates timetable. 3. Saves changes.
Alternative Path	There are flags in the system that alert the errors of publishing.
Preconditions	Admin must be authenticated and backend must allow schedule edits
Postconditions	Updated timetable for routes
Related Use cases	Map API Integration and Algorithm Performance insights

Use Case:	Algorithm Performance Insights	ID:10	
Actors	Admin		
Stakeholders and interests	Admin can review performance of algorithms in order to optimize the system.		
Brief description	This use case can allow the admins to compare the various routing algorithms as far as the speed, efficiency, and the frequency of the transfers are concerned. The reports on the performance assists in determining the algorithm that should be the default setting considering the optimal overall user experience.		
Main Path	1. Login in as Admin. 2. 2. Comparison of views report.		
Alternative Path	Display message indicating no report is available.		
Preconditions	The performance of each algorithm must collected with data		
Postconditions	Decisions can be made on a default algorithm		
Related Use cases	Multiple Routing algorithms, Admin Panel		

Use Case:	Stop/Station Details	ID:11	
Actors	Commuter		
Stakeholders and interests	Commuters need detailed information on stops in order to effectively commute		
Brief description	Such use case will enable users to find all the necessary information about any station or stop along their route such as facilities, schedule, and connection opportunities to make better travel-related decisions and prepare to change transportation.		
Main Path	1. User clicks station. 2. Pop-up presents details.		
Alternative Path	In case schedule is missing, give unavailable message.		
Preconditions	The data for each stop must be available		
Postconditions	The stop details are displayed or a message is shown if they are unavailable.		
Related Use cases	Time specific routing, Map overlays		

Use Case:	Settings Management	ID:12	
Actors	Commuter		
Stakeholders and interests	Commuters will prefer using the app if they are able to put the app into the settings that best suit them (i.e. miles instead of kilometers)		
Brief description	The use scenario allows the interface customization to units (e.g. miles/kilometers) and mode priority (bus, train, taxi). Such preferences take to the entire search results immediately and can be set aside back to the defaults anytime.		
Main Path	1. User goes to settings. 2. Updates preferences. 3. App reloads.		

Alternative Path	User falls back to defaults.
Preconditions	Settings panel must be accessible to the user
Postconditions	The map and instructions update to the new personalized settings.
Related Use cases	Route Filters and Preferences, Account and Saved Routes

Use Case:	Help and Tooltips	ID:13	
Actors	Commuter		
Stakeholders and interests	Commuters use the app for the first time might be confused and need to be explained how to use the applications features		
Brief description	This use case directs the new or mixed up users with help of FAQ, support contact. It lessens the learning curve in filtering, settings, and advanced features usage.		
Main Path	1. User taps “Help” icon. 2. Redirects to FAQ and help page.		
Alternative Path	There are user contacts in case of need.		
Preconditions	Help content must be available		
Postconditions	Commuter understands the features of the app or contacts help.		
Related Use cases	Route Filters and preferences, Visual loading feedback		

Use Case:	Congestion Display	ID:14	
Actors	Commuter		

Stakeholders and interests	Commuters might not want to travel during peak travel hours and prefer to wait till the roads and transport services are less busy
Brief description	The use case will include congestion data used in the travel planning process so that users can make comparisons between travel routes not only based on time or cost implications but also the level of crowding and traffic congestion occurring on the route, increasing the comfortability of the choice of travel.
Main Path	1. User searches. 2. App indicates ETA and traffic jams. 3. Route is taken by the user.
Alternative Path	Where not more current information about traffic is available, indicate forecasted congestion.
Preconditions	Live or predicted congestion data available
Postconditions	Route choice is made based on the congestion information
Related Use cases	Time specific routing, Map overlays

Use Case:	Map Overlays	ID:15	
Actors	Commuter		
Stakeholders and interests	Users who are unfamiliar with the area they are travelling want to see what facilities as well as attractions are in the area		
Brief description	This use case augments maps with other contextual information like tourist attractions, real-time traffic, or service outage. Overlays can be turned on and off so users can view different kinds of information that can be applicable to the journey.		
Main Path	1. A user activates an overlay. 2. Data updates the maps.		
Alternative Path	Overlays switching user.		
Preconditions	Overlay data available		
Postconditions	Map updated with selected overlays		

Related Use cases	Station Details, Map API integration

Design Overview

Pages

Pages are structured as the list of frames that are loaded on the startup and can be navigated with the help of the GUI buttons. The pages hold the different functionalities of the system and depict the view.

