University of Nevada, Reno

USA Rail Planner: A user-focused web-scraping solution for rail travel planning in the United States

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science and Engineering

by

Nicholas Alvarez

Dr. Sergiu M. Dascalu, Thesis Advisor
Dr. Engin Arslan, Thesis Advisor

December, 2022

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

Nicholas Alvarez

entitled

USA Rail Planner: A user-focused web-scraping solution for rail travel planning in the United States

be accepted in partial fulfillment of the
requirements for the degree of

Master of Science

Sergiu Dascalu, Ph.D.
Advisor

Engin Arslan, Ph.D.
Co-advisor

Richard Plotkin, Ph.D.
Graduate School Representative

Markus Kemmelmeier, Ph.D., Dean
Graduate School

December, 2022

Abstract

Planning a cross-country train journey in the United States can be a time-consuming process. The USA Rail Planner, presented in this thesis, provides travelers an easy way to plan a multi-city rail trip to any of the destinations served by Amtrak trains in the United States. The manual work of searching the Amtrak website and inputting information into a spreadsheet is no longer necessary. By interfacing with the website, information can be parsed by the application quickly and presented to the user in a simpler, ordered, and less cluttered format, allowing them to make educated decisions in their trip planning process. Dynamic route maps, detailed train information, and many other planning features are present in the application. Quality-of-life additions, such as train timetables, city tourism pages, and local transit connections, make the application well-rounded in the tourism and travel domains. Furthermore, this user-centered Python-based application that employs web scraping and other modern software technologies provides an efficient and easy way to create an itinerary which can be exported later. User study results (N=12) show that the USA Rail Planner is significantly better than existing methods, reducing the time to create an itinerary by 47% and it was the preferred method for all but one participant.

Dedication

To my entire family, and especially my wonderful parents, for their continuous support throughout my academic career. Without your love and guidance, you would not be reading this today.

Acknowledgments

I want to thank my advisors, Dr. Sergiu Dascalu and Dr. Engin Arslan, and my committee member Dr. Richard Plotkin for their time and support. Thanks to the encouragement from Dr. Dascalu, there is a thesis to read here today.

Credit for the application's city images goes to Google Images. Train information and search results credit goes to Amtrak. The station list information is from Wikipedia's "List of Amtrak stations" page. Route files are from Transitland. Status maps are made available via asm.transitdocs.com and dixielandsoftware.net.

And, of course, I want to thank Amtrak for inspiring me to build this application. This research is in no way meant to diminish the Amtrak website's comprehensive capabilities; it is meant as a complementary aid in planning. Without Amtrak and my cross-country trip, I would have had no reason to start my research and create this application.

Contents

List of Tables

List of Figures

List of Algorithms

List of Listings

Chapter 1

Introduction

Planning a multi-city rail trip in the United States can be difficult. With so many potential routes and destinations, it is easy to get overwhelmed trying to pick where one wants to go in a short travel timeframe. This also comes with the need to balance train connections, spend ample time in each destination, and to determine the feasibility of arrival and departure times. The Amtrak website's method for planning trips is searching by origin, destination, and departure date, one at a time [1]. A train must be selected from the results and its information recorded elsewhere, such as a spreadsheet or other organizational software. This is a time-consuming process, and if previous search results need to be checked, the search information must be re-entered, and the search resubmitted. The proposed USA Rail Planner (henceforth Rail Planner) provides a way for travelers to formulate a travel plan, with the application navigating and parsing the Amtrak website so the user does not have to. Selected trip segments (or "legs") can be saved in an in-app itinerary with relevant information for each train, which can be exported later to a spreadsheet.

1.1 Problem Statement

Immense planning is required to ensure a cross-country train trip comes together. Riders must consider which routes to take, where they will stop, any points of interest they want to see, the on-time performance of their desired train(s), connections to local transit, and a plethora of other variables. Documenting the information they research is a necessity, as building a rail travel itinerary is an iterative process, with routes and segments being added over time as the rider's potential travel paths come to fruition. Multiple journeys (collections of routes and stops, "travel paths") may be considered, thus increasing the complexity of this task.

There are no similar software solutions available to effectively plan rail trips in the United States. Manual work is required to search for and document each segment (leg of a trip) and the potential trains one can take. One existing method is Microsoft Excel, but this is a means for storing the data instead of finding it. The closest solution is simply Amtrak's website. While it is the absolute resource for finding train journeys in the United States, anything past purchasing the trip right then is limited due to the inability to save search results or specific trip segments. Results can also be spread over several pages, requiring users to navigate through each to see all listings, and are ordered by trip duration by default, not departure time. For this reason, the Rail Planner may present itself as a breakthrough application in the train travel field considering its novelty and feature set.

1.2 Motivation

I began very preliminary work on this application in December 2021. At that point, it was less than 100 lines of code and only had basic web scraping functionality. In short, it didn't work. However, the reason I started working on the project in the first place was because of my own upcoming cross-country train trip. I was frustrated with how difficult it was to plan an itinerary, even with only ten Rail Pass segments at my disposal. I had to perform each search by hand, inputting stations and a date, then peruse search results and record the information of selected trains to a spreadsheet. This spreadsheet quickly grew as I explored multiple route options given my short travel timeframe (2.5 weeks). And, with multiple potential routes, a lot of searches had to be performed, and a lot of train data needed to be transcribed to the spreadsheet. It was getting to be tiring, but around the same time, I began to seriously work on the application for a class, which required that I integrate a

GUI to complement the web scraping functionality. This working prototype was used to formulate my final trip itinerary, which I used as a reference to book my tickets and was also shared with family members who wanted to know my journey so they could track my trains. A few months later, my professor saw the prototype and recommended I continue researching and developing it and to write my thesis about it. I began adding more features and polishing the interface, and the complete application is the topic of this thesis.

1.3 Solution

The Rail Planner application presented here aims to simplify the travel planning process by eliminating the need to use the Amtrak website and the associated manual work. Every station in the United States national rail network is listed in alphabetical order to eliminate any guesswork about where one can travel by train. The origin, destination, and departure date will be selected, and the user can begin their search. After results (if any) are found, a list of trains and their attributes (departure/arrival time, number of segments, duration, etc) are be displayed and the user can select their preferred trip. This selection is saved and the user can begin their next search, repeating this process until their journey is complete. The final travel plan can be saved in a spreadsheet with all associated information about each trip, ready for later consultation. If the user wants to save their trip planning progress and continue later, they can do so with the application's `.RailPlan` files. The application features picture displays of cities where the selected stations are located to provide context, and they provide links to city tourism information if clicked. Additionally, helpful options are be available to the user, such as enabling or disabling the display of certain trip attribute columns, and supplemental information such as the Amtrak system map.

The intended users of the application are travelers who have purchased an Amtrak Rail Pass and need to plan their trip. The Rail Pass is a flat-rate ticket, purchased through Amtrak, that allows riders ten segments of travel anywhere in the Amtrak network [2]. In actuality, the user demographic extends beyond only pass holders, as the underlying search functionality would still allow for shorter or longer trips to be planned. However, one-way only searches and the UI elements related to the number of segments do cater to the aforementioned pass. Potential users will benefit from the application due to its immense time saving advantages. The inclusion of all stations may help inspire new travel destinations, but if a user already has an idea of which destinations they want to visit, the application will assist in compiling their trip segments into a spreadsheet itinerary for later review. This organizational benefit is not insignificant, especially if multiple potential groups of destinations are planned but the feasibility, such as the length of the trip or transfer times, is undetermined. The application should be both effective and efficient to use. That is, the desired results (a travel plan) should be achieved in as little time as possible. Any issues with a search or application function should be handled with an informational error message, as error handling is another goal. Finally, the application should be easy to learn, with the driving component of this goal being its minimal interactable UI elements that will guide the user to a completed search and finished itinerary.

The application would have some impacts across global, environmental, economic, and societal contexts. Globally, both Americans and non-Americans may be inspired to take a train trip, with or without the Rail Pass. The more people that choose to take a train trip over flying or driving, the less emissions are produced on a per-passenger basis, which showcases the environmental impact. On a societal level, it may promote train travel as a viable means for cross-country trips where urgency is not a concern. Families or friends could plan a trip together. The application shines with its economic impact, however. Those unaware of the Rail Pass and its benefits may be prompted to investigate further and buy one, instead of paying for each segment individually. On a more abstract level, the amount of time saved planning the journey, and potentially, the time maximized during the journey with a plan allowing for the most time at destinations, would further contribute. Destinations served by Amtrak may experience the impact of tourists arriving and spending money.

This Rail Planner application provides several enhancements over the existing method of manually searching the Amtrak website for each trip segment and inputting details into a spreadsheet. In summary, the thesis makes the following contributions. First, the act of searching by the user is entirely removed as the web driver runs all searches in the background and handles any errors as they arise. Second, all results are displayed at once in the application, instead of potentially multiple pages on Amtrak's site. Each train in the results table is shown with user-selected attributes, such as duration or prices. In addition, a popup window can provide more information about the train, such as amenities, available seats, and a per-segment breakdown of arrival and departure times. Third, results from previous searches can be viewed without the need to perform another search. These specific searches can be exported for later review as a spreadsheet but are also included alongside the user's itinerary in any saved `.RailPlan` file. Fourth, a specific trip from search results can be saved to an itinerary, thus forming another leg in the user's multi-segment journey. All trains in the itinerary can be exported to spreadsheet form, thus eliminating the need to scour the results when searching on Amtrak's site, as they'll already know what train to book. Fifth, a map is shown in a separate window, which can place markers for selected stations, a train's route with associated station stops, or an overview of the user's entire journey. Sixth, the user can save their planning progress in the application and revisit their work later without needing to start from scratch. Other minor contributions include photo displays of currently selected cities, tourism information about each city, and menu links to: route maps, train statuses, and timetables, and links to a selected train's information. These contributions are all part of this standalone application, so the user would only need to navigate to Amtrak's website to book their now-known trip segments once an itinerary is created.

The rest of this thesis is structured as follows: a background of national and regional rail travel in the United States, trip planning solutions for rail travel abroad (Europe) and domestically, and a high-level overview of web scraping are presented in Chapter 2. Related research to itinerary planning, both for rail travel and other means, as well as web scraping applications, is discussed in Chapter 3. Chapter 4 examines the application's specification, design, implementation, and the technologies in use. Chapter 5 presents the user interface design, typical scenarios a user might utilize the Rail Planner for, and the application's limitations. Chapter 6 contains the user study, including experiment methodology and environment, and its results. Finally, conclusions and avenues for future expansions are presented in Chapter 7.

Chapter 2

Background

In this chapter, brief backgrounds are provided for key components of the application. Section 2.1 covers the state of passenger rail travel in the United States. It provides context for the application's use and current challenges faced when travelling by rail. In Section 2.2, trip planning issues are discussed. Existing solutions from Amtrak are briefly presented. Next, Section 2.3 compares current trip planning solutions for passenger rail travel, examining benefits and drawbacks to their use. Finally, Section 2.4 provides a brief overview of web scraping. It details the technology behind web scraping, how it is used, and what it means for the Rail Planner.

2.1 Rail Travel in the United States

The United States is a large country with a national rail service. There are a variety of cross-country and regional routes available, often traveling through parts of the countryside that are otherwise inaccessible by road and serving communities where the train may be one of the only transportation options. However, unlike its European counterparts, the rail network in the United States is largely privately owned by freight rail companies and approximately 70% of passenger train miles are traveled on freight-owned tracks [3]. This can create delays, limit service, and inhibit route expansion for passenger rail. Several commuter rail lines exist, such as Metrolink in the Los Angeles area or Sound Transit in greater Seattle, and they may even own some or all of their rail mileage, but they do not offer the countrywide connectivity of a national rail system.

Amtrak [4] is the United States national rail network with service to over 500 destinations in 46 states. It is possible to travel from coast to coast on as little as two trains. Founded in 1971, its purpose is to provide accessible passenger rail service across the country. Some, such as those living in the northeastern United States or in California, use Amtrak trains to commute to work or take short leisure trips, as their service levels (frequencies) are much higher. Amtrak owns the Northeast Corridor tracks, thus eliminating freight railroad delays, and state-supported commuter rail, such as Capitol Corridor, generally see higher frequencies as well. Travel by rail shines when destinations are too far to drive and to close to fly. However, there are a selection of long-distance trains available for riders looking to go farther or access underserved communities. Despite their long travel times, either due to freight railroad trackage rights (defined as an Amtrak train using freight rails, despite not owning them) or the sometimes-treacherous mountain terrain in the western United States, they still provide an essential service and could be regarded as a lesser-known means of transportation. For those wanting to travel across the country by train, multiple connections (segments) are required, and the cost can often add up quickly. For this reason, Amtrak offers a "Rail Pass" for a flat price, granting the traveler ten segments between any destinations they choose, provided their entire excursion is completed in one month [2]. Amtrak's homepage provides destination ideas, current deals, and, most importantly, a search form, shown in Figure 2.1. This search area offers users a choice between one-way, round-trip, or multi-city searches.

2.2 Trip Planning Issues

For a rider who wants to plan their trip, options are limited. Spreadsheets, as mentioned, are the simplest option available, requiring the user to transcribe an individual train's information from the Amtrak website into their spreadsheet for reference later. Microsoft Excel or Google Sheets are prominent solutions in this category. Unless the user is saving each search result however, flexibility after inputting the information is

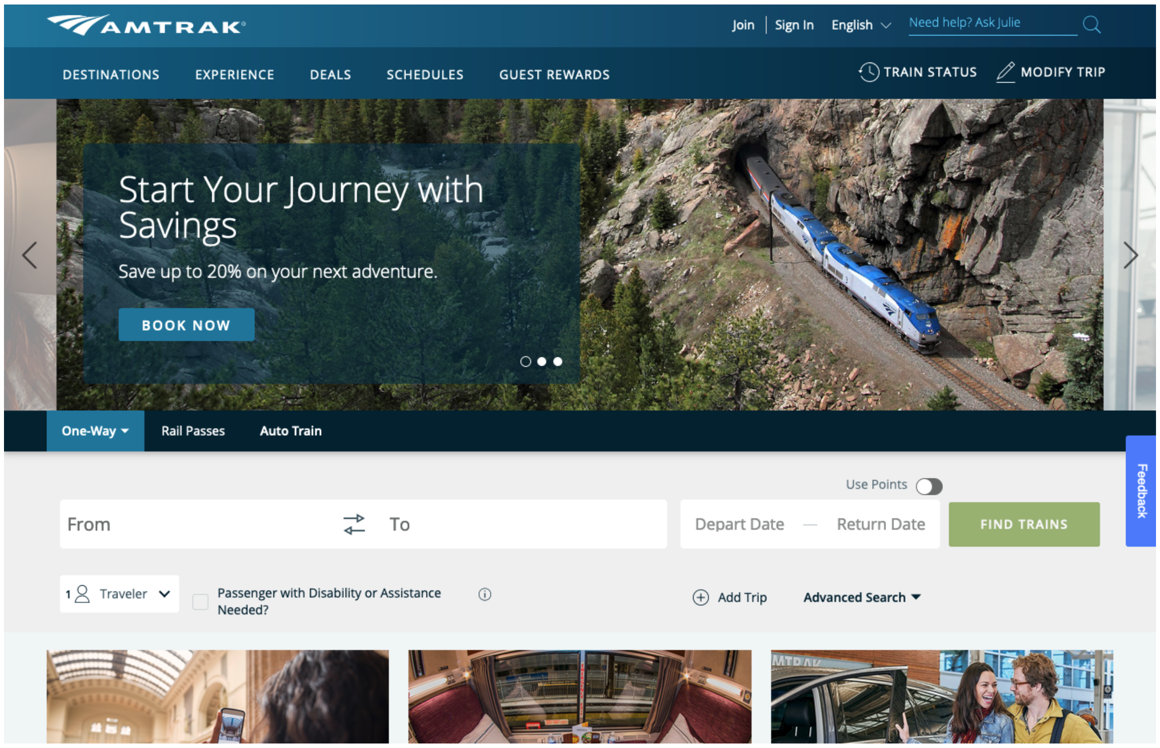
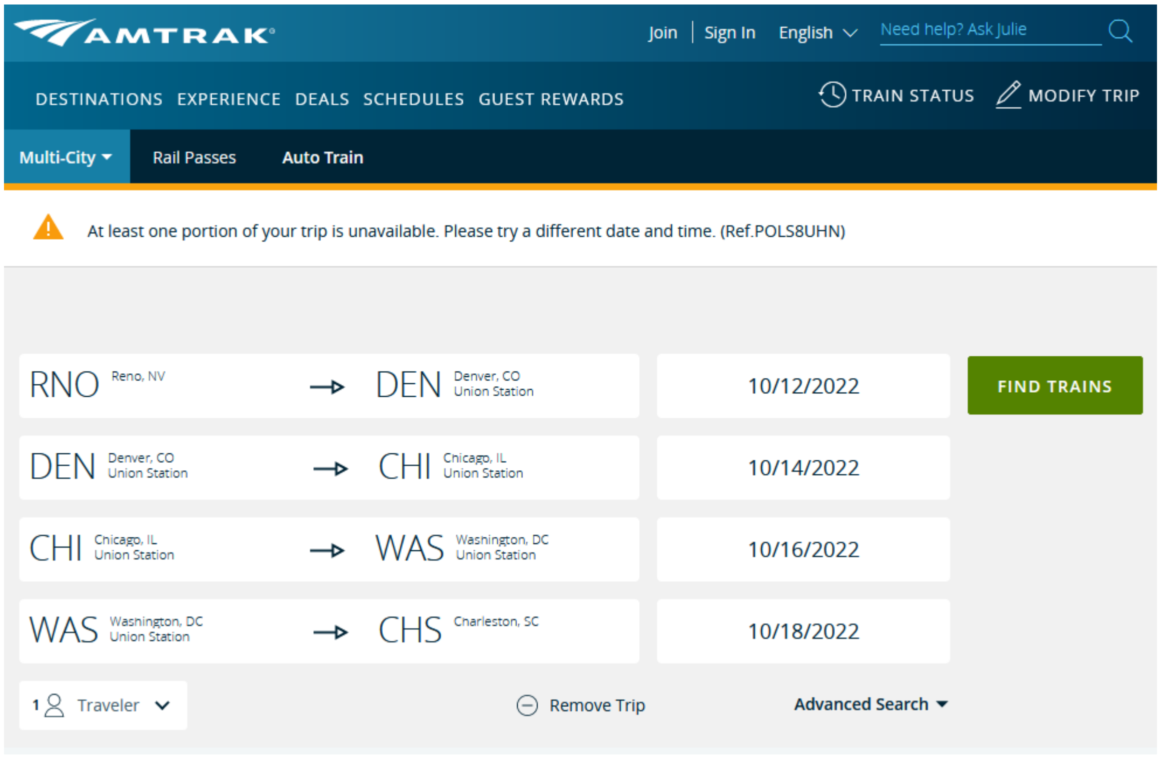


Figure 2.1: Amtrak Homepage

somewhat limited, such as returning to a set of search results to check alternate trains. Similar issues exist with other services, such as Notion or a calendar, in that obtaining the train data is a tedious, manual process of searching, parsing, and transferring the data into some storage medium.

Amtrak does offer a multi-city search function on their website [5], shown in Figure 2.2 with an error present. While it does include some of the functionality of the Rail Planner, it is still limited in that only four segments can be searched for at one time, and results are still listed on multiple pages with a great deal of wasted space. Searching multiple routes simultaneously does not provide the user any feedback as to whether there is sufficient connection time or if there is train service on the selected departure days – something only a rider with prior timetable knowledge would be able to discern. This can create unnecessary trial and error to perform a search. Additionally, it does not offer any way to save trains for later to an itinerary, unless the user books a ticket at the time of search.