Signup

The Sign-Up page will allow new commuters to create secure accounts to be able to save routes, set preferences, and see trip histories. It is also designed similarly to the rest of the site: it has a branded header, a looping video that is muted on big screens, a semi-transparent white registration card and a footer containing support contacts. The form gathers name, surname, email, username, password and React state validation and POST call to /api/signup/. Immediate feedback is provided to users--on success, they are redirected to a login page; when they make mistakes, they are presented with corrective information. The navigation is made easy with a login link. The page supports some of the most important use cases, e.g. Account and Saved Routes, Frontend Journey Loader, and Settings Management, as an account is the gateway to all the customized capabilities.

FAQ

Quick, layer help is given in the FAQ page. It maintains the standard header/footer and emphasizes a list of headline-foldable questions in the middle, a button called Back to Home and a video tutorial. Answers can be scanned or expanded to include activities such as planning a journey, saving routes or filter adjustments. Novices may view the video as a summary, more experienced persons go directly to certain questions. Immediate human support is provided through contact details, responsive design and clear branding creates trust. The page confirms the Use Case 13 (Help and Tooltips) and strengthens such route related features as saving journeys, filtering preferences, and learning about visual loading feedback.

HomePage

PathPilot dashboard is the Home page on which commuters find and save routes by filtering. It is a dark-blue header which offers access to Saved Routes, FAQ, User Settings, and (with superuser access) the Admin Panel. The primary text is divided into a sidebar on the left and a big interactive map. The sidebar provides Google Maps Autocomplete in the origin/destination, route search, filter (e.g., minimal walking) and journey saving. Chosen routes and live directions are shown on the map and the actions

or mistakes are confirmed by the toast notifications. The advertisements are placed in a subtle position below the map. Since the page is entirely responsive, the page links to core use cases: Backend Route Planner, Multiple Routing Algorithms, Route Filters and Preferences, Saved Routes, Visual Loading Feedback, Error Handling and access by the administrator.

Login

The Login page makes both the commuters and the administrators legal and offers security and user-friendly user experience. It has a branded header, a central semi-transparent login card and a responsive background (looping video large screens, constant grey on small screens). After typing in a username and a password, the user is granted an access token that is saved in the browser and allows one to navigate freely to the Home page. Unsuccessful efforts result in definite warnings. An enrollment button is used to serve novices. Admins make the same interface but are allowed privileged access after authentication. The page is essential in order to make personalized functions and links to such use cases as Frontend Journey Loader, Account and Saved Routes, Admin Panel, Algorithms Performance Insights, and Settings Management.

SavedRoutes

The Saved Routes page is only accessible to logged-in users and allows the commuters to view, show, and control frequent journeys. There is a regular header and footer that frames a primary set of panels: a sidebar of saved routes including delete buttons and a dynamic Google map panel which displays the path chosen in transit mode. The routes indicate the point of origin, the point of destination and the date saved. Confirmation also alters the list automatically by deleting entries that update the list immediately. The design will be responsive and will make sure that it is usable on any screen. The key use cases are Account and Saved Routes, Frontend Journey Loader, and fault tolerance, in case of backend or map services failure.

User Settings

This user secure page enables the user to update personal details and change passwords. It has a branded header, intuitive back navigation, and two white-card forms, one of them being the profile updates (name, email, username) and the other being the password update. They both communicate with backend APIs, and give either instant success or error messages above the forms. It is made of responsive design, round edges, and uniform branding, which bring out clarity and trust. It promotes the use case of Settings Management directly and reinforcement of the Account and Saved Routes use case as it keeps user information accurate.

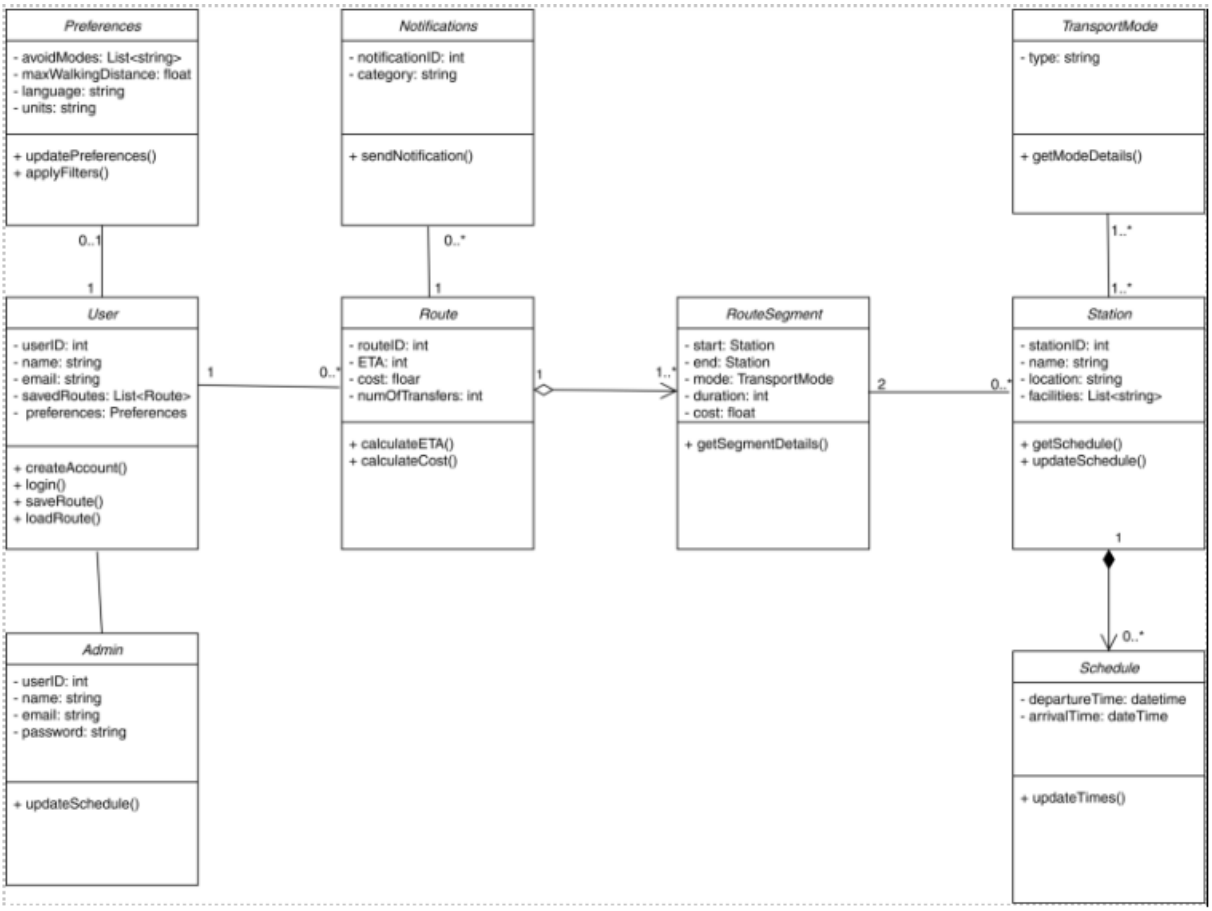
Admin

Django Admin interface enables system administrators to work with the information about user accounts and transport routes. Details about routing can be kept up-to-date by admins adding and removing users, modifying permissions, and updating schedules. These functionalities play a crucial role in the Route Updating, Admin Panel, Backend Route Planner and Multiple Routing Algorithms use cases, through which commuters can always know about the reliability of their travels.

System Architecture

PathPilot has a modular architecture which is an object-oriented architecture. The classes that are the most significant to be used are User (create account, log in, saved routes), Preferences (transport modes, walking limits), Route (full trip with ETA, cost, segments), and RouteSegment (separate segments of each trip). Schedule handles departure/arrival schedules, and is able to update locally without affecting the entire system. The notification dispatch route-specific notifications, whereas Admin maintains the data integrity. This division is reflective of actual travelling trends and contributes to scalability, personalization and dependable real time updates.

Class Diagram



Data Processing

Our decision to use the format of a General Transit Feed Specification (GTFS) instead of raw CSV files is due to the fact that the latter offers the standardized format needed by the routing algorithms used in this project. The algorithms like Dijkstra and RAPTOR are based on clearly defined relations among them in terms of stops, routes, trips, and schedules to calculate efficient journeys. GTFS divides this data into a number of interrelated files (calendar.txt, calendar_dates.txt, agency.txt, stops.txt, routes.txt, trips.txt and stop times.txt), which collectively define the transport network in a uniform manner. The absence of such standardization would make the application of the algorithms much more complicated, since the raw CSVs do not have a standardized schema, and would incur more preprocessing and reorganization.

Technically, the application of GTFS has a number of benefits. To begin with, it is compatible and scalable as it is easy to include several providers (MyCiTi, Metrorail,

Golden Arrow) into one transport graph. Together with this, it enhances data integrity and maintainability. This is because the GTFS format has uniform format of times, identifiers and relationships. Third, it allows to utilize existing libraries and parsing tools dedicated to GTFS that saves time on the development and minimize data processing mistakes. Lastly, GTFS is also suited to Django, which uses relational database as the storage model, allowing transport data to be easily stored, indexed, and queried to provide the frontend React interface with the results of real-time route planning. To get the GTFS format a mixture of manual and automatic (python scripts) manipulation of the data was used.

Implementation

Significant Classes

User

User class will be used to represent commuters that engage with the system, which is the basis of personalization and account management. Its fundamental features are that it has a distinct userID, name and email with the help of which users can be identified and managed one by one. In addition to identification, it also provides savedRoutes, a list of Route objects and preferences, which connect them to a collection of Preferences with which to customize their journeys. CreateAccount is the most significant method that creates new users, the second method, is login which authenticates the users, savedRoute that stores frequent routes, and loadRoute which retrieves the routes efficiently. The User class is important because of these attributes and provides continuity and custom experiences in the system.