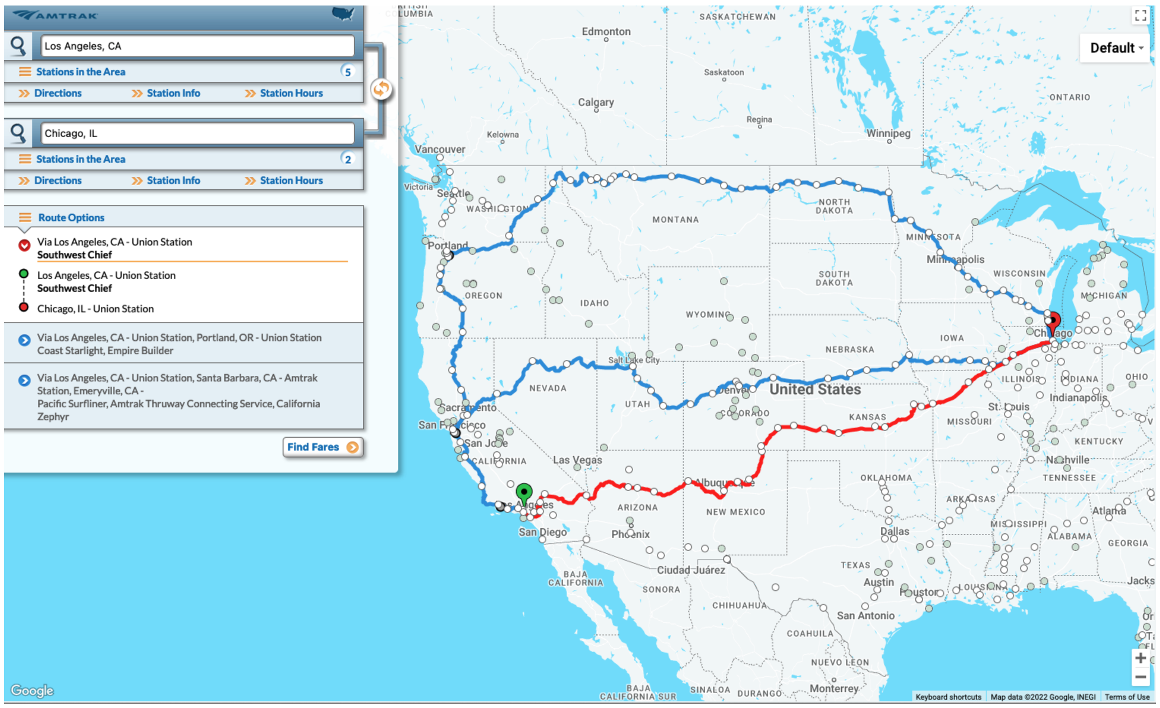
At least one portion of your trip is unavailable. Please try a different date and time. (Ref.POLS8UHN)

RNO Reno, NV	DEN Denver, CO Union Station	10/12/2022
DEN Denver, CO Union Station	CHI Chicago, IL Union Station	10/14/2022
CHI Chicago, IL Union Station	WAS Washington, DC Union Station	10/16/2022
WAS Washington, DC Union Station	CHS Charleston, SC	10/18/2022

1 Traveler

Figure 2.2: Amtrak Multi-City Search

There is a counterpart to the multi-city search offered by Amtrak, and is somewhat more similar to the Rail Planner: the Travel Planning Map [6]. An interactive map of all Amtrak stations (train and bus) with train routes displayed is presented to the user, and they can select any station on the map as a start- or endpoint, or they can be manually searched in the form fields for origin/destination. Users can find which routes serve those two locations, shown in Figure 2.3 with routes from Los Angeles to Chicago, however, it does not provide any distinction to the day of the week, so tri-weekly routes would be shown but may not provide travel options on the day the user is interested in. The "Find Fares" button opens a search form and, when completed, will bring the user to a new tab with any results for a specified day, somewhat solving this issue. One useful feature is the "nearby stations" portion of the station list, so users can find alternative stops in the area. Station information is also provided, such as amenities, parking, and hours of service – all useful information for trip planning.



Los Angeles, CA

Chicago, IL

Via Los Angeles, CA - Union Station
Southwest Chief

Los Angeles, CA - Union Station
Southwest Chief

Chicago, IL - Union Station

Via Los Angeles, CA - Union Station, Portland, OR - Union Station
Coast Starlight, Empire Builder

Via Los Angeles, CA - Union Station, Santa Barbara, CA - Amtrak Station, Emeryville, CA -
Pacific Surfliner, Amtrak Thruway Connecting Service, California Zephyr

Figure 2.3: Amtrak Travel Planning Map

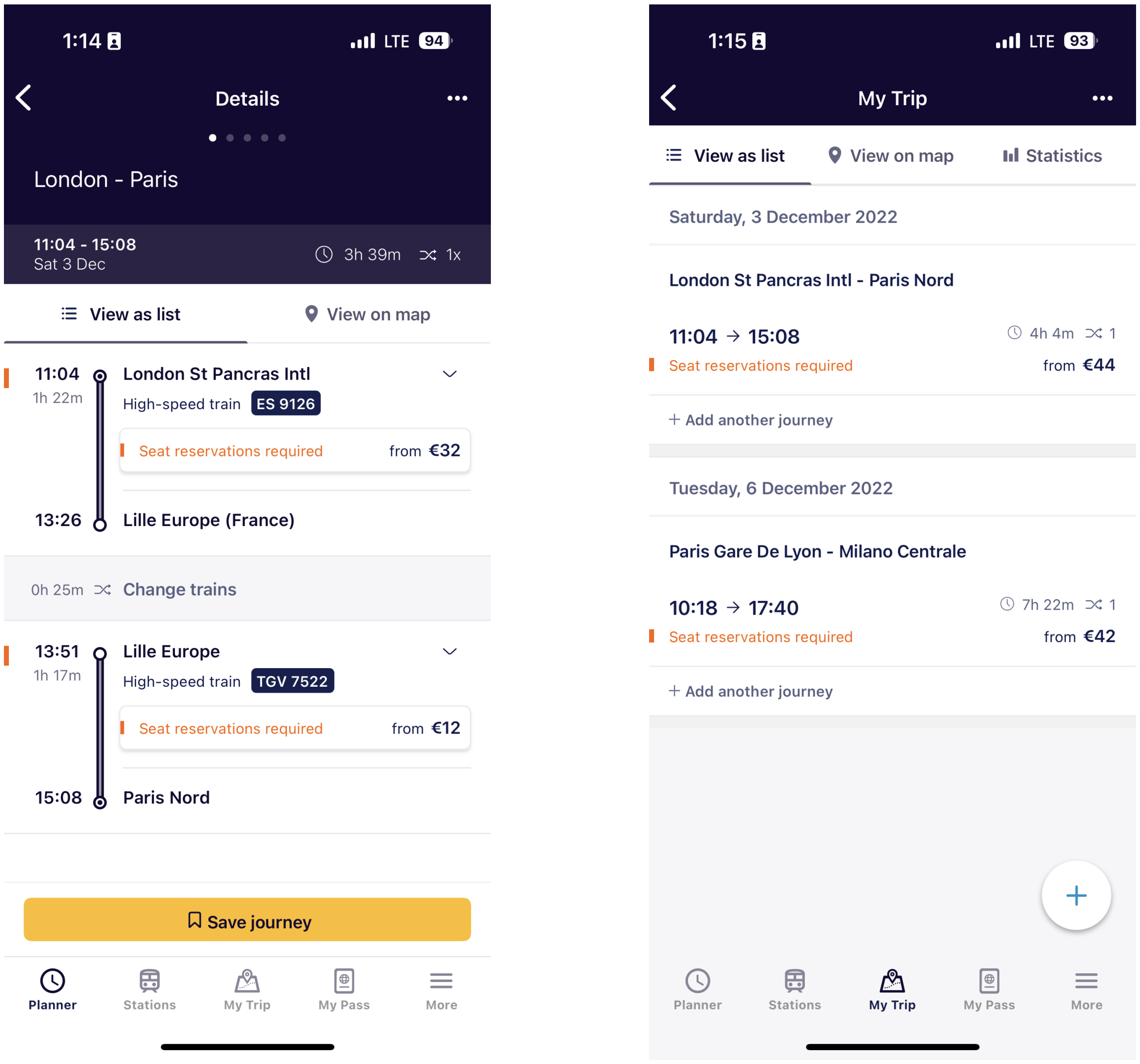
2.3 Similar Planning Solutions

The most similar approach to the Rail Planner is Eurail's trip planner map [7]. Users input travel dates and the number of passengers to be brought to the planner page. It features a map of partner countries, showing a popup for common destinations within the selected country, along with the number of days required at the destination. Repeating this process yields an itinerary, however each locations dates appear to be based on the number of days required at the destination and does not factor in travel time. For example, a trip from Stockholm, Sweden to Lyon, France indicates on the itinerary that the departure day is the same as the arrival day. This trip would actually take over a day on multiple trains. There is a "travel day" section that does appear to update based on the number of segments but not in regard to train travel duration. However, specific (and/or optimal) train selections are not present on this page. Alongside the web version is Eurail's mobile "Rail Planner" app, which holds an extended set of features compared to the web version, including the ability to function offline. This application is functionally much similar to the USA Rail Planner, in that users can search for trains between two locations and save them to a trip (ex. "My Trip"). An example of saved trains to a user trip is shown in Figure 2.4, with a search from London to Paris and the saved journey from London – Paris – Milano City. Their saved journeys can be viewed as a list or a map and includes a statistics tab listing the number of countries visited, kilometers traveled, and CO2 savings. A big benefit of the mobile application is that timetables are saved offline, so if a user does not have a cellular or WiFi connection, they can still plan their trip or check trains. It is a very capable application but does not apply to North American train travel.

Rome2Rio [8] is a popular travel planning website which incorporates air, bus, rail, and other transportation means. There are options to book tickets and hotels. Points of interest are not included, nor are exact prices for tickets, only an estimate. A map shows the route for the selected travel method, seen in Figure 2.5. Detailed train information cannot be viewed, only basic data such as arrival/departure times and the frequency of service at the origin. In a test search, it appeared to favor "shortest time" train journeys over one with fewer transfers, as it recommended the user disembark a train bound for Washington, D.C in Rockville, MD only so they could board a Metro train bound for Washington, D.C. Similar to Rome2Rio is Wanderu [9], another travel planner website. Filtering of results is included on their site, and it appeared the test search produced the "proper" Amtrak results (all via train) but was lacking in a map or any specific scheduling information. One benefit was the exact price display instead of an estimate. However, neither of these sites allowed the user to save results, add multiple stops to their search, or display points of interest along the route.

2.4 Web Scraping

To understand how the application works in the background to retrieve Amtrak search information, a brief overview of web scraping is necessary. Websites (or services, generally) may provide an application programming interface (API) to retrieve or


1:14 LTE 94

Details

London - Paris

11:04 - 15:08
Sat 3 Dec
3h 39m 1x

View as list | View on map

11:04 1h 22m London St Pancras Intl
High-speed train ES 9126
Seat reservations required from €32
13:26 Lille Europe (France)

0h 25m Change trains

13:51 1h 17m Lille Europe
High-speed train TGV 7522
Seat reservations required from €12
15:08 Paris Nord

Save journey

Planner | Stations | My Trip | My Pass | More

1:15 LTE 93

My Trip

View as list | View on map | Statistics

Saturday, 3 December 2022

London St Pancras Intl - Paris Nord
11:04 → 15:08 4h 4m 1
Seat reservations required from €44
+ Add another journey

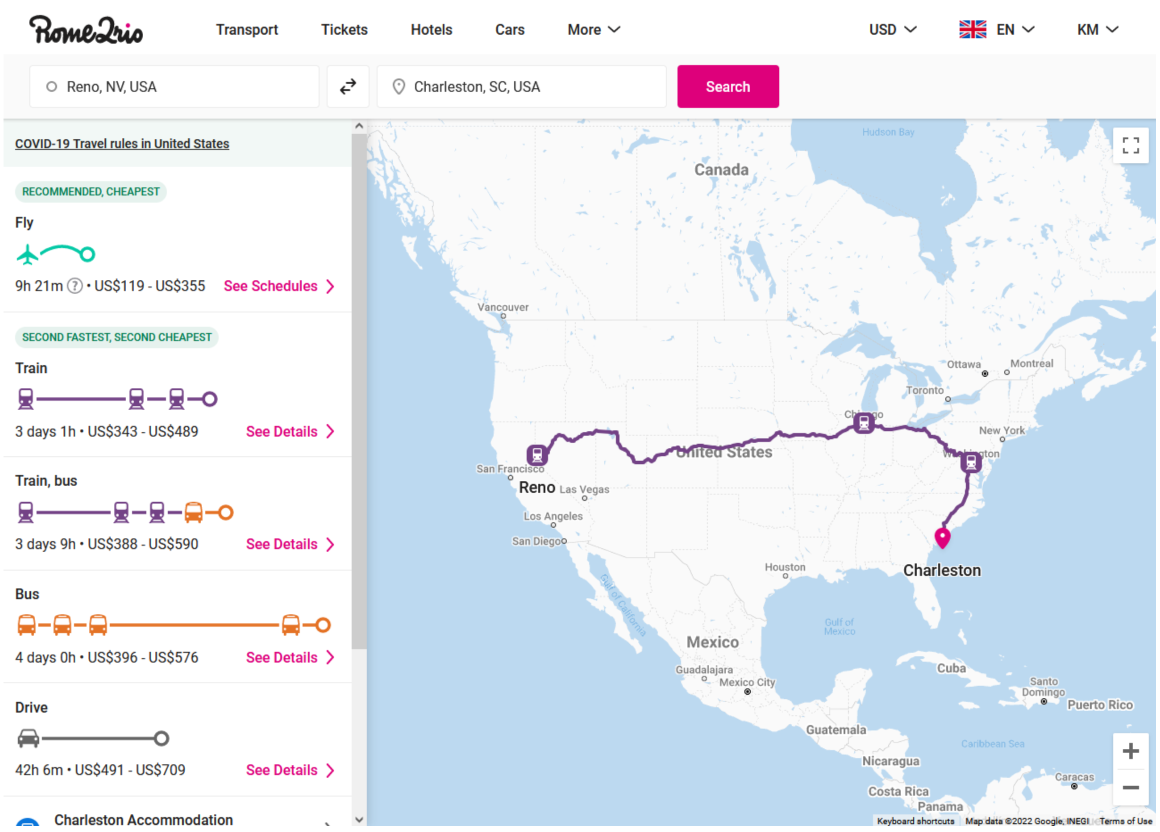
Tuesday, 6 December 2022

Paris Gare De Lyon - Milano Centrale
10:18 → 17:40 7h 22m 1
Seat reservations required from €42
+ Add another journey

Planner | Stations | My Trip | My Pass | More

Figure 2.4: Eurail/Interrail Rail Planner Mobile Application

send data from the site. One such example of this is booking an airline ticket from a third-party travel website, such as Kayak. While the third-party site itself may not know the current cost for a seat or how many tickets are remaining, it can ask the airline company for this information so it can present it to the user. The exchange of information is enabled by an API, which, at a very high-level, receives some data from the client or user, processes it or does something on the server side, and returns some data back to the client. In this way, an API acts as a messenger of sorts. It may seem logical that a website (or application) could ask Amtrak's API for train information



Reno, NV, USA

Charleston, SC, USA

COVID-19 Travel rules in United States

RECOMMENDED, CHEAPEST

Fly

9h 21m • US$119 - US$355 See Schedules

SECOND FASTEST, SECOND CHEAPEST

Train

3 days 1h • US$343 - US$489 See Details

Train, bus

3 days 9h • US$388 - US$590 See Details

Bus

4 days 0h • US$396 - US$576 See Details

Drive

42h 6m • US$491 - US$709 See Details

Charleston Accommodation

Figure 2.5: Rome2Rio Search Results

based on origin, destination, and date (number of travelers, flexible dates, etc. could also be parameters). However, Amtrak does not have a public API to use for ticket searches, only a semi-private API for train statuses (location tracking) [10], meaning all of that information must be obtained by visiting the website. Without another solution, users are back to the original problem of manually searching and recording data.

Web scraping refers to the extraction of data from a website, even without an API. Simple scrapers load the HTML code and nothing else, usually completing this task quickly. This form of web scraping is used in the Rail Planner to gather station information (name, transit connections, location, etc) and addresses. However, more complex websites incorporate Javascript elements, often containing the desired information. If the site can be loaded by a simple scraper, it is unlikely that it could gather anything useful from the Javascript elements. However, just as websites become more complex, so do web scrapers. When scraping with the help of a web driver, which takes complete control of a full-fledged web browser like Chrome and uses it to scrape, it can render more advanced parts of the website, such as Javascript elements, and access the data from them. Selenium is a Python library to assist with this task, providing an interface to control the web driver. Commands such as inserting text into (Javascript) forms, clicking on elements, scrolling, and other "interactive" functions are available. In the context of this application, a web driver is used to render the Amtrak site and is controlled via the Selenium library. Searches are performed as if a user was controlling the browser: typing, clicking, and scrolling. The resulting search page is interpreted by the application and parsed results are presented to the user.

Chapter 3

Related Work

In this chapter, related work in the trip planning domain is presented. The user interfaces for trip and transportation planning are important topics in human-computer interaction. While the algorithms behind the applications can be impressive, they are not of much use without a potential user being able to interact with them in a simple and effective way. The primary stakeholders of these applications are people that are planning a trip (at a small or large scale) and need to get from point A to point B, and beyond. In some scenarios, the user may be unfamiliar with the city they are in (such as a tourist) and are under stress because they might miss their connection to their destination. While an application should be easy to use in normal conditions, it is in these extreme situations where an intuitive design is critical. Public transit, such as bus or rail transit, is prevalent, but other facets are also researched, including ride-sharing [11] and tourism planning (walking tour) [12, 13]. Section 3.1 covers trip planning solutions for a variety of travel modes. Recent work is discussed in Section 3.2. Trip planning solutions, schedule creation, and region-specific are all presented here. Finally, web scraping is covered in more detail, including its applications in previous work, in Section 3.3.

3.1 Trip Planning Overview

There are a variety of publications in this domain, especially with route planning for local public transit networks. An early work in this field, at the dawn of the information age, is Peng and Huang's Web-based transit system with integrated GIS map [14]. At the time, riders were limited to printed schedules which could be hard to interpret and offered little in the way of service disruption notifications. The online service could pre-calculate shortest paths every 30 minutes so the users would always have up-to-date information about their route. Google Maps, nearly ubiquitous with public transit planning now, did not launch their first transit-enabled city (Portland, Oregon) until 2005 [15] and slowly expanded. This only demonstrated the need for novel or specialized trip-planning research. Later work expanded on the public transit routing idea by changing the mapping system interface, including point-and-click origin/destination selection, and incorporating a different city [16].