Preferences

The adjustable travel restrictions and filters that affect a user's journey planning experience are determined by the preferences class. Its properties include units, which indicate whether distances and times should be displayed in kilometers, miles, etc., maxWalkingDistance, which indicates how far the user is ready to walk, and AvoidModes, which is a list of modes of transportation that the user does not wish to use (e.g., avoid trains during strike). The primary methods, updatePreferences and applyFilters, allow users to dynamically modify these parameters, allowing them to fine-tune the outcomes of route computations. Because preferences have a direct impact on the trip's outcome, they make the travel recommendations pertinent to each person's wants and circumstances.

Route

The Route class represents a complete journey and gathers information that are necessary to compare and contrast travel choices. This class has the following

attributes: a unique RouteID, ETA (estimated time of arrival), cost and numberOfTransfers, which provide commuters with important information in decision-making. The approaches computeETA() and computeCost() enable re-computation of routes in real-time using new schedule information or delays, which makes them accurate and responsive. As a route consists of several RouteSegments, this class serves as the general container of journeys and allows the effective consideration of various options.

RouteSegment

The RouteSegment class records the individual part of a journey that represents a complete journey and thus a person can divide a complete journey into small manageable parts. It has characteristics such as the start, and end stations, mode of transport selected, time, and cost. The system can model complex routes that include buses, trains, or taxis by modeling each of them separately. The procedure getSegmentDetails() will give a detailed information on each leg and it is beneficial to the system as well as the user to know how the segment will add to the entire trip. This granularity is flexible, as single segments can be re-computed and the remainder of the route can remain unaffected.

Station

The Station class is a geographic and operation anchor point of journeys. It is a point in routes where a route starts, terminates or intersects. Its primary fields are stationID, name and location, which identify the station and facilities, a list of available facilities, i.e. ticket offices or restrooms. The process is dynamic and provides flexibility to the stations since the stations relate to the timetable and can be updated as the conditions vary through the methods getSchedule and updateSchedule. Storing contextual and logistical information, the stations constitute the core of route-building and timetable.

Schedule

The Schedule class describes the time aspect of travel in that it describes the particular departure and arrival times of a station. The departureTime and arrivalTime are its primary features that allow making accurate ETAs and transfer times that have a direct input in route planning. The approach updateTimes makes sure that the schedules could be modified due to the delays, strikes, or administrative changes to keep the systems reliable. Since timeframes are generated inside stations, the architecture enables the localized updates that enable scalability and preciseness.

TransportMode

TransportMode class indicates the mode of transportation which is present in the system e.g. bus, train, or taxi. It's one attribute, type, determines the type of travel which is

directly related to route segments. The getMoreDetails method enables one to access more contextual information on that mode like speed ranges, comfort levels or environment. In this class, the necessary flexibility in the route planning is presented by the ability to differentiate among the various types of services and make sure that the system can blend effectively.

Admin

Admin class is a kind of extension of the user in the sense it poses the control and management of the system. It has the properties of userID, name, email, and password that identify and grant access to privileged operations. The most important method is updateSchedule, with which administrators can create and update transport schedules in order to provide the correctness of route calculations and the accuracy of the journey planner. As this position is crucial to ensuring data accuracy and the capacity to react quickly to disruptions, the administrator is the guardian of system dependability and credibility.

Libraries Used

Two important libraries were used to facilitate functionality and reliability in the development of the system. bounding boxes were calculated with the RTree library, which is essential to calculate possible transfers between the stations or modes of transport within a particular geographic region efficiently. This feature of spatial indexing is important in terms of processing large sets of routes and locations due to which the transfer calculations do not slow down and are correct. Moreover, unit testing was done throughout the system with the help of the Pytest library. Offering an easy, but effective, format of writing and running tests, Pytest remained able to validate the key approaches and class behaviours and ensured that each of the elements worked properly and that the entire structure was resounding as it developed. These libraries combined to increase the performance and reliability of the solution, which helped in the correct planning of journeys and reliable system behaviour.

Program Validation and Verification

Class Testing

Integration Testing

Validation Testing

System testing

Regression testing

User Testing

Discussion of Test results

User Manual

How to use

The Journey Planner need to be efficient and easy to use. First, confirm that your system satisfies all of the requirements listed in Section 9. It is advised to create a Python virtual environment in order to install the required libraries and avoid dependency conflicts. The application can be launched by running the main.py Python program once the setup is complete. Plan a Journey to create a new trip or Saved Routes to view or manage existing routes are among the navigation options that appear when a user first opens the Home Screen. Other options include Notifications to review alerts about delays and disruptions, Preferences to select travel settings, Account to manage personal information, and Help to access tutorials and frequently asked questions.

To plan a route, select a starting place, a destination and select a mode of transportation like fastest route, the least expensive route, or minimum transfers. The system will also create the list of routes, including such information as estimated time of arrival, cost, and transfers. Users have the ability to expand a route to see the detail of the segments (start and end stations, transport modes, and duration). Favorite routes can be stored to be used later. Under the Saved Routes section, the user may access routes previously stored, update them in order to receive the latest timetable changes and rename the routes to avoid confusion. There are notifications that can be used to deliver real-time information applicable to active or saved journeys such as delays, cancellations, and alternative recommendations.

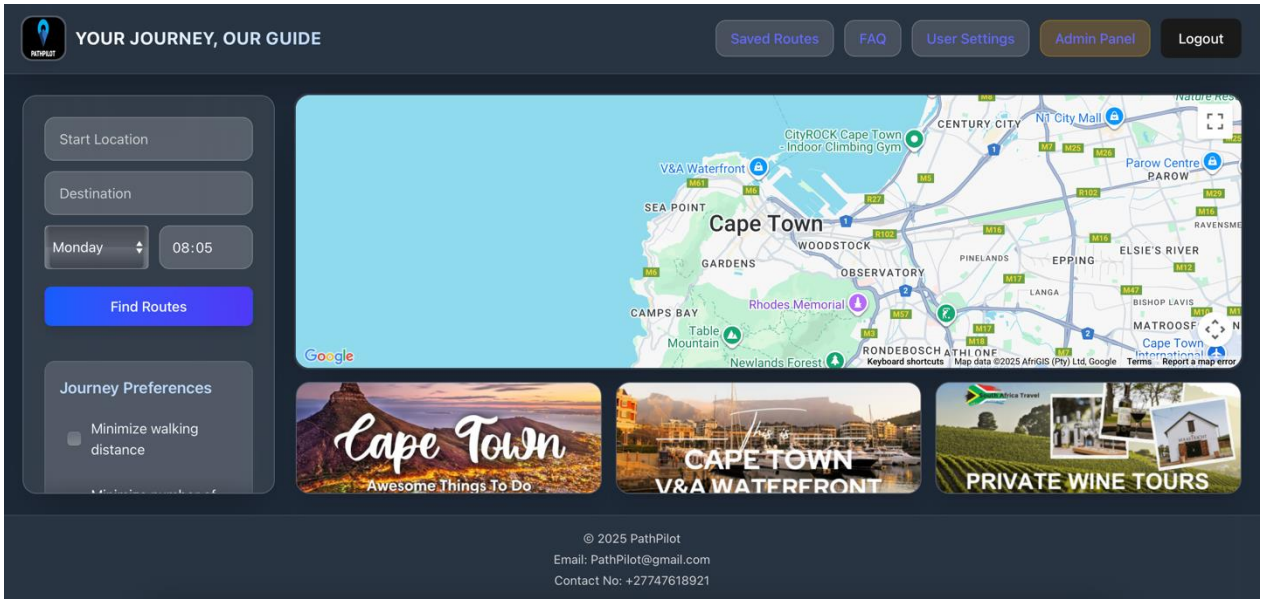
Preferences are also modifiable, and one can choose which modes should be avoided whenever taking a transportation, which is the maximum distance between each part and whether you find it better to show the distance in kilometers or miles. These modifications are implemented in real-time regarding search of routes in the future. A single click on any station in a journey gives detailed information of the station name, location, facilities available, and schedules updated. To the administrators, there is a secure administrator panel that allows updating timetables, correcting problems and look at flagged issues before making changes to ensure that all users have correct data. To help, you can use the Help menu which gives you tutorials, frequently asked questions, and troubleshooting instructions, and more help is available using the support page.

Requirements to run

Before using the Journey Planner, users must confirm that their system satisfies the requirements. To handle dependencies, it entails installing Python 3 (3.8 or later), a command-line interface, and (optionally) a Python virtual environment. Some Python