In the area of transportation and trip planning development, there have been three major accomplishments that increase the user's ability to accomplish their planning goals. First, the democratization of mapping and transit data, or General Transit Feed Specification (GTFS) data, has allowed numerous applications to integrate local bus or train information into their services. This is largely thanks to Google for pushing transit agencies to provide their data in this new specification [17]. The early work in this domain relied on transit authorities providing stop, route, and schedule data or the researchers would obtain the information themselves. With the GTFS, agencies could now provide their data in a standardized format for use in the wide range of applications soon to come. Google now has over 800 United States transit authorities in their transit planner, and more worldwide [18]. The second major accomplishment for transit/trip planning is Google Flights (and, similarly, Kayak, Trivago, etc), the online service to research and book flights across the world. Instead of potential travelers searching multiple airline websites for a price, it's available in one place with a simple user interface and near-instant results. It also offers recommendations for different travel dates if prices are lower. While this tool focuses on flights, the third major achievement comes in the form of car-based trip planning with sites such as Roadtrippers [19] or Furkot [20]. Users can plan their ideal road trip by inputting whichever destinations they want to visit. As they add more places, they receive recommendations for lodging, food, and tourist attractions within a customizable distance off the main travel path. There are some elements to the website that can be unintuitive, and Roadtrippers charges a fee for more complex trips. These sites, however, and those like them, allow users to accomplish their goal and are products and accomplishments of the field.

3.2 Current Work

In this field, research generally focuses on trip routing algorithms with a UI element accompanying it [12, 13, 14, 16]. Other work consists of requirements gathering and user studies to assist with the development of transit/trip applications.

The Transportation Research Board of The National Academies released a program report for an intercity travel planner in 2015 [21]. They note that there is no single place to gather multi-modal ground transit information. Whereas airlines have few destinations in the United States, the combination of bus and train stops is a much larger number. The intercity trip planner acts as the single place for users to plan longer trips. The interface was designed, schedule data for the test region (the northeast portion of the country) was loaded, and stakeholders reviewed the application. Despite its positive response, the primary issue with this tool, or those similar, are private transit companies refusing to make their schedule data publicly accessible (in GTFS form). Reviewers pointed out that some suggested routes were "nutty" and asked the rider to make unreasonable connections when a simpler service was in place. It is possible this is due to the application not having those services' transit data, but the authors do provide a plan to mitigate this issue.

In a 2016 paper, Zhang et al. gathered requirements for a collaborative trip planning application [13]. Groups of users who were planning to travel were given diaries to record individual planning processes, such as information gathering (ex. search engine versus tourist applications) and group discussion (ex. face-to-face or email). With consistent updates to the diaries, the researchers were able to glean some design implications from the users studied. Regarding presentation of information, they constructed a top-down approach, starting with an overview of the point-of-interest, then transportation details, and finally a more detailed background of the location. Or, with intra-group communication, they believe that each member's travel preferences can and should be visualized, allowing others to view their opinions and potentially vote on itinerary changes. The authors planned to validate the proposed requirements.

Elliott et al. completed similar work, but with the One Bus Away system requirements [22]. In this research, the application was already developed, and users were studied when completing a pre-defined task. The results of the study were primarily focused on errors, either system or human/intuitiveness, that arose during testing. A significant issue was the specificity required to search for an address. If the user inputs it incorrectly, the app does not have a way to offer "Did you mean" suggestions. However, in other areas, users provided suggestions to increase the clarity of the application, as another issue was the representation of bus, route, and stop data (all as numbers). The research for One Bus Away provides information as to what users look for when using this application, and, by extension, those like it.

Svangren et al. evaluated user perceptions in their study on ride-sharing services [11]. While the majority of the data gathered from users related to perceptions on planning trips, driver negotiations, considerations when selecting a driver/car, and handling uncertainty in trips, the authors did discuss HCI implications from this information. Multi-platform and multi-device recommendations were made, as they found users liked the flexibility and options available from multiple services but wish they did not have to check all of them to find the best deal or type of ride they were looking for. The multi-device paradigm, they note, becomes especially important when users move from ride-sharing to public transportation in the same trip. The security and privacy aspect was also discussed, but focused on information sharing (or lack of) prior to a ride due to social media usage.

Multi-modal trip planning in Greece was explored by Zografos et al. in their research [23]. While this is older research from 2008, it is similar to Adler's research [21] in that it is multi-modal, incorporates local and regional transit, and provides users with a completed itinerary. However, the system only supported Greece at the time of publication. This research does go beyond Adler's work in that a comprehensive survey was issued to a key demographic (travelers at important Greek hubs) prior to system development. With this, they created functional requirements and designed a prototype, which was accepted by a secondary user study.

There are some more works and available options in the area of schedule creation. One example is the "Intelligent Traveling Service" from Navabpour et al. [24]. Their work is a travel planning service combing air, bus, and rail transportation. They did use a REST service with OpenKapow to retrieve Amtrak results for rail portions of the search. However, this paper was published in 2008, and the Amtrak service, along with OpenKapow, is now defunct and unreachable. However, the data returned from the service is similar to what the Rail Planner retrieves from Amtrak's search results. Other itinerary planning research has been conducted but does not necessarily relate to train travel. Roy et al. proposed an interactive itinerary planner focused on points-of-interest (POIs) in a location [25]. Users provide feedback on the proposed plans and the program creates new itineraries based on their feedback. In the context of the USA Rail Planner, this would be akin to users providing a list of cities to visit and the application generating potential routes and trains to take. Given the overhead of performing an Amtrak search, even with an automated browser, generating an itinerary could be very lengthy without an existing set of local timetables to run against. However, with existing timetables, research such as Witt's TripBased algorithm for Pareto-optimal journeys in a public transit system [26] could be used. With timetables for the public transit network and some preprocessing time to precompute transfers, it can produce a set of journeys which minimize arrival time and the number of transfers, wherein the user will reach their destination. The prohibiting factor would be finding a reliable, up-to-date, and parsable source of Amtrak timetable data. The first two requirements are fairly simple but ensuring the application can understand the schedule data could be a challenge.

3.3 Web Scraping in Action

Web scraping is not a new field, but there does not seem to be much, if any, existing research using this technology with Amtrak's site. Gheorghe et al. discuss methods for scraping data from websites with varying layouts, such as pure HTML or with the inclusion of Javascript elements [27]. They discuss the use of Selenium, as is used in the application, and its automation capabilities with modern web browsers and element parsing abilities. However, web scraping is not limited to Amtrak's website. It is often the technique chosen when APIs are limited or nonexistent (in Amtrak's case). For the former case, Twitter is a prime example [28]. Twitter provides an API but limits how many requests can be sent to the API per application (i.e., per API token, a unique identifier). Hernandez-Saurez et al. instead opted to use a web scraper with the Twitter search function and the returned HTML content for each query. In doing so, they were able to retrieve more tweets in less time, surpassing the performance of Twitter's own API. They also built a web service to act as a GUI for their scraping solution. Similarly, Instagram's API allows user profile content retrieval but has restrictions on its use. Himawan et al. proposed a web scraping solution that can bypass the API [29]. They include a media crawler to collect the data, storage of scraped data in a NoSQL database, and a Flask application acting as the interface between users and the underlying service. Overall, web scraping bridges the gap between functional (and comprehensive) APIs and manual data collection and is vital to the Rail Planner's functionality.

Chapter 4

Software Specification and Design

In this chapter, the application's software design specifications are covered, which guided its development. Section 4.1 details the requirements imposed on the system. Functional requirements, describing application functionality, and non-functional requirements, covering system design constraints, are both here. Use cases are shown and discussed in Section 4.2. Next, the application architecture is discussed in detail in Section 4.3. A high-level overview of the application is provided alongside its behavior behind the scenes (UI design is discussed in Chapter 5). Finally, technologies used by the application are briefly documented in Section 4.4.

4.1 Requirements

The Rail Planner had numerous initial requirements which were at the core of its planned functionality. In the first subsection, functional requirements are discussed, which detail key features of the application. Some were critical (level 1 requirements) to the prototype while others were "nice to haves" (level 2 and 3 requirements) and may not have been fully implemented in the final version. In the second subsection, non-functional requirements are reviewed, which are not user-facing functionalities but rather system-level requirements when considering its design. Some requirements were gathered based on the responses in an informal survey early in the application research and development, posted to the Reddit subreddits r/Amtrak (Amtrak enthusiasts, riders, and general discussion) and r/SampleSize (users can post their own surveys with the intended demographic and others will fill them out). The questions are listed in Table 4.1.

Table 4.1: Survey Questions, March 2022

Number	**Question**
Q1	What are your thoughts on Amtrak train travel?
Q2	Have you purchased train tickets through Amtrak's website in the past?
Q3	How often do you travel by Amtrak train?
Q4	What would the majority of those train trips be classified as?
Q5	Are you aware of Amtrak's "Rail Pass" and what it is used for?
Q6	Have you ever purchased and used a Rail Pass?
Q7	If you answered YES, how did you plan/formulate your trip itinerary?
Q8	When planning a multi-segment/city Amtrak trip, how do you determine where to go?
Q9	When taking a multi-segment/city Amtrak trip, what is your final destination?
Q10	Were an application available to plan a multi-segment/city Amtrak trip, would you use it to assist in some or all of your planning?
Q11	What kind of features would you look for in an application that would assist in planning a multi-segment/city Amtrak trip?
Q12	What are your thoughts on train trip planning?
Q13	When selecting a station for the origin/destination, what would you expect this feature to do and/or look like?
Q14	When viewing a list of resulting trains from an in-app search, what would you expect this feature to do and/or look like?
Q15	When viewing a list of saved segments (selected from earlier searches), what functionality would you expect from this feature?
Some potential features were discussed.	
Q16	Now that you know some of the included features in this application, are there any that should be thought about or approached differently?
Q17	Now that you are more familiar with the application, are there any other features you would want to see? Are there any you wouldn't?
Q18	How do you normally access the internet and/or complete everyday technological tasks?
Q19	What is your preferred web browser?

4.1.1 Functional Requirements

Functional requirements are a list of features and functions implemented in the prototype. Level 1 requirements are fully implemented and function as expected and are considered the core functions of the application. In the context of the Rail Planner, its core requirements relate to accomplishing the goal of creating an itinerary. Level 2 requirements are often extra features that are not essential to accomplishing the goal but provide quality-of-life improvements or additional amenities for the user. And, whereas level 1 and 2 requirements are fully or mostly implemented, Level 3 requirements are either partially implemented and may be non-functional or could be classified as future work, discussed in Chapter 8.

Level 1 Requirements

There are 13 level 1 requirements which describe essential application features, listed in Table 4.2. The first and second requirements detail the station selection feature, where users can choose their origin and destination locations from dropdown menus. Search functionality is included in each dropdown, meaning users can type some letters and press the Tab or Enter key to view all entries containing the entered string. A "Swap" button allows users to easily interchange the origin and destination selections if they needed to include a round-trip segment or change the most recent destination to the new origin. The third and fourth requirements relate to the departure date selection. Users can click the "Select Departure Date" button or the label of the currently selected date to spawn a calendar area, which allows for a new date selection. Clicking either "open" button or a new date on the calendar will close the popup. Alternatively, users can use the plus and minus buttons to increment the current date by one day to save the time of opening a calendar when only minor date changes are necessary. The fifth, sixth, and seventh requirements establish how train searches work. Clicking the "Find Trains" button launches the background web search, updates the search header (from "here" to "there" on some date), creates a progress bar, and the status bar reflects the current search actions to indicate the

Table 4.2: Level 1 Functional Requirements

Number	**Priority**	**Description**
FR01	1	The user will be able to select the origin and destination from respective lists of stations, with the ability to search in the list if necessary.
FR02	1	A Swap button will, when clicked, swap the origin and destination selections.
FR03	1	Clicking the "Select Departure Date" button or the current date label will display a calendar selection popup. If the popup is already visible, clicking either will close the popup.
FR04	1	Clicking the increment (+/-) buttons should increment the current date up or down by one day, respectively.
FR05	1	A "Find Trains" button, when clicked, will begin the (background) search for trains based on selected origin, destination, and departure date. It will also create a search progress bar.
FR06	1	A heading area for the current search results, with "<Origin>to <Destination>" and the date, will appear when a search is initiated. If already present, it will update with new information, if any.
FR07	1	A status bar will be present at the bottom of the application, which normally displays "Ready." During a search, details of what is occurring will display to signify something is happening.
FR08	1	An area for the train search results table will be visible, and, when a search is completed successfully, will populate with results and a scrollbar, if necessary.
FR09	1	Clicking a train in the results area will reveal an option to "Save Segment" so it can be recorded to the itinerary.
FR10	1	The user will be able to export the itinerary (of some n saved segments) to a spreadsheet.
FR11	1	If the Amtrak search returns an error (no service between stations, no trains on this day, etc), the user should be notified with an error popup box containing the contents of this message.
FR12	1	Search validation should be performed in regards to station selection and departure date.

program is functioning properly. As trains are found, the "number of trains" label will update accordingly. The eighth and ninth requirements designate functionality for the results list, which populates with all the trains found in the search. Clicking a train allows the user to save the segment to their itinerary, which ties to the tenth requirement: users can export their itinerary as a spreadsheet. While the application does not allow the user to book tickets directly, nor is it the intention to do so, the saved itinerary provides groundwork for later booking. Requirements 11 and 12 involve search handling. Searches must be validated before starting any web search. For example, the origin and destination cannot be the same, or the departure date should be greater than or equal to the most recent segment's arrival date. Secondly, once a web search is started, if an error related to the search is returned, the user should be notified as to what went wrong and how it could be resolved.

Level 2 Requirements

The eight level 2 requirements were all implemented but do not drastically change the core features of the application, and these are displayed in Table 4.3. The 13th and 14th requirements note how photos of the selected stations' cities appear and their functionality. As users change their origin or destination cities, the associated photos are found online and displayed. Alongside the updated photos is the ability to click on either one to bring up that city's tourism page, or the application's best guess as to what it is, as not every city has an official page. The 15th requirement describes navigation through past searches with left and right arrows above the results list. Requirement 16 relates to the results list in that the user can choose which columns (name, arrival date, price, etc) are displayed. The itinerary window and its functionality is described in requirements 17 and 18. The "View Itinerary" button will, unsurprisingly, open the itinerary window where users can view their saved train segments. In this window, they can delete or move segments up/down in the list. Clicking a specific segment allows the user to recall that segment's search results in case they wanted to select another travel option from that search. The 19th and

20th requirements designate how the map window works. When selecting origin and destination stations, the map will put markers down for each and draw a line between the two. However, when a search is performed, right-clicking a train and selecting the Route Map will redraw the map with the train's route and intermediate stops, with emphasis placed on the origin and destination. The last level 2 requirement notes that users can save their progress in the application and load it later, instead of starting from scratch. This saves their past search results and itinerary.

Table 4.3: Level 2 Functional Requirements

Number	**Priority**	**Description**
FR13	2	City photos will update themselves to a different location based on any new user station selections associated with the respective image.
FR14	2	Clicking on a photo of a city should bring up information about the city, such as its tourism page.
FR15	2	Left/right arrows in the heading area will allow the user to move between all searches performed while the application is open, in case they need to check previous results.
FR16	2	The user should be able to select which train attributes (columns such as departure time, price, travel time) are displayed in the search results table.
FR17	2	A popup window to list selected segments should open after clicking a "View Itinerary" button.
FR18	2	The itinerary window should allow the user to reorder and delete segments, as well as view their search results.
FR19	2	Clicking on the name of a train from within the results table should update a map window with the train's route and intermediate stops.
FR20	2	A map of the United States with pins/markers for the origin and destination should be displayed and dynamically updated as station selections change. A line should be drawn between the two points.
FR21	2	The user should be able to save and re-open planner files into the app for further modification.

Level 3 Requirements

There are few level 3 requirements, and almost all have not been implemented but would be useful for future iterations, shown in Table 4.4. The most advanced requirement is enabling the application to determine travel dates for the user solely based on a list of stops. This would likely require schedule data to link trains (routes) to stations. Schedule data is a part of the second level 3 requirement (23rd requirement overall). Amtrak provides public General Transit Feed Specification data which contains routes, stops, and other pertinent train data. Parsing the data into a usable form and incorporating a routing algorithm would eliminate the need for a web search in most cases. If arrival times were ignored, only a list of stops would be necessary, which would be easier to implement. Such algorithms could be adapted from related work, such as TripBased [26] or RAPTOR [30], or using a heuristic search algorithm like A-star [31] (adapted from Djikstra's algorithm). This can be considered the most useful level 3 requirement. Third, and finally, displaying station information such as distance to the city center or public transit options (partially implemented) is another requirement. Future level 3 requirements, not listed, will be explored in Chapter 7.

Table 4.4: Level 3 Functional Requirements

Number	**Priority**	**Description**
FR22	3	Allow the user to input each stop on their journey, and let the program figure out the travel dates.
FR23	3	Use GTFS and schedule data to perform searches instead of the online website.
FR24	3	Display information related to each station, such as distance to the city center, availability of parking, or other transit connections.

4.1.2 Non-Functional Requirements

The non-functional requirements define a set of system operations and constraints. As opposed to functional requirements defining application features, non-functional requirements describe those that impact the system design. All requirements are listed in Table 4.5 and are discussed in the following text.

Table 4.5: Non-Functional Requirements

Number	**Description**
NFR01	The application will use the Python programming language.
NFR02	The application will use a Chrome webdriver controlled by Selenium.
NFR03	The application will run on Windows and MacOS.
NFR04	The application will be packaged as a standalone executable for Windows.
NFR05	The application will require internet connectivity to perform searches.

The first non-functional requirement stipulates that the application is built in the Python programming language. While this is not a requirement for an application of this type, it is due to the ease of use of the Tkinter GUI library to create the interface, as well as the web driver integration via the Selenium library. In fact, Selenium and a Chrome web driver is requirement two. A Chrome installation is required to run the application, and without one, no searches can be performed. Requirements three and four detail the target operating systems for the application, Windows and MacOS. However, a standalone version, created as a “click-once” executable with the appropriate Python binaries and libraries packaged within, will only be created for Windows. Finally, requirement five stipulates that the application requires an internet connection to work, as this is necessary to perform searches, load images, and populate some station data on the map. It is possible to load a saved planner file and manipulate itinerary data, but any new searches cannot be performed.