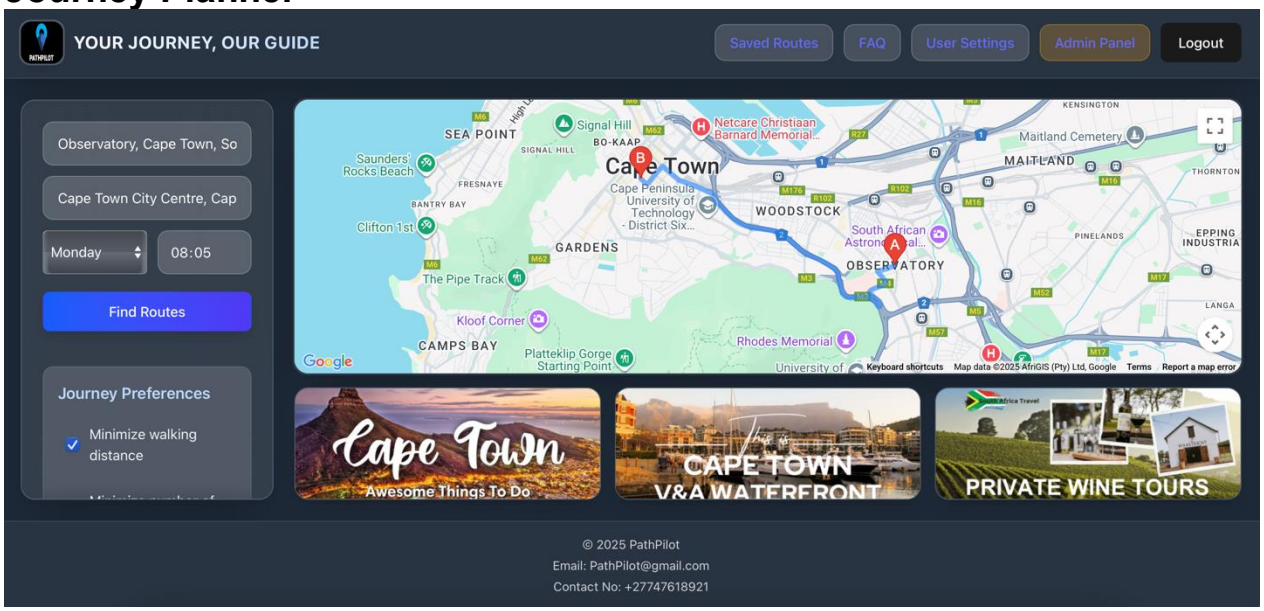
libraries have to be installed by the users as well: RTree (spatial computations), Pytest (testing), NumPy (numerical operations), Pandas (timetable), Requests (HTTP requests), Flask (web interface), and Geopy (geolocation).This is the process of setup and activation of a virtual environment, and installation using pip of the necessary libraries. After this is done, one is able to launch the application using the command python main.py where they can access the Home Screen.

Home Screen



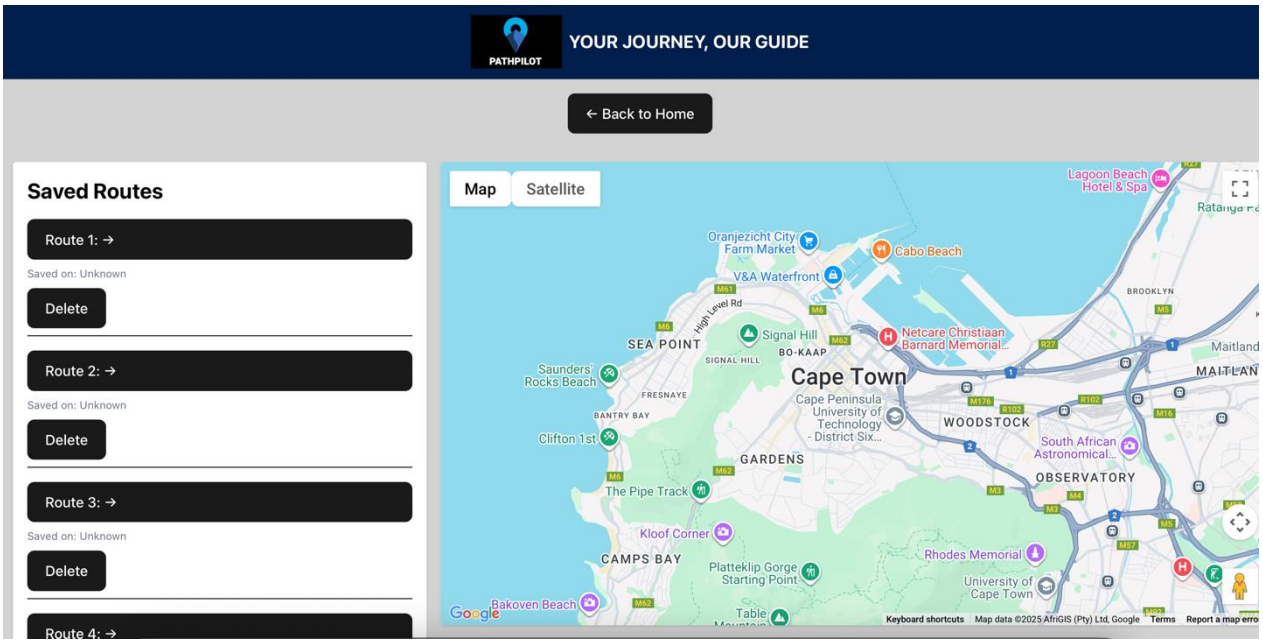
The Home Screen is shown as the user opens the Journey Planner. This acts as the application's primary navigation hub. The user makes the decision on what they want to do at this stage. They can enter their starting location, destination, and desired travel time when they select Plan a Journey, which will lead them to the journey planner screen. A list of previously saved routes will show up when the user clicks on Saved Routes, and they may see, edit, or rename them. Real-time notifications regarding delays or interruptions to planned or ongoing travel are provided by the Notifications feature.. The user is able to edit the travel settings via Preferences, including the modes of transport that he or she does not want to use, maximum walking distance, and units of measurement. The Account option enables the user to log in, create account, or update with account information. Lastly, there is the Help, which provides a tutor and the FAQ which will take the user through the feature of the system.

Journey Planner

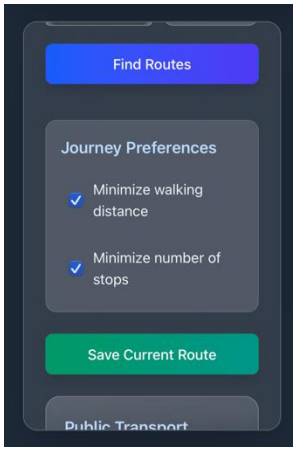


On clicking Plan a journey, the user is taken to the screen Journey Planner. In this case the user starts by typing in the starting point and the point at which they are going in the given input fields. They can then select their travel preference; whether it is the fastest route, the cheapest or the least number of transfers. After the search is started, the system provides a list of the options of routes with estimated arrival time, price, and number of transfers. Any route can be expanded by the user to display detailed route segments which display the details of each leg of the route e.g. names of the stations, mode of transport and duration. When the user is done with the review, he/she chooses the desired route and has the choice of keeping that as a default.

Saved Routes



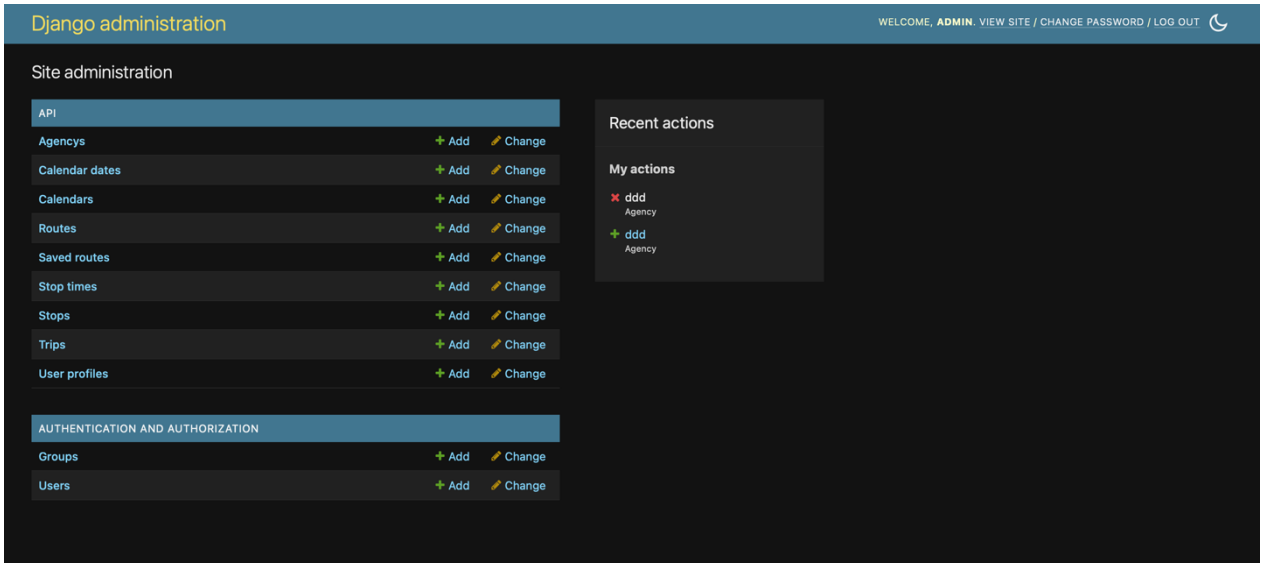
Saved Routes screen helps the user to view and organize saved trips. Based on this screen, the user may choose any of the saved routes and reload it with new schedule data, such that the trip would be correct. In the event of unforeseen delays that would not allow the route to be completed, the system automatically proposes other routes. The user is also in control of renaming saved routes in case of clarity hence when they have a number of similar trips. The user can easily locate known journeys without having to re-type the details with the assistance of this screen.