4.2 Use Case Modeling

In this section, a use case model is showcased for the application and each use case is described from Figure 4.1. Item colors correspond to their associated functionalities. While there are certainly more use cases available, the essential features of the application are included. As the Rail Planner is a single-user application, there is only

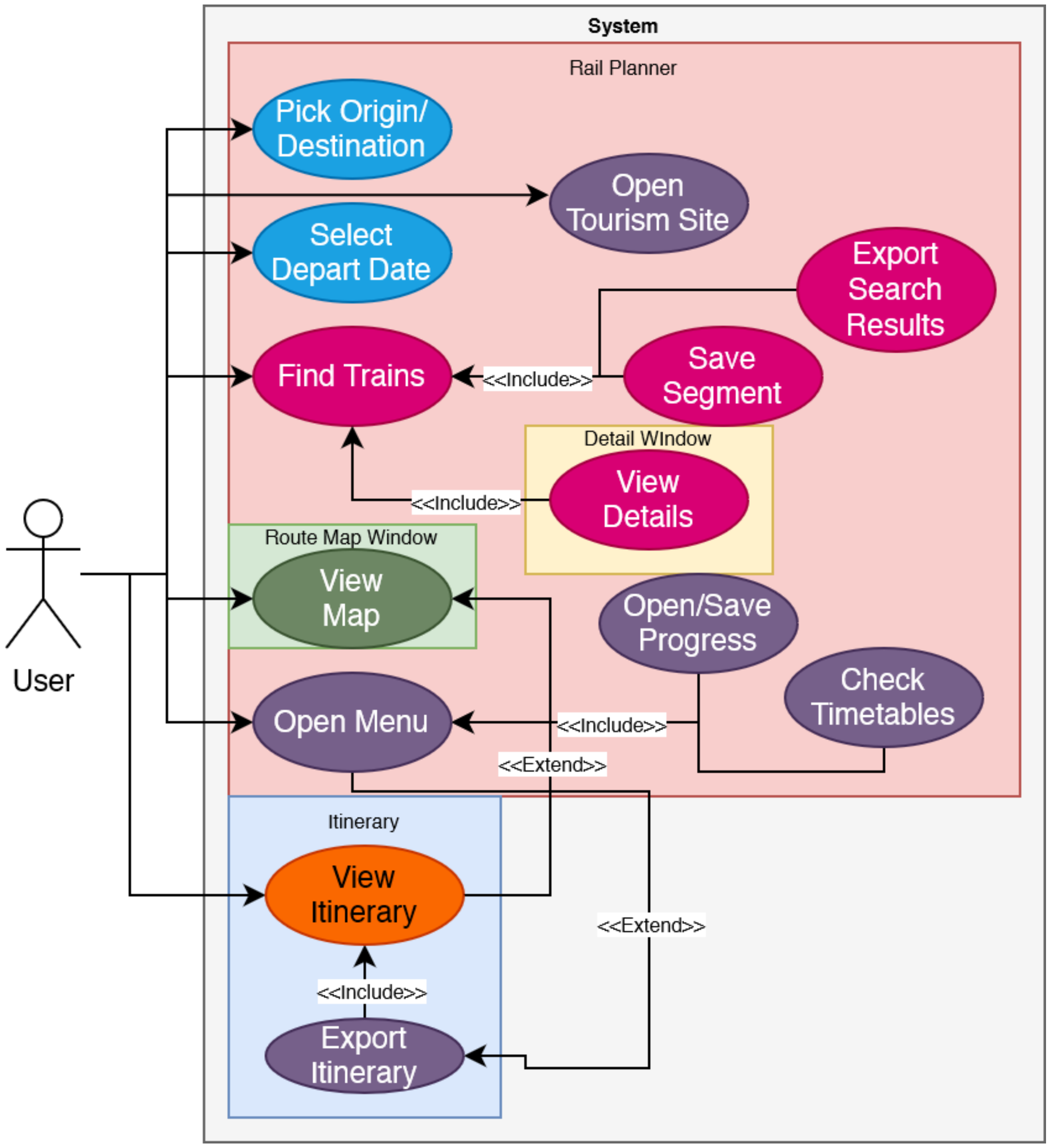


Figure 4.1: Use Case Diagram

one actor: the traveler (user).

As the traveler begins their planning, they can choose their origin and destination stations to be used as search parameters. The second use case lets them select the departure date to complete all the required search parameters. The third use case is available based on the origin or destination station selections, as clicking the associated photo opens the tourism site for that location. Fourth, users can find trains (perform a search). Only after this is done does the application allow the user access to the fourth, fifth, and sixth use cases. They can save a segment to their itinerary, export the search results to a spreadsheet, and view additional details about each result in the detail window. The sixth use case is viewing the route map, which could be the origin and destination, the route of a selected train, or their complete journey map. The seventh use case is opening the menu, which enables the user to save (or load) their progress (8) and view timetables for specific trains (9). The last two use cases are within the itinerary window, one that lets them view their itinerary (and potentially make changes to it) or export it to a file. As mentioned, the route map window can display a journey map when called from the itinerary window.

4.3 Architecture

In this section, a closer look at the application's design is presented. A high-level overview of the structure of the application is discussed. Information on classes and their objects and methods are covered. Then, the application behavior is examined in detail, from initialization on startup to the many features included in its main areas. Information about the backend methods of specific functionalities is described. Finally, the searching process is touched on, including its transition from a naïve approach to a more resilient searching platform.

4.3.1 High-Level Overview

The Rail Planner has five GUI windows available to the user, notated by circles, which interface with a host of helper classes in the application, shown as rectangles, and are all seen interacting in the system-level structural diagram presented in Figure 4.2. GUI windows are circles, while class objects are rectangles. The Main Window holds most interactable elements, and, as the "parent" window, can control its children top level windows.

Application startup primarily waits for the initialization of the Image Searcher

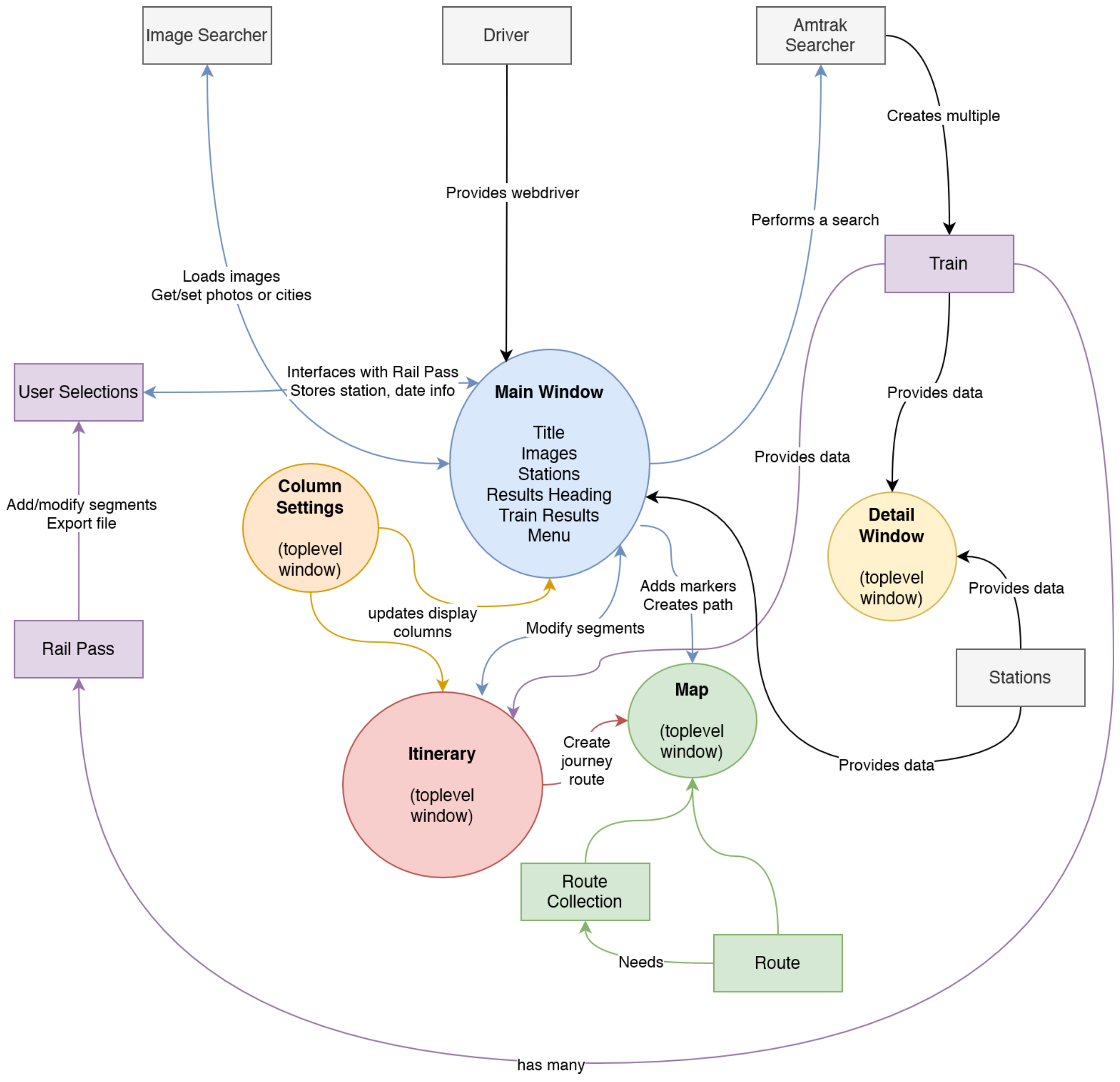


Figure 4.2: System-Level Structural Diagram

and Stations classes, as their startup blocks the application until they finish. The Image Searcher is responsible for retrieving station (city) images for the Main Window using a web driver from the Driver. Stations, similarly, retrieves all Amtrak stations and provides numerous helper functions to provide various parts of the station data, such as a “friendly” display string, its code, connecting trains, or city and state.

The Amtrak Searcher creates a thread at startup, after the blocking functions, so it can initialize in the background when the Main Window is opened. Its function is simply to perform a search and provide the train information found, if any. Those results are transformed into Train objects and are available to retrieve data from later on, which is especially useful when it comes time to export the itinerary. Train objects provide data to the search results table in the Main Window, the per-train information in the Detail Window (alongside Stations with its information), and results from each search are saved to the Rail Pass object. By using the Train object across multiple facets of the application, handling train data becomes much simpler with one object type, and thus the concern becomes how to store the objects.

The Rail Pass class is the primary data storage means for the application. The user's itinerary and all search results are saved in this class. While it may have made more sense to use a "proper" database structure, it was implemented for the initial prototype creation and worked, and thus was kept as-is in favor of introducing other application features. The benefit of this consolidation into one class is the ease in saving and loading user progress (`.RailPlan` files) as the class is simply pickled and un-pickled, respectively, using Python's built-in `pickle` module. These RailPlan files are lightweight, generally increasing in size at roughly 11 kilobytes per search performed, depending on how many results are found. The Rail Pass also handles file exports, such as the user itinerary or search results. The User Selections class, originally meant to be what the Rail Pass is, holds the Rail Pass and user "preferences" such as station selections, departure date, and display columns for the results table and Itinerary.

The remaining elements are top level GUI windows. Column Settings is a simple popup that asks the user which columns they would like displayed in the results table and itinerary, such as departure date or coach class price. This interacts with the User Selections class and update any visible tables. The Itinerary holds the user's saved trip segments and can modify their order or delete them, affecting the Rail Pass. Critically, this window also has the export functionality required to save the user's itinerary to a spreadsheet file. Finally, the Map window displays a map of either the selected stations, a specific train's route, or the entire journey (via the Itinerary). It interacts with the Route and/or Route Collection classes to glean required data (path coordinates and stops) to display a train's route and can also retrieve station addresses from the Amtrak website, which are converted to coordinates.

4.3.2 Application Behavior

In Figure 4.3, a behavioral diagram is shown with the most important behaviors of the Rail Planner. Smaller features, such as some individual menu options, were excluded. Arrows with a solid line are blocking functions, while those with dashed lines are non-blocking functions. A legend in the bottom-right describes the color scheme by application elements.

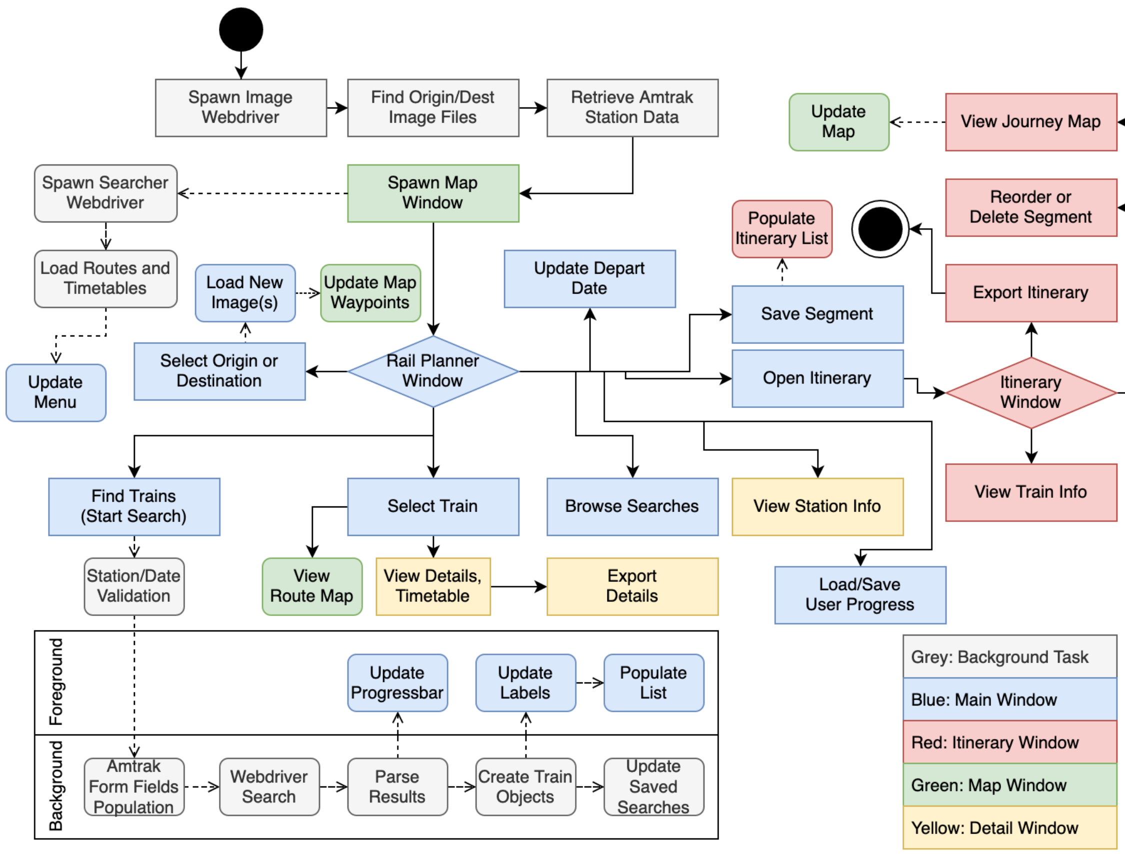


Figure 4.3: System-Level Behavioral Diagram

Initialization

The user will launch the application which begins the startup processes. Since web drivers are used in the application, the user must have an up-to-date Google Chrome browser installed on their system. The application will download the appropriate web drivers if they are not present, but they will be for the latest Chrome version. Any errors with this process are displayed to the user, but the application will quit if it faces web driver issues. An image web driver is created, which finds the origin and destination images to be displayed in the main window. Google Images is used for obtaining these photos. The web driver will query the site with the city and state, find the first returned image, and attempt to convert it from base 64 or find the full-resolution version. The image is turned into a Tkinter `PhotoImage` object the application UI (Tkinter) can display and is cropped to a reasonable size. Amtrak station data is then retrieved from Wikipedia [32], in an effort to be up-to-date and not rely on a static list of stations. While new or decommissioned stations are not necessarily a common occurrence in the Amtrak network, long-term use was the goal for the application, which means providing users an accurate station list. For example, Amtrak plans to restore train service along the Gulf Coast, which was previously suspended due to Hurricane Katrina [33]. Stations formerly served, such as Pensacola, Florida, are not in the "Active" train station table on Wikipedia but would be moved from the "Suspended" table as appropriate, whereas a list included with the application could not react to this change as easily. Alternatively, station data could be retrieved from the browser's session storage when visiting the Amtrak website (`stationsData_stations`), but the breadth of Information available from the Wikipedia list, as well as its discrimination between train-served stations and thruway motorcoach stations, meant Wikipedia was the preferred method. Next, the map window is created, which, at this point, will have a line drawn between two markers on the map: the origin and destination stations. After the map window is created, the main Rail Planner window is shown, and the application is ready for use. Simultaneously, a thread is created to spawn another web driver, this time for the Amtrak searching functionality. Using this web driver after it navigates to the Amtrak website, a list of trains can be extracted from session storage (`traincodes`) and parsed for unique values. Then, with those trains, URLs are created that can be launched in the user's web browser to find timetables. Since Amtrak no longer provides PDF timetables for their routes as of June 2018, the application relies on DuckDuckGo's "Ducky" search (automatically redirecting to the first result) and a specific search term: "`<Train Name>` train timetable schedule Amtrak filetype:pdf". In testing, a reasonably high success rate was observed with DuckDuckGo usually finding the correct schedule. With URLs created, train names are added to a new Timetable menu and are available for the user.

Main Window

With the main Rail Planner window now present, the user can begin their trip planning work. There are a multitude of functions available to the user, but they will be presented in an order similar to how the user would interact with the system. While progress can be loaded from (or saved to) an existing `.RailPlan` file, it can be assumed the user will not have any saved work on their first use of the application. When changes are made in the application, an asterisk (*) will appear in the title bar of the main window to denote unsaved work. Exiting the application with unsaved work will prompt the user to save their changes, discard them, or cancel the exit operation.

Selecting origin and destination stations will likely be the first thing the user will do. As a new station is picked, the application will ask the Image Searcher to find a photo of the new station's city for display, and the appropriate image is updated. Additionally, the map will use the new station's code to find the address from Amtrak's site, which is converted to coordinates and used to update the appropriate marker. Source code for retrieving Amtrak coordinates is available in Appendix A. Finally, the path between the station markers is redrawn and the map view is recentered on the two points. The user will then choose a departure date, either by using the plus and minus (increment by day) buttons, or the calendar "dropdown" feature. They cannot choose a date earlier than the current day. Now, with the origin and destination stations selected and the departure date chosen, the user can start their search.

Specifics of how searches work are discussed in Section 4.3.3.

With the search results table now populated, the user has multiple options available to them. Left-clicking any train result allows them to save a segment to their itinerary. However, by right-clicking an entry, the user can select multiple options relating to that specific train. They may want to view its timetable or view Amtrak's official page relating to that train (this URL uses a standard format and can be generated easily: `www.amtrak.com/<the-train-name>-train`). They might view the train's route map, which clears the map window of any existing markers or paths, then uses included `GeoJSON` data from Transitland [34] to draw a *true* route: one that follows the train tracks, and not just a straight line between two points. The user may also be interested in details about the train or its destination, in which a separate Detail Window can be launched. Both Detail Windows (station or train) can have their information exported as HTML, JSON, or text files. After the user has performed multiple searches, they can review earlier results without needing to find them again, saving a significant amount of time. This can be done with the "<Previous Search" and "Next Search>" buttons, which navigate the searches in chronological order. Previous searches are also included in saved `.RailPlan` files.

Itinerary

The most important part of the user's trip planning experience is the Itinerary, which can be opened from the Main Window. There are a variety of functions available in this window, but the most important is the export feature, which allows the user to save their trip itinerary to a spreadsheet file, which signifies the user's goal (endpoint) in the diagram and application use. They can also modify segments by reordering or deleting them. The right-click context menu from the main window is also available in this Itinerary window, so users can view additional train information about their saved segments or return to their search results. Finally, they can view their entire journey in the map window, which adds sequential numbering to the displayed stations.

4.3.3 Searching for Trains

When the user initiates a search, multiple preparatory tasks are done before the searching begins. This starts with data validation for the user's departure date and station selections. There is only one failure state for data validation, otherwise the user is instead asked if they want to continue if any errors arise. For example, they may be starting a search with the same stations or departure date selections as they had with a previous segment, thus prompting for confirmation of their search. The only condition that would prohibit a search is if the origin and destination stations are the same. If the search is to continue, the Amtrak Searcher is provided pertinent search data, such as stations and date, as well as the UI elements to control: the progress bar and "number of trains" label. With all pre-search tasks complete, the application will then start a thread for the search.

Algorithm 4.1 Searching and Parsing Method

```
Initialization
Load Amtrak search page
Input origin station
Input destination station
Input departure date
Click search button
Search handling
if search results found then
    if scraping method in use then                    ▷ By default, False
        Check every page
        Gather page results data
    else
        Find session storage data
        Parse train results
        if parsing fails then
            Use scraping method
        end if
    end if
    return search data
else
    return error message
end if
```

A high-level overview of the search algorithm is shown in Algorithm 4.1. A sample of the `AmtrakSearcher` class source code relating to searches is in Appendix B. Any residual data from previous searches are cleared from the class and the web driver loads the Amtrak search page (`https://www.amtrak.com/tickets/departure.html`) and scrolls to the top if it is not already loaded. However, if the previous search returned any errors, the page will be reloaded regardless. The "New Search" button is clicked, and the web driver waits for the form elements to appear, then fills in the origin station code, departure station code, and the departure date. Uppercase station codes were used as they will autofill after sending a Tab key, and the search will not be allowed to start unless the stations are "properly" inputted (meaning Amtrak recognizes them as a station, not just inputted text). Finally, the web driver waits until the "Find Trains" button is enabled and clicks it, thus starting the search on Amtrak's website. Errors may occur for any given search, such as no trains running on a given day or a lack of service between two stations. Were this to occur, the application will capture the error message and present it to the user as a warning, and they will need to restart their search after adjusting their selections. However, if no errors are returned, then the application will assume that trains have been found, and it will begin to parse results.

Listing 4.1: Locating an element by XPath with Selenium

```
x = driver.find_element(by=By.XPATH, value="//button\
[@aria-label='FIND␣TRAINS'␣and␣@aria-disabled='false']")
```

The original method for parsing results was tedious to implement and prone to failure if Amtrak changed their website implementation or layout. This is due to its heavy reliance on XPath, a syntax for defining parts of an XML document (in this application, the web page), for element location. In combination with Selenium, it can be used to find one or many elements that match a given XPath or search string. Such an example is Listing 4.1, where the driver is finding an button by XPath that is labeled "Find Trains" and is not disabled. For the first iteration of the search parsing, each required element in the search had to have its corresponding XPath manually located and added to the searching logic. A multitude of try-except blocks were present to handle any number of errors that could be encountering when scraping for data in this manner. Often, the data would be in a “human friendly” form but would require additional processing and cleanup when loaded into the application. This also required logic for a variety of formats the data could appear in while searching. It was incorrectly assumed the elements (and their XPaths) on the page would not change. However, as application testing continued, the names of a significant number of elements changed, and each problem element needed to be re-checked and added back to the search logic. The primary issue with the web scraping method was its speed, often consuming almost five seconds to scrape information per page of results.

During the time of web crawling and XPath search implementation, an experimental approach was underway, focused on obtaining search data while reducing or eliminating the use of a web driver. Determining the endpoint to which Amtrak form data was submitted was the first approach. This was done by analyzing network traffic at both the browser level and interface (Wireshark) level. However, due to the data encryption and obfuscation, this method was deemed unfeasible. Another method involved finding request headers for a search and attempting to modify and resubmit them, but the aforementioned challenges were also present with the headers. The final approach was examining browser session storage before and after searches, and this was where progress was made. After a search is completed, Amtrak stores a dictionary in the storage (`searchresults`) containing information about the results. Since the application is parsing a dictionary object (with its associated built-in methods), results are obtained in a fraction of a second once the session storage data is retrieved. In fact, the dictionary provides a wealth of information compared to web scraping with Selenium and enabled the addition of the Detail Window. Data could now be extracted at a per-segment level as opposed to viewing each result holistically. Where the web scraper would report a multi-segment result as “Multiple Trains” or “Mixed Service”, the new method could examine each segment and find information such as remaining seats, individual departure/arrival times, and on-board amenities, with the last item being a highly requested feature during initial requirements gathering. The "Train JSON" method of gathering data is the application's preference, however, as seen in Algorithm 4.1, it will fail over to the old web scraping method if parsing encounters an issue.

As the searches are conducted, the status bar at the bottom of the main window will update with what the application is currently doing, such as "Searching – entering travel dates" and will either revert to "Ready" when the search is complete or "Error" if something went wrong. Such examples of errors include failure at any point in the searching process, such as web driver failures or search problems (no trains between these two stations, no service on this date). Additionally, the progress bar that appears during searches will also update corresponding to search progress. If the search is successful, results are examined to ensure none of the trips are sold out. If they are, they are excluded from the returned entries. Finally, all data is sent to the results table while also being saved in the `RailPass` class.

4.4 Technologies

The Rail Planner, at its core, is a user interface heavy application, with a large part of the background tasks consisting of data gathering and manipulation and does not necessarily call for any specific language. However, the incorporation of web scraping, as well as familiarity and experience with the technologies implemented, prompted for their use in this application. However, an application such as the Rail Planner would not necessarily need to be made using these exact technologies, only ones that support the same functionality and UI design capabilities.

4.4.1 Python Programming Language

Python [35] is an interpreted language, so it does not necessarily enjoy the speed benefits of a compiled language, but its extensive community support and custom modules make it useful for this application. Additionally, it is a cross-platform solution, and only required minor adjustments in code for application compatibility. Modules such as `PyInstaller` allow a programmer to create packaged executables that can be run on a client machine that might not even have Python libraries installed, as it includes them in the executable. The entirety of the application's codebase was written in Python. While not deserving of their own subsection, some useful built-in modules were `json`, for parsing scraped data, and `threading`, to create threads for background tasks to keep the user interface responsive during intensive workloads.

4.4.2 Tkinter

While technically `tkinter` [36] is a built-in Python module, it requires its own section due to its breadth, both as a module and for its use in the application. It provides an interface for the Tcl/Tk GUI toolkit, which allows Python to work with operating system facilities for user interfaces, such as Cocoa on MacOS. The application uses Tkinter for all user interface elements. To incorporate a more modern look in the application, widgets from the `ttk` (themed Tkinter) submodule were used, shown in Figure 4.4. Some additional modules besides what was built-in to Tkinter were necessary to get the required functionality. TkCalendar [37] was used for the date selection area in the main window. It provides an accurate calendar widget for the user to select a departure date, with some styling options to make it look how the programmer wants. `pillow` was also used to add support for displaying image files from the web in the application.

4.4.3 Rail Map View

Another Tkinter subsidiary is the `TkinterMapView` module [38], which is a "tile based interactive map renderer widget" for Python. It uses OpenStreetMap as its default tile server, and provides a selection of functions for drawing markers, paths, and shapes on the map based on given coordinates. However, the default styling of the map (notably: marker positioning and style) was not consistent with the Rail Planner's design and ideal look. For this reason, stylistic changes were made and included with the application as `railmapview`, but functionally the module is identical. The

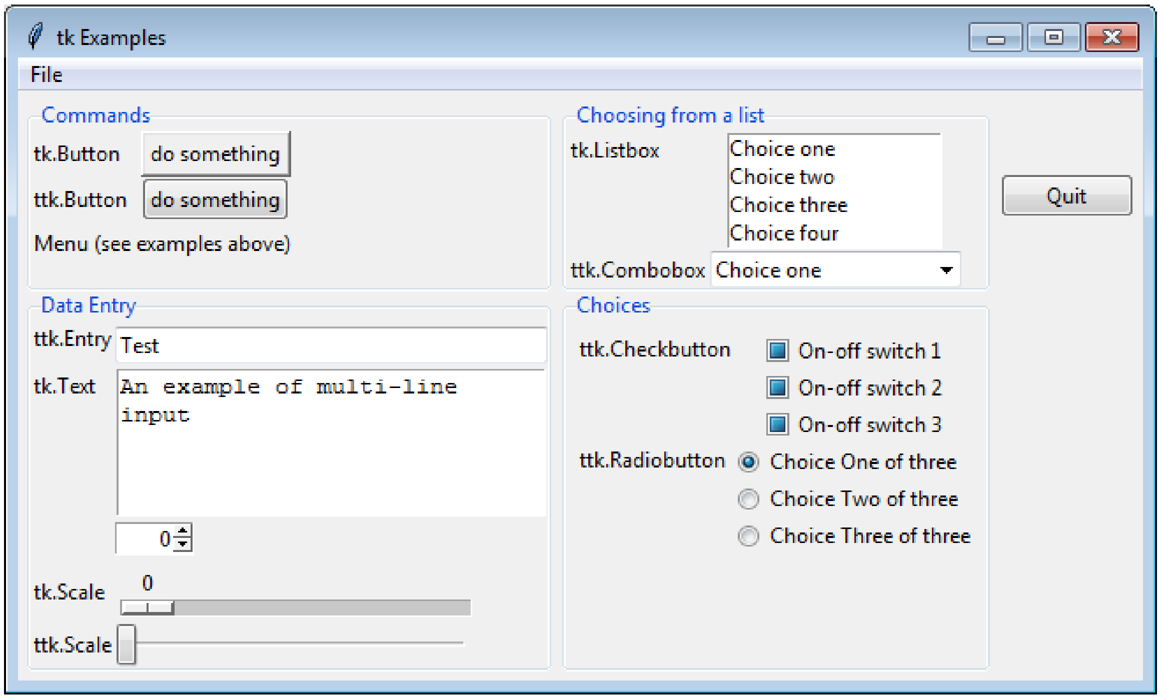


Figure 4.4: Tkinter GUI Window with Tk and Ttk Widgets.

application utilizes this module to create markers for station stops and draw route paths.

4.4.4 Selenium

Selenium automates (controls) web browsers and provides libraries for five programming languages, such as C, Java, and Python [39]. There are a range of tools available within Selenium, but the Webdriver API is what is primarily used in the Rail Planner. Both the Image Searcher and Amtrak Searcher require Selenium to control the web drivers for their respective functions. Were it not for this library, the application could not emulate user interaction with the Amtrak website, and therefore could not find any search results. The Webdriver API only provides methods to control a web driver so the correct web driver must be obtained externally. To promote the ease-of-use of the application, the `webdriver_manager` module is included, which automatically downloads and runs the correct version of the Chrome browser web driver without any interaction from the user [40]. Without this module, end-users would be required to download the correct Chromedriver (which changes with each new version of the Chrome browser) and add it to their system PATH environment variable so the application could find and use it. Additionally, a specific version of the Chromedriver is used for the Amtrak Searcher, which tries to circumvent web scraping protections on websites. The `undetected_chromedriver` [41] module is used for this purpose.

4.4.5 Beautiful Soup

The last important technology utilized is the `bs4` module which has `BeautifulSoup`, and this package can be used to parse HTML pages [42]. This has some useful web scraping implications, as using this module is much quicker and less resource intensive than a Selenium and web driver configuration. However, as it only receives HTML, it cannot handle Javascript elements (critical for the Amtrak Searcher). The application uses Beautiful Soup to scrape station information, both from Wikipedia and Amtrak.

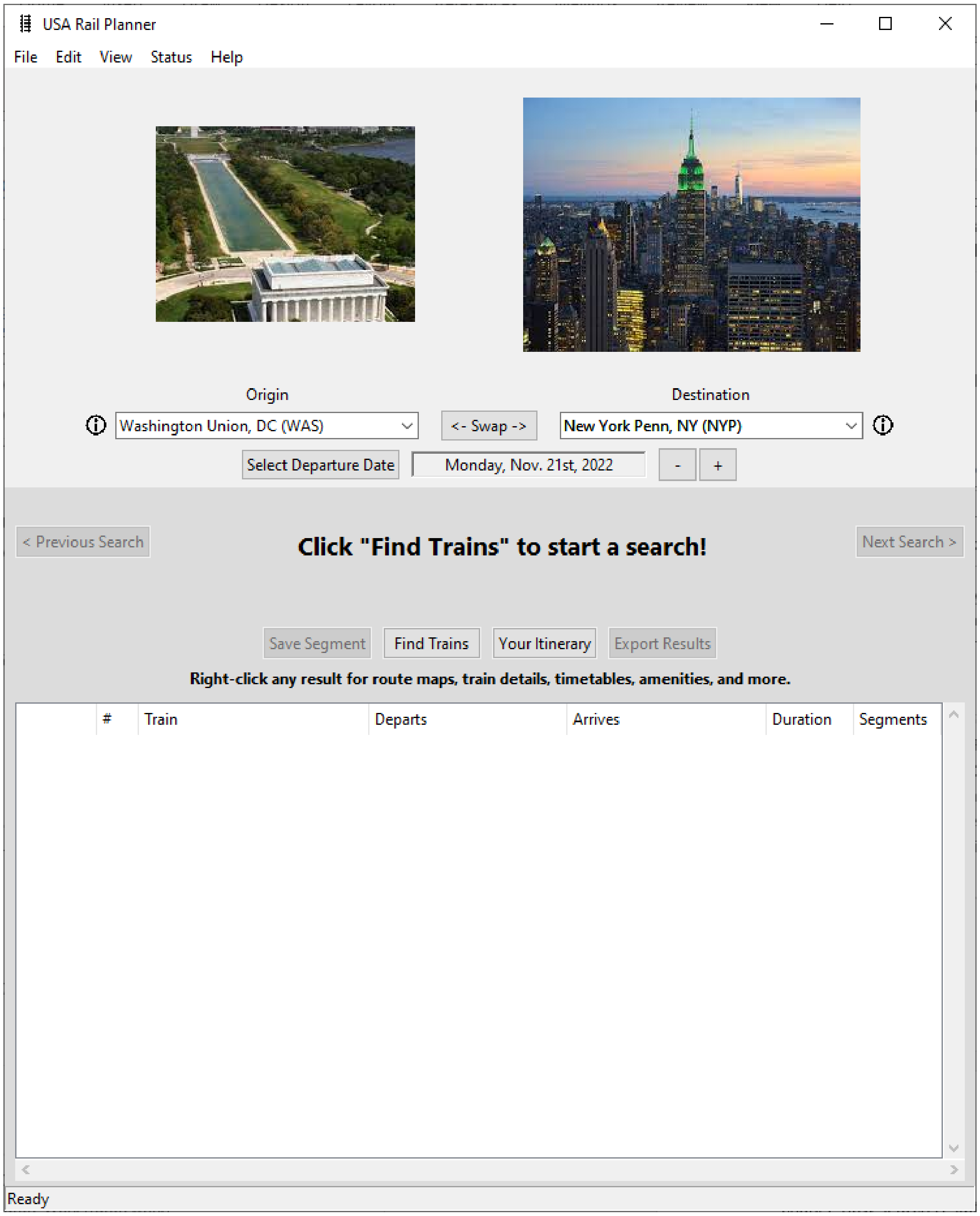
Chapter 5

Prototype in Action

In this chapter, an in-depth look at the Rail Planner interface is contained. In Section 5.1, the application is broken down window-by-window and its design elements are discussed. Multiple snapshots of the interface are provided. Then, two user interaction scenarios are presented in Section 5.2. These detail what a typical user might look for and do in the application. Finally, application limitations are discussed in Section 5.3, which are known problems with the application that may or may not be resolvable.

5.1 Interface Design

The Rail Planner is comprised of a main window and some subsidiary windows that each provide a unique type of information. The Main Window is where the user will spend most of their time, as it contains the tools to start searches and view results. The Itinerary window is the second-most important, containing all saved segments, and thus comprising a trip plan – the user's goal. Third, the Map window serves as a useful visualization for selected trains and/or stations, and information on its design is discussed. Finally, some additional elements, not necessarily substantial enough for their own subsection, are covered.



Click "Find Trains" to start a search!

Right-click any result for route maps, train details, timetables, amenities, and more.

Figure 5.1: Main Rail Planner Window on Startup

5.1.1 Main Window

Upon first launching the application, the user is presented with the Main Window (henceforth MW) and can begin their trip planning. This window is shown in Figure 5.1 as it would be at launch. Elements were placed in a specific manner from top to bottom to guide users towards the search button: start with station selection, then a departure date, and then they are ready to find trains. If no searches have been performed, the search results header displays a message about how to start searching.

Stations

The station selection area provides two dropdown menus with the complete list of Amtrak train stations, scraped from Wikipedia. This is displayed in Figure 5.2. They are formatted as "Station name, State (Code)" to provide as much identifying information as possible. As users change their selections, the images above the dropdown menus change to reflect the station's city and/or location. Simultaneously, the map window will move the appropriate marker to the station's position in the United States. Clicking an image will load a tourism website for that location, such as a tourism bureau (ex. `VisitNewYork.com`). The mouse pointer will also change to reflect the image is a clickable element. However, users may want some logistical information about the selected station. To view this, they can click the information "i" icon adjacent to its associated station. This will spawn a pop-up Detail View window, shown in Figure 5.3. It has pertinent station information, such as its address, Amtrak routes serving it, and local transit connections.

There are also features that make navigating station selection an easier process. Users may want to move their destination station to the origin slot and select a new destination, such as after saving a train segment. The swap button accomplishes this, swapping both the listed station and the images, without the additional overhead from re-searching for images. It is also apparent that the length of the dropdown menus and the number of stations included can be overwhelming to peruse. For this reason, an autocomplete feature was included to cut down on the amount of scrolling required

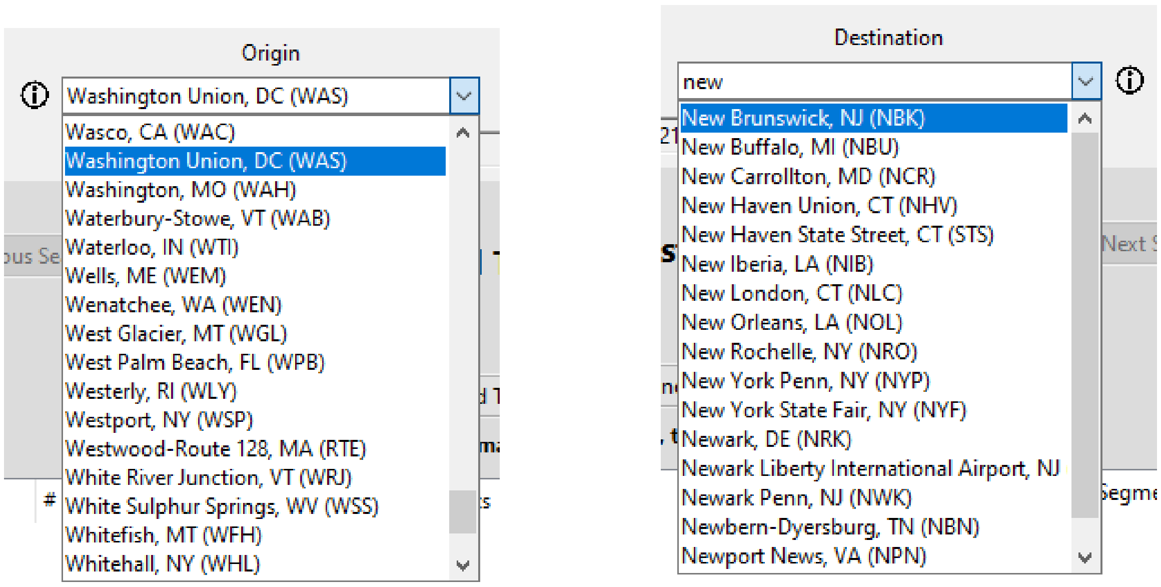
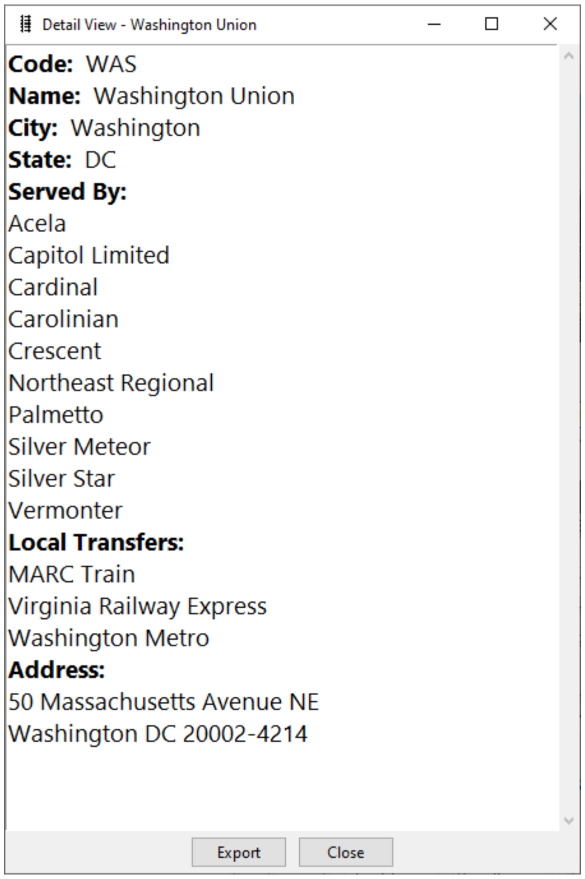


Figure 5.2: Stations List and Station List Search

for navigation. After typing in a keyword in the dropdown menu, the user can press Tab or Enter to load all the matching results, shown in Figure 5.2.