Preferences

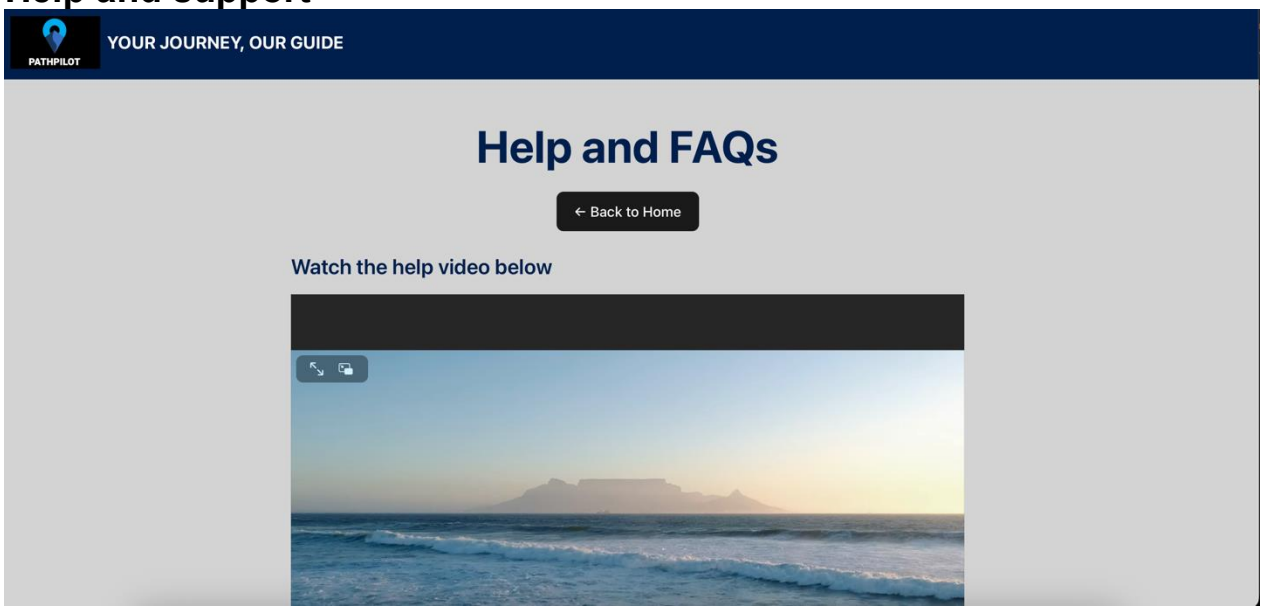
The Preferences screen can be used to configure the way the Journey Planner works to suit the user based on their individual requirements. The user is given the option of not using means of transport, e.g. buses or trains. They are also in a position to establish maximum walking distance that they are ready to cover between the parts of their trip. Also, the user is allowed to select his/her desired unit of measurement either kilometres or miles. Preferences are implemented in real time when they have been established, and the outcomes of future route searches will be based on the newly set preferences. This interface gives the user the ability to customize the journey planning process to his/her comfort and convenience.

Admin panel



Admin Panel provides the system to be directly managed by users with administrator access. When there is a need to change route schedules, to correct the errors in the timetable, and to review the flagged problems, administrators are able to log in and make adjustments to schedule changes before publication. As an administrator, this section helps in keeping all the route information accurate and reliable to all the users. Any revisions performed using the admin panel are automatically implemented throughout the system, thus ensuring the system has uniform service.

Help and support



The Help option is aimed at helping the user to navigate the Journey Planner. It includes tutorials on how to use the system, frequently asked questions about how to use it, like vague location or unavailable routes, and troubleshooting hints on issues like network or backend delay errors. As a user, the Help section is a forcefield, which makes them find solutions to their problems or clear up their questions promptly. To obtain additional help the user can contact the system administrators through the support page.

8. Conclusion

The project clearly showed that the PTJP system can incorporate data management, route creation and pathfinding methods into a transport planning solution. With the addition of the Django backend and a React frontend, we have developed a platform where administrators are able to manage stops, routes, and trips and to provide real-time access to APIs that provide map information, including Google Maps. Without including algorithms, it would be possible to generate routes, but to achieve efficiency in them, thereby providing the system with more than a mere data store. The movement to a structured database increased reliability and scalability of the text files whereas the use of REST API offered flexibility in frontend-to-backend communication. In general, this project emphasizes the significance of integrating software engineering concepts and real-life algorithms to address the issue of transport in the real world. Further improvements have allowed PTJP to be used as a baseline towards a more advanced and bigger set of intelligent transport systems.

9. Appendix A — Code Legibility and Output

Backend: (Django model for routes)

```python
class Route(models.Model):
    route_id = models.CharField(max_length=50, unique=True)
    agency = models.ForeignKey(
        Agency, on_delete=models.SET_NULL, null=True, blank=True, related_name="routes"
    )
    short_name = models.CharField(max_length=50, blank=True)
    long_name = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.long_name or self.short_name or self.route_id
```

10. References