Departure Date

The next step in trip planning is selecting a departure date. As the application is meant for rail *journeys*, and not necessarily a round-trip between two points, a return date area was not included. However, that could be easily included in a later revision. Users have two options for modifying the date: choosing from a calendar, allowing the most freedom, or incrementing the displayed departure date forward or backward by one day. The first option is most useful when selecting a start date for the entire journey, while the second is better for subsequent selections where the date difference is not as drastic between searches. Users may click the "Select Departure Date" button or the currently displayed date to open a calendar selection area, seen in Figure 5.4. They can close this calendar by clicking the select button again or by choosing a new departure day on the calendar. Incrementing the date is simply done by using the plus and minus buttons to the right of the displayed date. One stipulation to the date selection functionality is that users cannot select a date in the past.

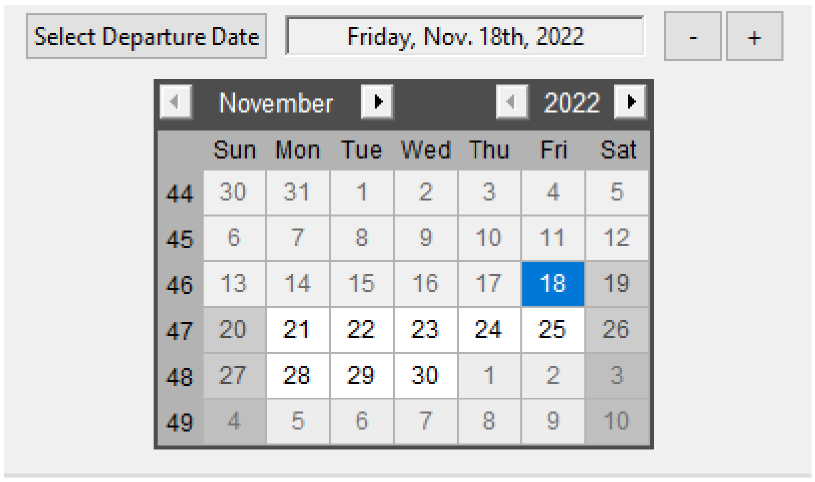


Code: WAS
Name: Washington Union
City: Washington
State: DC
Served By:
Acela
Capitol Limited
Cardinal
Carolinian
Crescent
Northeast Regional
Palmetto
Silver Meteor
Silver Star
Vermonter
Local Transfers:
MARC Train
Virginia Railway Express
Washington Metro
Address:
50 Massachusetts Avenue NE
Washington DC 20002-4214

Figure 5.3: Detail View for Washington Union Station, DC

Searching and Results

With the stations and departure date selected, users can begin their search. The searching area contains a host of buttons and information for the user and could be perceived as the most complex part of the application to use. Only two buttons are initially enabled. The "Find Trains" button begins the background search, and the user is presented with a progress bar. The "start a search" header is replaced with the title of the current search, which includes the origin, destination, and date. While seemingly redundant when juxtaposed with the station and date areas above it, the true use of the header comes when the user performs more searches. Once searching is complete, a label with the number of trains found is updated and the search results table is populated with all results, pictured in Figure 5.5. Result "type" is shown by an icon to the left of the entry, alongside its order in the table. Default



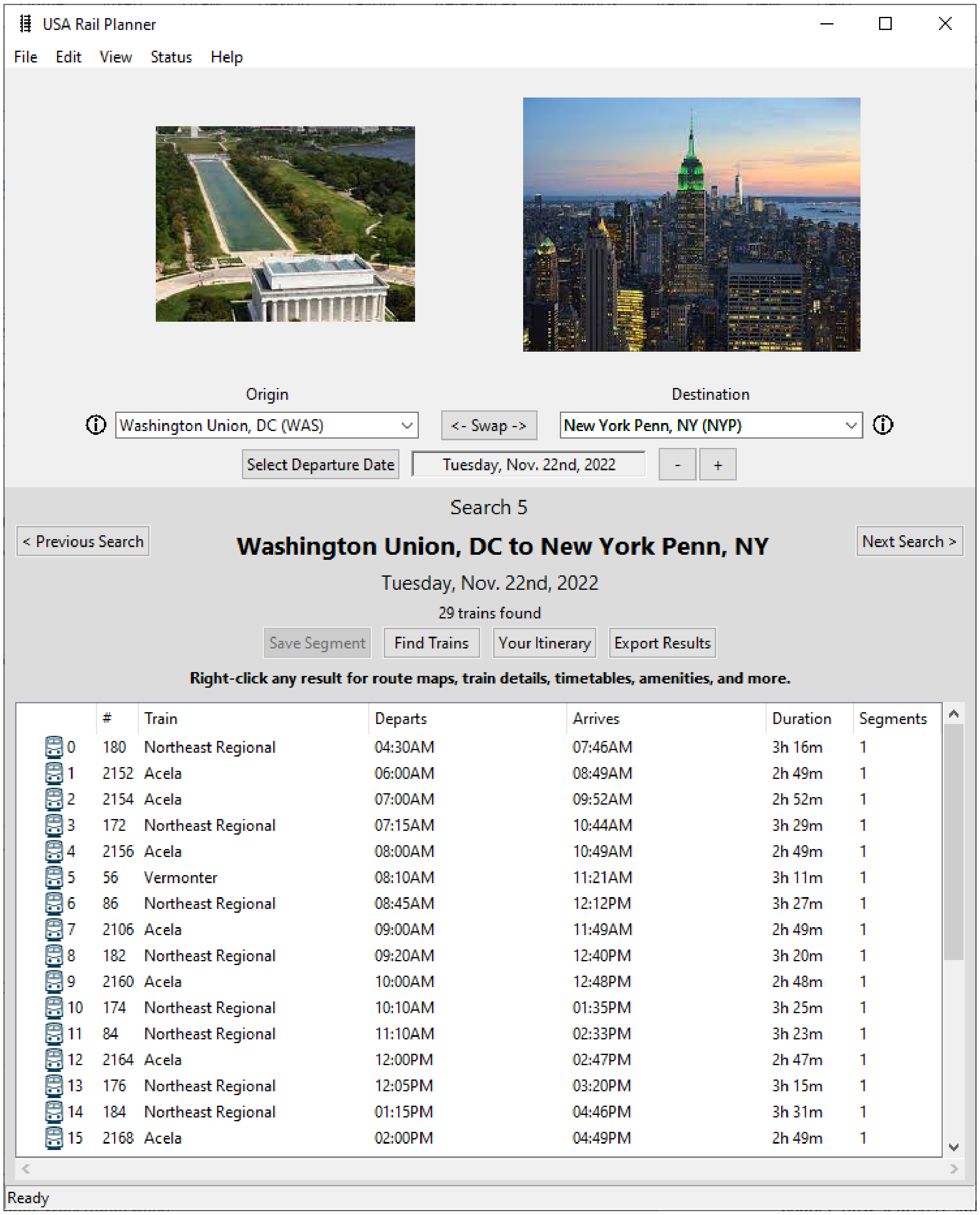
Friday, Nov. 18th, 2022

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
44	30	31	1	2	3	4	5
45	6	7	8	9	10	11	12
46	13	14	15	16	17	18	19
47	20	21	22	23	24	25	26
48	27	28	29	30	1	2	3
49	4	5	6	7	8	9	10

Figure 5.4: Calendar Popup in Date Selection Area

columns list the most important information, but the user can change the displayed columns through a menu setting and include prices or station codes. Take note of the departure and arrival times: trains arriving on the same day will not have a date listed. For multi-day or overnight trips, more detail is included, like the weekday and day of the month.

With results in the table, users now have more options available to them. They can export the results, if so desired, and, after clicking on any table entry/row, can then use the “Save Segment” button to record it into their itinerary. However, more functionality is present from within the Train Menu, a right-click activated context menu spawning from a selected result, seen to the left (results table) and right (Itinerary) in Figure 5.6. They can view online information (official Amtrak page for the train) or its timetable (from a “best guess” search for timetables). Both options will launch the user’s web browser to display the data. Offline functionality is viewing the train’s route map or more details, the latter of which will launch a Detail View window, similar to that of a station. In the spawned pop-up, users will find information about the trip as a whole, such as price or duration, displayed in 5.7. However, it also breaks down the information to a per-segment level, including additional information like available seats, on-board amenities, and individual arrival/departure times. This is especially useful for multi-segment trips which involve a connecting



Washington Union, DC (WAS)

New York Penn, NY (NYP)

Tuesday, Nov. 22nd, 2022

Search 5

Washington Union, DC to New York Penn, NY

Tuesday, Nov. 22nd, 2022

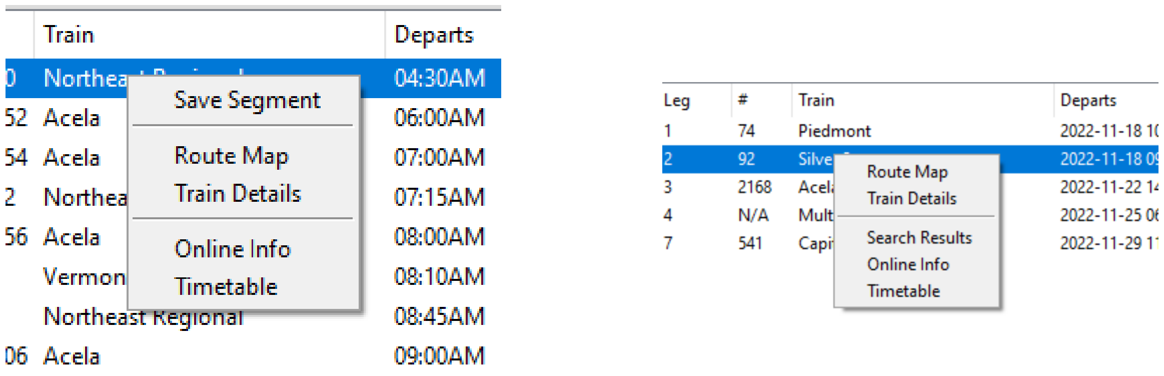
29 trains found

Right-click any result for route maps, train details, timetables, amenities, and more.

	#	Train	Departs	Arrives	Duration	Segments
0	180	Northeast Regional	04:30AM	07:46AM	3h 16m	1
1	2152	Acela	06:00AM	08:49AM	2h 49m	1
2	2154	Acela	07:00AM	09:52AM	2h 52m	1
3	172	Northeast Regional	07:15AM	10:44AM	3h 29m	1
4	2156	Acela	08:00AM	10:49AM	2h 49m	1
5	56	Vermonter	08:10AM	11:21AM	3h 11m	1
6	86	Northeast Regional	08:45AM	12:12PM	3h 27m	1
7	2106	Acela	09:00AM	11:49AM	2h 49m	1
8	182	Northeast Regional	09:20AM	12:40PM	3h 20m	1
9	2160	Acela	10:00AM	12:48PM	2h 48m	1
10	174	Northeast Regional	10:10AM	01:35PM	3h 25m	1
11	84	Northeast Regional	11:10AM	02:33PM	3h 23m	1
12	2164	Acela	12:00PM	02:47PM	2h 47m	1
13	176	Northeast Regional	12:05PM	03:20PM	3h 15m	1
14	184	Northeast Regional	01:15PM	04:46PM	3h 31m	1
15	2168	Acela	02:00PM	04:49PM	2h 49m	1

Figure 5.5: Main Window with Search Results

train or bus, as users can determine the transfer time and modes of travel.



	Train	Departs
0	Northea	04:30AM
52	Acela	06:00AM
54	Acela	07:00AM
2	Northea	07:15AM
56	Acela	08:00AM
	Vermon	08:10AM
	Northeast Regional	08:45AM
06	Acela	09:00AM

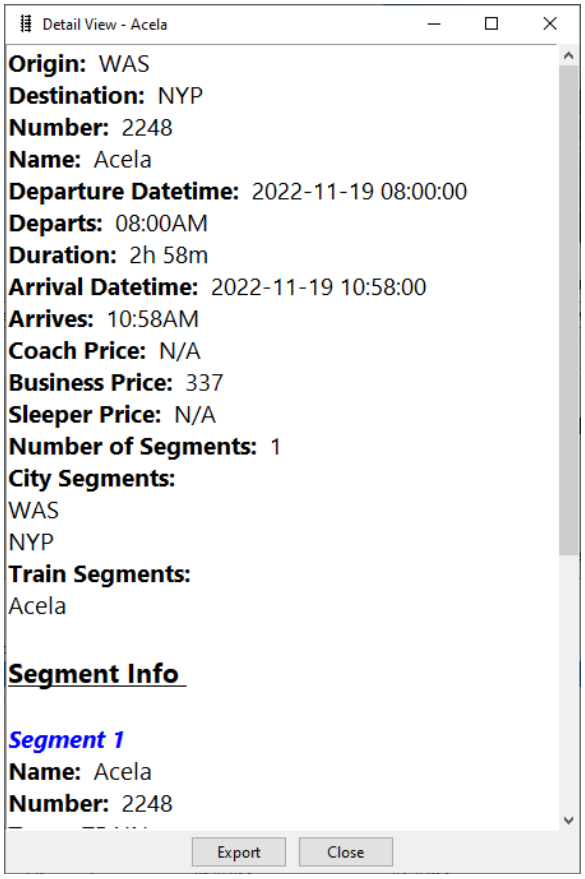
Leg	#	Train	Departs
1	74	Piedmont	2022-11-18 1(
2	92	Silve	2022-11-18 0
3	2168	Acel	2022-11-22 14
4	N/A	Mult	2022-11-25 0(
7	541	Capi	2022-11-29 1

Figure 5.6: Train Right-click Context Menu for Results Table and Itinerary

As the user performs more searches and gradually grows their itinerary, they may want to revisit a previous search. Perhaps a future connection can only be made with an earlier train, so an earlier segment needs to be modified. With the “Previous Search” and “Next Search” buttons in the search area, users can traverse their past searches without the need to perform another one. This is a major advantage over the Amtrak website and saves a considerable amount of time. The search number at the top of the area reflects the search number, and the heading changes to the proper stations and date. The respective buttons will become disabled when reaching either endpoint: first search or most recent search.

5.1.2 Itinerary

The Itinerary is a valuable resource to the user as it contains all their saved segments and the ability to export them, seen in Figure 5.8. The table listing the segments is identical to the search results table except for an extra column at the beginning, denoting what leg of the trip the entry represents. The departure/arrival columns also include the full date and time as trains in the itinerary likely have different days for their departures. Users still have the right-click context menu functionality for any train in the itinerary. The most important feature in this window is the ability to export the itinerary to a spreadsheet so it can be used to help book the trains later. Users can also reorder or delete itinerary elements, where the former would be useful



Origin: WAS
Destination: NYP
Number: 2248
Name: Acela
Departure Datetime: 2022-11-19 08:00:00
Departs: 08:00AM
Duration: 2h 58m
Arrival Datetime: 2022-11-19 10:58:00
Arrives: 10:58AM
Coach Price: N/A
Business Price: 337
Sleeper Price: N/A
Number of Segments: 1
City Segments:
WAS
NYP
Train Segments:
Acela

Segment Info

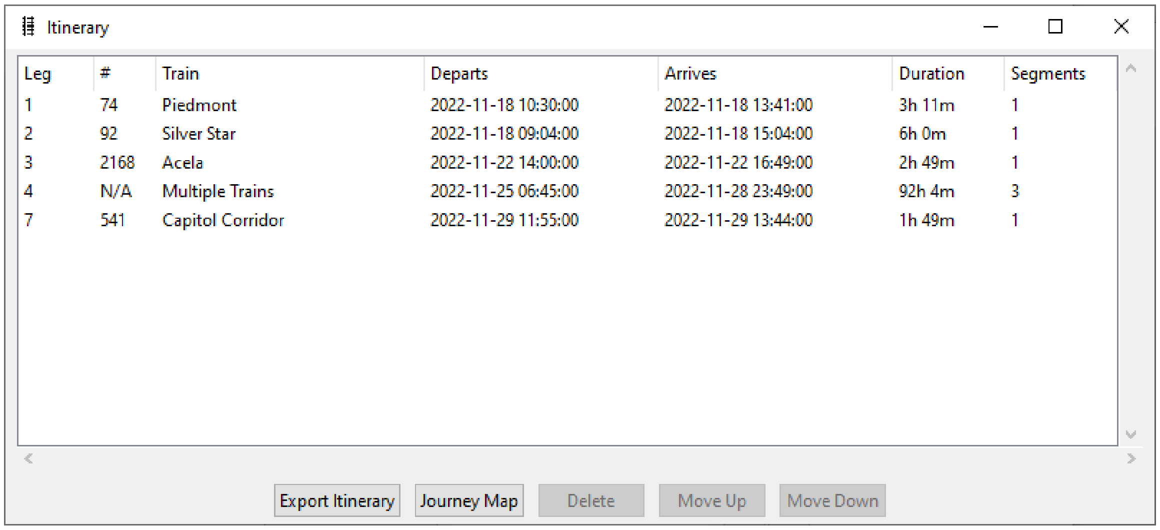
Segment 1
Name: Acela
Number: 2248

Figure 5.7: Detail View for Acela #2152

if the trip was planned out of chronological order. Finally, users can select a saved train and view the search results (from right-click menu) from where it was selected in case they need to view alternate options for the same parameters (stations and date).

5.1.3 Journey View (Map)

The map window is launched at application startup but can be closed or reopened (by using any route map button/menu option). It has three modes of map display. The first and primary mode is displayed during station selection, showcased in Figure 5.9. As the user changes their origin and destination choices, the map will redraw the appropriate markers and the line between them. This map mode will take priority over anything currently on the map, erasing it. Figure 5.10 displays the second mode,



Leg	#	Train	Departs	Arrives	Duration	Segments
1	74	Piedmont	2022-11-18 10:30:00	2022-11-18 13:41:00	3h 11m	1
2	92	Silver Star	2022-11-18 09:04:00	2022-11-18 15:04:00	6h 0m	1
3	2168	Acela	2022-11-22 14:00:00	2022-11-22 16:49:00	2h 49m	1
4	N/A	Multiple Trains	2022-11-25 06:45:00	2022-11-28 23:49:00	92h 4m	3
7	541	Capitol Corridor	2022-11-29 11:55:00	2022-11-29 13:44:00	1h 49m	1

Figure 5.8: Itinerary Window with Five Saved Results

which is associated with individual train trips. As users inspect search results or their itinerary, they can right-click an entry to access the route map functionality. This redraws the map with updated styling specific to the Rail Planner. Inclusion of intermediate stops on the route was another highly requested feature, and a map is a better option than a simple list. The third mode is similar to the second mode but features all segments of the user's itinerary and is entitled the Journey View, seen in Figure 5.11. Routes are still displayed but intermediate stops are excluded as it would make the map too cluttered. Additionally, all important stops are prefixed with a number corresponding to the leg of the journey, so the users can follow their trip on the map. Stations with multiple trips through them will have multiple numbers prefixed, and stations acting as an origin and destination will have two numbers. For all modes, clicking on emphasized stations (large font) will open the station's Detail View window. The map view, no matter the mode, provides a useful visualization for every aspect of the user's trip planning.

5.1.4 Additional Elements

There are some additional features that did not warrant their own section but still provide value and enhance the user experience.

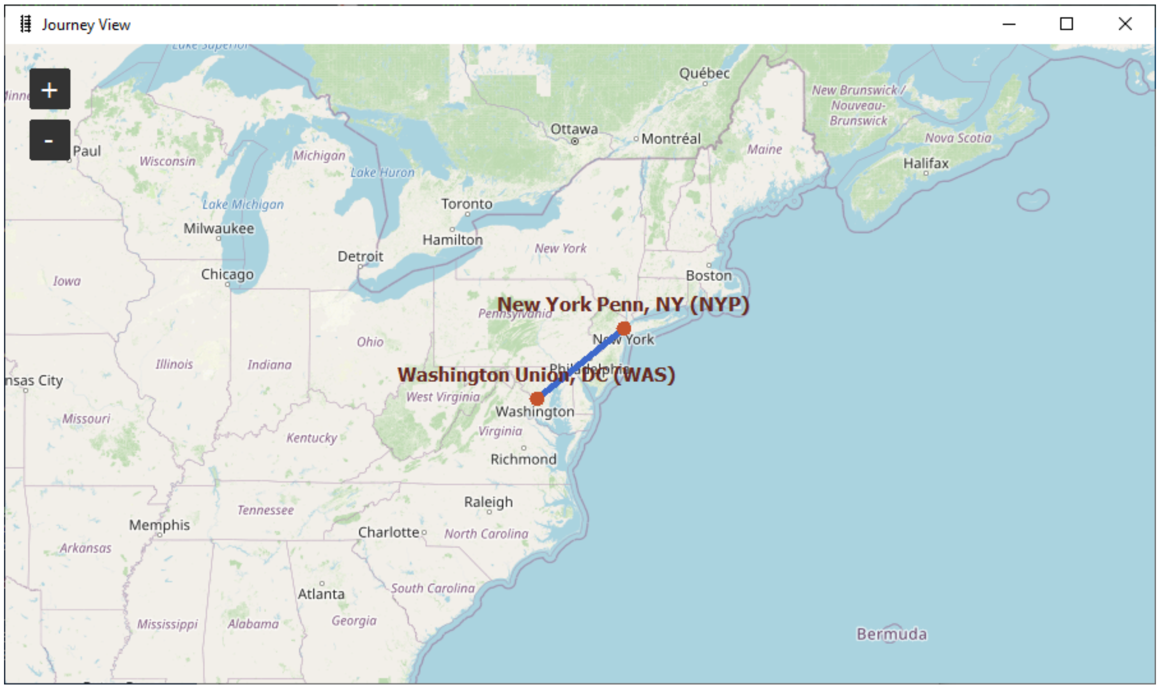


Figure 5.9: Map Window during Station Selection

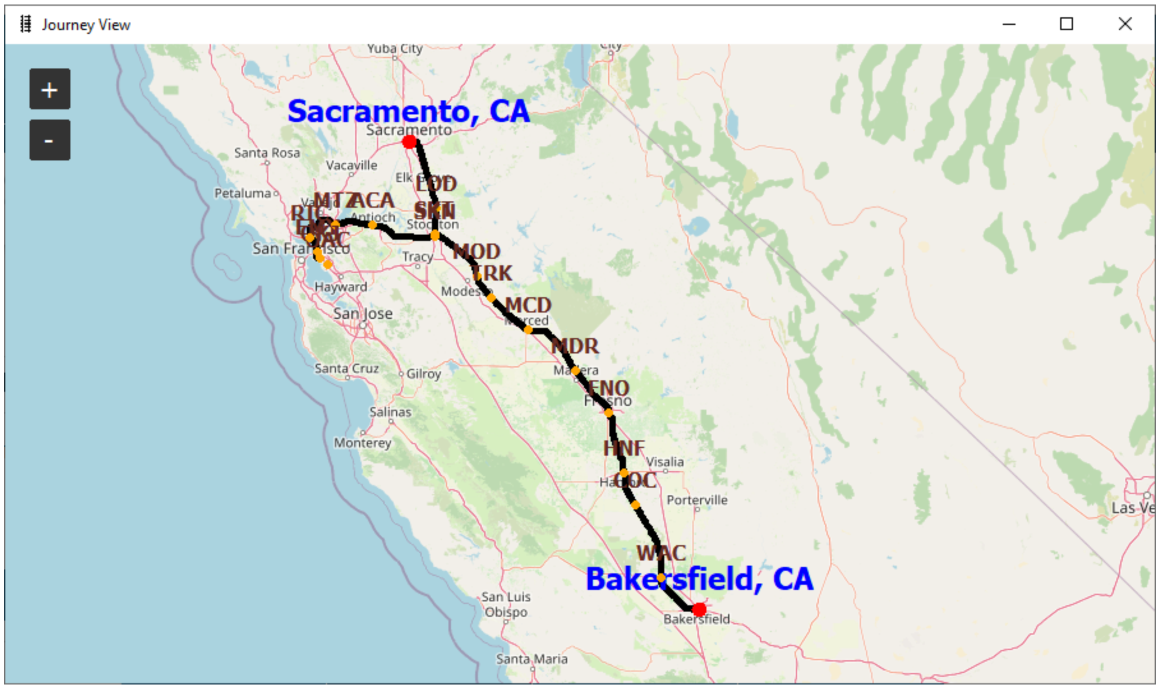


Figure 5.10: Map Window with Single Train Route

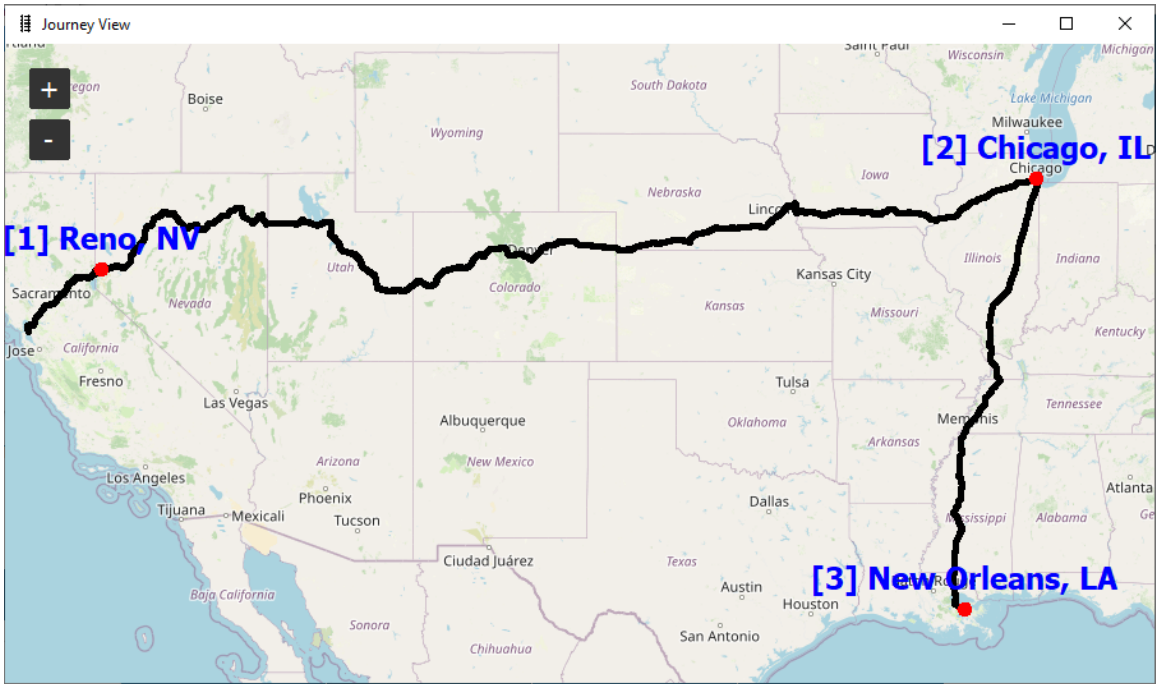


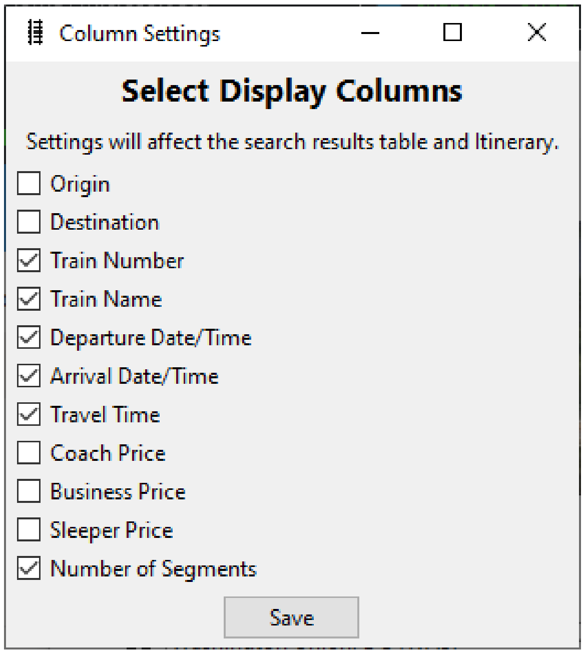
Figure 5.11: Map Window with Journey (Itinerary) View

Detail View

The Detail View window, discussed earlier, provides additional information about a station or train. Data already stored in the application is taken from its dictionary object and displayed in a friendly form for easier reading. Keys are in bold, while segment information has some additional styling to differentiate the section from the main information. The user can also export the information into a `.txt` or `.json` file, or a formatted HTML document, which is nearly identical to the Detail View.

Menu Options

At the top of the Main Window is a menu bar with five menus embedded. The File Menu provides file manipulation functions (New, Open, Save) for the `.RailPlan` files. The itinerary can also be exported from this menu. The Edit menu provides one option: modifying the display columns for the results/itinerary tables. From this popup menu, shown in Figure 5.12, users can choose which columns are shown. By default, prices, origin, and destination columns are hidden. The View menu can open the Itinerary and Map windows, if closed. It also provides links to the Amtrak system map (all routes in the network) and on-time performance information via Amtrak or the Bureau of Transportation Statistics. The Status menu provides links to train statuses across the country. The Nationwide option loads an interactable map with all trains currently in service and their details, while regional links load the respective region. The Help menu has a link to the application's GitHub page and the About pop-up displays the current version of the Rail Planner.


Column Settings

Select Display Columns

Settings will affect the search results table and Itinerary.

- ☐ Origin
- ☐ Destination
- ☑ Train Number
- ☑ Train Name
- ☑ Departure Date/Time
- ☑ Arrival Date/Time
- ☑ Travel Time
- ☐ Coach Price
- ☐ Business Price
- ☐ Sleeper Price
- ☑ Number of Segments

Figure 5.12: Display Column Editor

5.2 Common Usage Scenarios

The Rail Planner is not a "generalist" application; its goal is to find trains and create itineraries. As such, usage scenarios are not all that varied. However, its included features allow for some variations on the planning process.

5.2.1 Cross-Country/Rail Pass Trips

For the average user, this is where their usage will be. Perhaps they purchased a Rail Pass, which allows for ten travel segments, and they need to use them all. They may have a start point and end point and everything in-between is undecided. The Amtrak system map, accessible from the application, can provide them with ideas of where to go. From there, they can begin searching for trains. They may inspect results for certain trains and view their route maps, and perhaps a station along the way piques their interest, and they change that segment to stop there instead. The user may not want to ride any buses along the way, so they view the train details for multi-segment results (trips with transfer(s)) to make sure each segment is a train or check the icons next to table entries for the trip type. Similarly, they may look for trains with WiFi on-board or with ADA accessible boarding, information which is also provided in the train Detail View. Perhaps they want to check local transit connections at the station (ex. SEPTA at Philadelphia's 30th Street Station) so they can get to a tourist attraction they found by checking the site linked from the location's picture. Finally, once their itinerary is complete, they want to see their entire journey on a map and export their itinerary so they can book the tickets. Perhaps that same exported itinerary is shared with family members or those they're meeting along the way, so they know which trains to watch or track. In this way, the application demonstrates that it is more than an "Amtrak searcher" and provides useful information to those who plan trips in detail and at their own pace.

5.2.2 Route Variations

Advanced travelers (or railfans) may have multiple train routes they want to ride but have a short timeframe or a limited number of travel legs (Rail Pass). For this reason, the ability to save and load progress is invaluable. Users could have two or three potential sets of destinations to visit but must choose based on feasibility given their limitations. Separate itineraries can be searched for and planned, all saved in separate files for later use or comparison. Without this feature, all searches and itineraries would be ethereal unless exported to an itinerary spreadsheet file. Previous searches would need to be redone and the itinerary recreated. With a user's preference for specific routes in mind, the train detail view also becomes an important planning tool. Multi-segment trips list which trains make up the journey, and their respective departure/arrival times and origin and destination stations.

5.3 Limitations

While the application was designed to handle errors thoroughly and avoid interface glitches, it is important to document limitations in its design so future work can avoid these pitfalls. Problems primarily exist with the user interface and with searching the Amtrak website.

5.3.1 User Interface

Tkinter widget managers (pack, place, grid) can sometimes take finesse when programming to make elements appear correctly and behave properly during window geometry changes. However, some widgets do cause issues when application windows, or the elements themselves, are resized. One example is the “button area” in the Detail View window, which is squished out of the window when the height is decreased. Another case is the itinerary table when more columns are added. Instead of changing column widths according to the window size, the length of the table increases. A horizontal scrollbar is present, which mitigates the issue, but the vertical scrollbar is pushed out of the window. With some experimentation, these issues could be resolved, but they were a lower priority than new features in the application.

With high-DPI screens and a high scaling percentage, some portions of the application may be cut off at the top or bottom of the screen. This is especially prevalent on smaller screens (such as a 13” or 14” laptop screen). As the application has a minimum defined size for all the elements to fit properly, this is a current limitation for laptop users. Moving the results table/area to the side or a separate window is a potential workaround.

5.3.2 Searching and Web Drivers

The automated search of Amtrak's website is, by far, the weakest link in the application. Aside from locating elements by XPath, and the issues that brings, Amtrak is not privy to web scraping. In the past, the application was blocked from searching using the web driver, and the Undetected Chromedriver [41] was used in place of a regular web driver to mitigate this. However, it seems that the blocking was based on IP address and browser type, as repeating the search from a different location (different IP) did work. As such, using a virtual private network (VPN) service with the application prevents this browser block, and initial testing of this method supported the theory.

It also seemed that using the browser in "headless" mode (no window is created) increases the likelihood that it is flagged, something that is noted in the Undetected Chromedriver documentation as well. To mitigate that, the browser was started as a minimized window, but that produced mixed results. For example, sometimes the page would not load properly or at all if the window was minimized. Generally, the first search would be successful, but future searches would timeout, thus providing an unpleasant user experience. Hiding the browser by placing the Chrome window on a separate desktop (multiple desktops feature in Windows/MacOS) also provides mixed results. At this time, leaving the browser window open and maximized is the safest way to use the application. Finally, the data retrieved from Amtrak is expected to follow a similar form so the application can parse the information it needs properly. While it is not expected that Amtrak would change how search result data is stored (such as key-value information/names), this too would break the application's parsing feature.

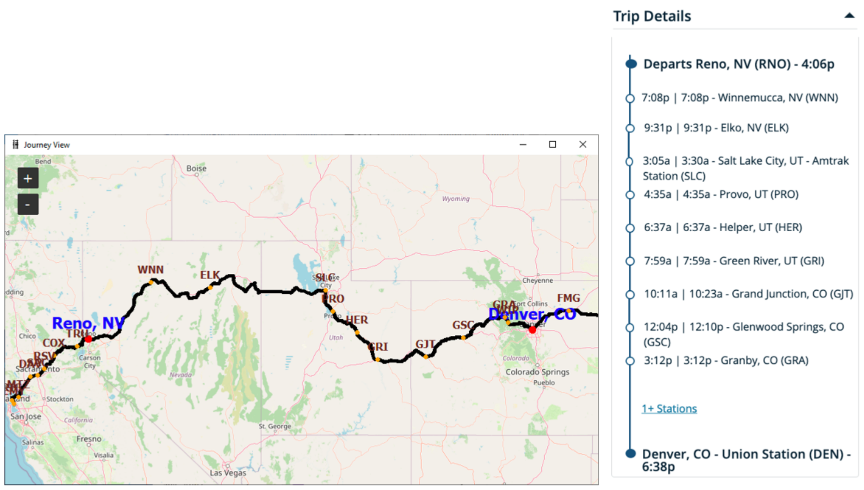
Chapter 6

Validation

In this chapter, the user study to evaluate the performance of the USA Rail Planner is presented. In Section 6.1, details about the participants are discussed, such as demographic information and train familiarity. The pre-study questionnaire is also included. Section 6.2 describes the testing environment. Specifically, hardware and software utilized and the physical setup of the study. Section 6.3 covers the experimental design and testing procedure. The study results are presented in Section 6.4 with significant conclusions. Finally, a general discussion of the study, with user experiences and habits, is in Section 6.5.

6.1 Subjects

For the user study, 12 participants were recruited. Each of them were given a pre-study questionnaire, listed in Table 6.1. Question 4 was a somewhat vague question, as “train” can be interpreted in a variety of ways. One participant asked if a shuttle, such as a tram inside an airport, would count as a train, to which they were informed it did not. Specificity was originally included in the question, naming subway and light rail, but there was a concern that participants would not know the “formal” definition of light rail and would include or exclude previous train trips that were outside the scope of the question. For question 6, any answers that mentioned a travel-specific service, such as the Southwest Airlines website, were included in the “Domain specific applications” response. Question 9 had pictures of the two methods, which is shown in Figure 6.1. These pictures were only referred to as “Option A” and “Option B” in the questionnaire, but USA Rail Planner’s map window is the left picture, and Amtrak’s “Trip Details” dropdown for a train on the search page is on the right.



Trip Details

Departs Reno, NV (RNO) - 4:06p

7:08p | 7:08p - Winnemucca, NV (WNN)

9:31p | 9:31p - Elko, NV (ELK)

3:05a | 3:30a - Salt Lake City, UT - Amtrak Station (SLC)

4:35a | 4:35a - Provo, UT (PRO)

6:37a | 6:37a - Helper, UT (HER)

7:59a | 7:59a - Green River, UT (GRI)

10:11a | 10:23a - Grand Junction, CO (GJT)

12:04p | 12:10p - Glenwood Springs, CO (GSC)

3:12p | 3:12p - Granby, CO (GRA)

1+ Stations

Denver, CO - Union Station (DEN) - 6:38p

Figure 6.1: Pre-Study Questionnaire Q9 Comparison Images

There were 9 males and 3 females with a mean age of 29.75 years and a median age of 23 years. Two-thirds of them were between the ages of 18 and 24, however the rest fell into the remaining age brackets, seen in Figure 6.2. Out of all participants, eight had traveled by train in the United States in the past. All eight had done so for personal or leisure travel purposes, but two of them also commuted by train. Of those that had traveled by train, 75% reported that it was done rarely (1-3 times per year) while the rest traveled often (1-4 times per month), shown in Figure 6.2. However, despite the relatively large number of participants who had ridden a train, the mean familiarity with Amtrak and its services was 3.667. Group chats (8) and domain-specific websites (10) were the most preferred method to plan trips for participants. Similarly, most determined where to go by having a few places in mind to visit beforehand. When presented with the two station stop figures (Figure 6.1), the study group was about neutral in their preference for one or the other. The Rail Planner’s map etched out a slight lead over Amtrak’s trip details, with the mean ranking for

Table 6.1: Pre-Study Questionnaire

Question		Response
1	What is your age?	Number entry; decline
2	What is your gender?	Male; Female; Other; Decline
3	How familiar are you with Amtrak? Routes, where it travels, commuter service, etc.	1-7 scale
4	Have you ridden a train (not subway) in the United States in the past?	Yes, personal; Yes, commuting; No
5	If you answered YES to question 4, how often do you travel by train in a typical year?	Frequently (5+ times/month); Often (1-4 times/month); Occasionally (4-6 times/year); Rarely (1-3 times/year)
6	Describe how you plan a trip (road trip, ariplane travel, rail, etc). What methods do you use to accomplish this? Mark all that apply.	Organizational websites (Notion, MS Planner, Evernote, etc); General purpose software (Notepad/Notes, MS Excel, Google Docs, etc); Group chats/messaging (iMessage, WhatsApp, Snapchat, Facebook Messenger, etc); Domain specific apps (Kayak, Rome2Rio, Google Flights, Roadtrippers, etc); Other
7	How would you likely determine where to go on multi-city trips?	I generally have a good idea of some destinations I want to visit; I pick a route/highway or two and visit places along the way; I only know where I'd start and end, the rest is undecided
8	How confident are you that you can plan a multi-city train trip?	1-7 scale
9	Please rate your preference for these two methods to view a train's stops between its origin and destination.	1-7 scale

the question being 3.833, an advantage of 0.167. This may indicate that users found both types of information useful, such as how Amtrak's details include station arrival times and are in text form, but the map provides geospatial context.

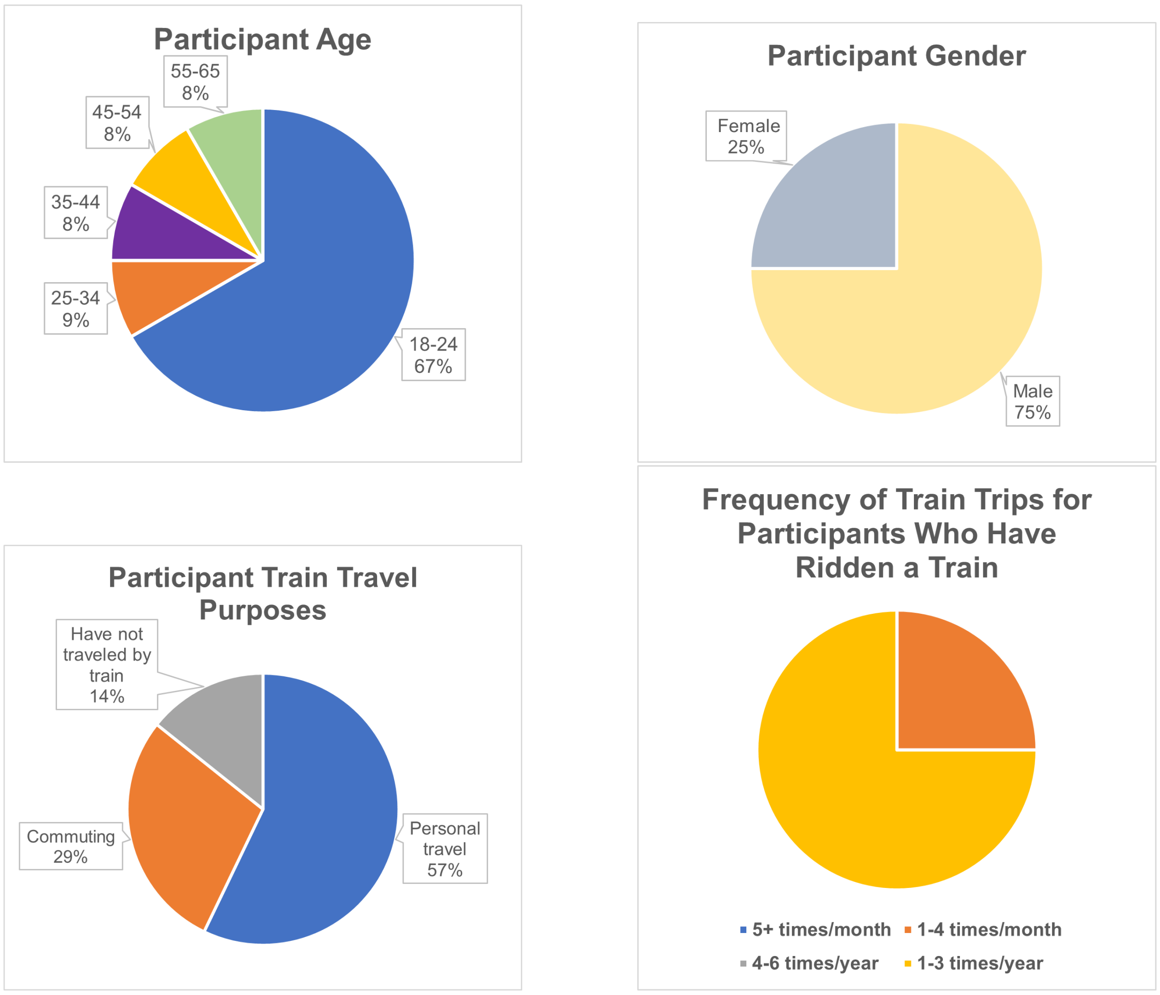


Figure 6.2: Participant Demographic Information

6.2 Apparatus

Each participant received as nearly identical a testing environment as was possible. The testing environment, shown in Figure 6.3, consisted of a few key items. A Macbook Pro laptop (13-inch Retina screen, Early 2015: Intel Core i5-5287U CPU, Intel Iris Graphics 6100, 16 GB RAM) was used for all computer interaction. A Logitech M705 wireless mouse was available for use, if the participant preferred a mouse over a laptop trackpad. The laptop ran a Windows 10 21H2 virtual machine using Oracle VirtualBox 6.1.38. For this virtual machine, 2 processors and 4 GB of memory were allocated. Google Chrome browser and Microsoft Excel LTSC Professional Plus 2021 were used for the Amtrak + Excel task. Windscribe VPN [43] and Blackout [44] was installed and in use for the USA Rail Planner task only. Mousotron [45] software was used during each task to gather interaction metrics.

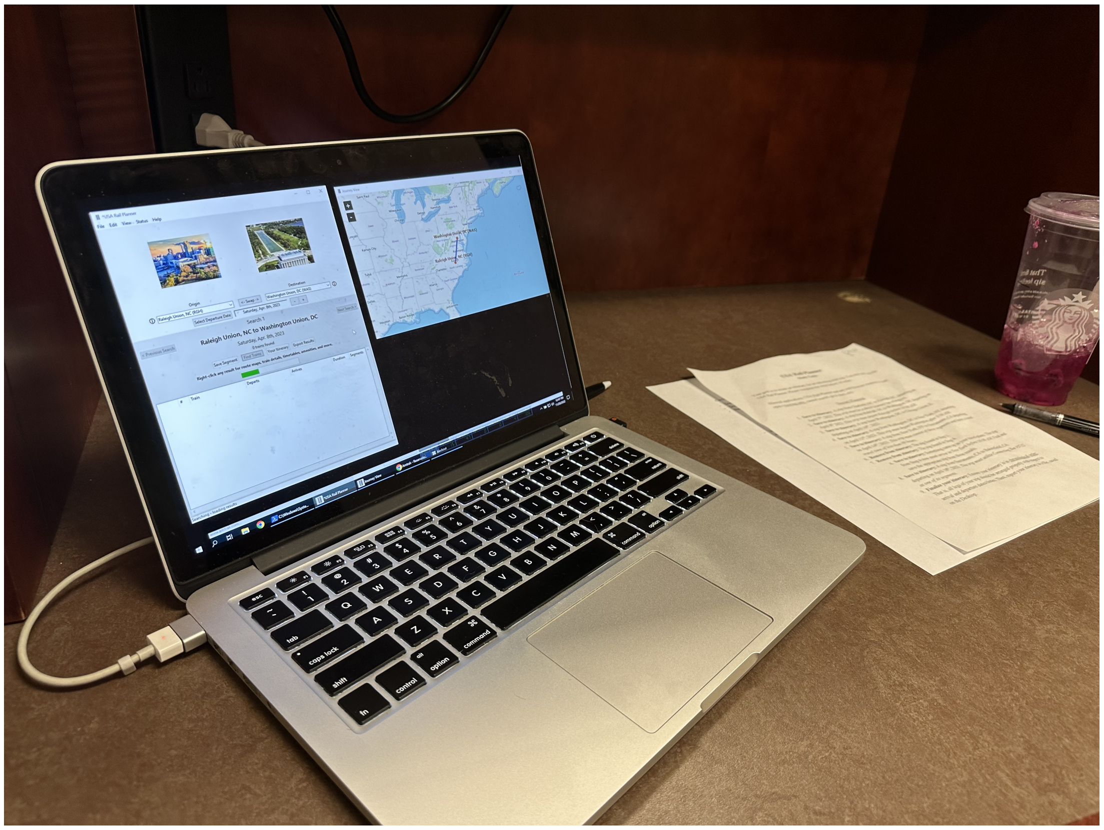


Figure 6.3: Testing Environment Setup

Studies were conducted in quiet, private spaces with ample desk space, such as a university conference or study room. While it was not possible to use the same room for each study, the rooms and layouts were similar enough to not introduce any unnecessary bias to the study. For each task, participants were given two sheets of paper. One had the steps to complete for the current task and the other was the associated trivia questions to answer. A choice of two pens, one a traditional black ballpoint pen and the other a blue gel pen, were provided to use.

6.3 Experimental Design and Procedure

A within-subjects design was chosen for this study to gather quantitative performance information for two itinerary planning methods. Additionally, this also allows for comparison questions in the post-study questionnaire for later analysis. This study had one independent variable: itinerary planning method, which is either the USA Rail Planner or the Amtrak website with an Excel spreadsheet for data recording (Amtrak + Excel). Each participant had steps to complete for each method (task), which required searching for trains and saving the trip data. The task would end if they took longer than 100 minutes, however, no participants reached the time cap. The study duration ranged from 80 to 120 minutes for most participants. Dependent variables in the study were the time it took to complete the task (planning steps and trivia questions) and accuracy, the latter being divided into task and trivia accuracy. Counterbalancing was implemented in the study to reduce any learning effects. Participants were assigned a group when they joined the study. The first group used the Amtrak + Excel method first, then the USA Rail Planner. The second group performed the inverse. No training or demos were given to the participants. The first time they see the Rail Planner is when the task begins.

The experiment began with the participant being seated at the desk in front of the testing laptop and the researcher on either side. A consent form was provided, detailing the study goals and risks. After this form was signed, participants were given the pre-study questionnaire to fill out. This questionnaire's content is listed in Table 6.1. Then, a brief overview of the testing portion was conveyed to the participants. They were informed that they would need to create an itinerary for an upcoming hypothetical train trip. The trivia document was described, with particular emphasis on the fact that it is meant to be completed simultaneously with the task and that it would be a good idea to consult this document after completing each task step. They were also told not to spend too much time on any one question and that they could be skipped if they could not find the answer. Finally, the question policy was covered,

Table 6.2: User Study Task Steps

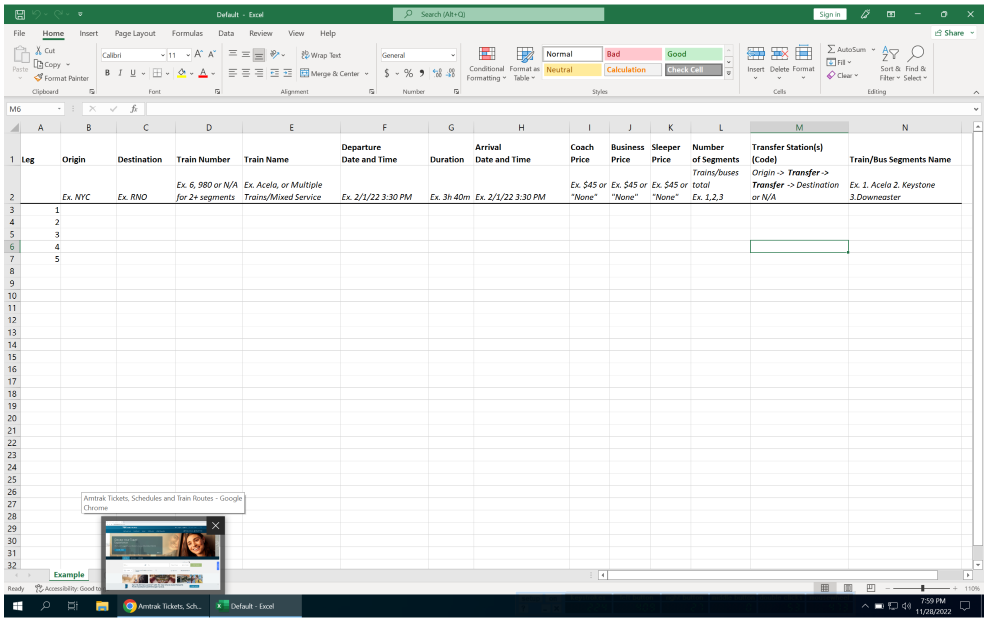
Step	**Action**	**Task**	**Rail Planner**	**Amtrak/Excel**
1	Save	A trip from [**A**] to [**B**], departing on April 7th, 2023. This trip must be a direct journey (no transfers).	Charlotte, NC; Raleigh, NC	Santa Barbara, CA; Los Angeles, CA
2	Save	A trip from [**C**] to [**D**] departing on April 8th, 2023. [**E**]	Raleigh, NC; Washington, DC; This trip must depart Raleigh before 10:00 AM	Los Angeles, CA; Fresno, CA
3	Save	A trip from [**F**] to [**G**] departing on April [**H**], 2023. [**I**]	Washington, DC; Chicago Union, IL; 8th; no conditions	Los Angeles, CA; St. Louis, MO; 9th; arrive on a Wednesday
4	Save	A trip from [**J**] to [**K**] departing on April [**L**], 2023. This trip must depart [**M**] after [**N**].	Washington, DC; New York, NY; 14th; Washington; 8:00 AM	St. Louis, MO; Chicago, IL; 12th; St. Louis; 12:00 PM
5	Save	A trip from [**O**] to [**P**] departing on April 14th, 2023. This trip must consist of [**Q**] segments and have a total travel time of less than [**R**] hours.	New York, NY; Sacramento, CA; 3; 88	Chicago, IL; San Jose, CA; 4; 80
6	Remove	The trip found in Step [**S**].	3	2
7	Remove	The trip found in Step 4.		
7a	Save	Something came up in your travel plans. The trip from Step 4 now must arrive in [**T**] before [**U**]. Find and save the appropriate trip from those results.	New York; 10:00 AM	Chicago; 2:00 PM
8	Save	A trip from [**V**] to [**W**] departing on April 18th, 2023. This trip must utilize [**X**] as one of its segments.	Sacramento, CA; Bakersfield, CA; Connecting Bus 3712	San Jose, CA; Los Angeles, CA; Pacific Surfliner 784
9	Finalize	Ensure your itinerary is in chronological order. That is, all legs of your trip should be arranged properly with respect to arrival and departure dates/times. Then, [**Y**].	export your itinerary to a file, saved on the Desktop.	save your itinerary in Excel with the floppy disk icon at the top left.

Table 6.3: User Study Trivia Questions

Q	Type	Trivia	Rail Planner	Amtrak/Excel
T1	Yes/No	Does the trip from Step [**A**] make a stop in [**B**]?	2; Fredericksburg, VA (FBC)	8; Goleta, CA (GTA)
T2	Yes/No	Does the trip from Step 5 make a stop in [**C**]?	Fort Madison, IA (FMD)	Ontario, CA (ONA)
T3	Write-in	Name the [**D**] train segments from the Step 5 trip.	three	four
T3a	Write-in	What date and time does the [**E**] segment depart?	second	third
T4	Write-in	Indicate the train number for the trip found in Step 4 or Step 7a. Name the amenities that are on board this train.	New York, NY; Sacramento, CA; 3; 88	Chicago, IL; San Jose, CA; 4; 80
T5	Yes/No	Does the [**F**] station have train service from the "[**G**]" train?	New York, NY; Downeaster	San Jose, CA; California Zephyr
T5a	Write-in	List the local transportation options from this station.		
T6	Write-in	What is the address of the [**A**] station?	Sacramento, CA	St. Louis, MO

which stated that questions could not be answered but to ask anyways so their thought process could be determined. In some circumstances, questions were answered, such as clarification about a task step or where to save the itinerary file. Additionally, participants were informed that assistance would be provided if they were seen to be struggling with a task step. While not explicitly told to the participants, eight minutes would need to pass during a step before assistance would be provided. No such assistance was available for struggles with trivia questions.

The task and trivia documents are then provided to the participant. These are respectively shown in Table 6.2 and Table 6.3. As the tasks differed between methods to prevent any prior information from being used in the second task, some key details were changed. However, the stations and dates were chosen such that the quantity of results and types of trains are similar between tasks. Bolded markers in the tables are placeholders for substitutions from the respective columns for the methods. The reasoning behind including the task and trivia simultaneously is that it was meant to simulate a typical planning process. While the primary part of planning a rail trip is searching for trains, secondary information is useful to have, and would likely be researched during planning. Such examples of this include local transportation options from a station or amenities on-board (like WiFi). In this way, a more holistic approach to evaluating the applications, representative of real usage scenarios, is in place. A brief description of the upcoming task is told to the participant, including the applications they are allowed to use. During the tasks, a timer is running, and the researcher took notes on user interaction methods and correctness of the itinerary and trivia.



Leg	Origin	Destination	Train Number	Train Name	Departure Date and Time	Duration	Arrival Date and Time	Coach Price	Business Price	Sleeper Price	Number of Segments	Transfer Station(s) (Code)	Train/Bus Segments Name
	Ex. NYC	*Ex. RNO*	*Ex. 6, 980 or N/A for 2+ segments*	*Ex. Acela, or Multiple Trains/Mixed Service*	*Ex. 2/1/22 3:30 PM*	*Ex. 3h 40m*	*Ex. 2/1/22 3:30 PM*	*Ex. $45 or "None"*	*Ex. $45 or "None"*	*Ex. $45 or "None"*	*Trains/buses total Ex. 1,2,3*	*Origin -> **Transfer** -> **Transfer** -> Destination or N/A*	*Ex. 1. Acela 2. Keystone 3.Downeaster*
1													
2													
3													
4													
5													

Figure 6.4: Amtrak + Excel Task Setup in VM

Group 1 participants started with the Amtrak + Excel method, shown in Figure 6.4. The Excel itinerary file is displayed, and a window preview of the Google Chrome browser with the Amtrak website already loaded is seen in the taskbar. Excel is also an allowed application. Note that the Excel spreadsheet has labeled columns with a short example just beneath them, so participants knew what information to gather. The columns are identical to what the Rail Planner would report in its exported itinerary, save for an extra Rail Planner column which saves granular details for each train in a result. In this way, the Amtrak + Excel method was at a slight advantage, especially considering the breadth of information in the Rail Planner's train Detail View. For this task, participants can use the web browser with no restrictions, so they are not limited to Amtrak's website.

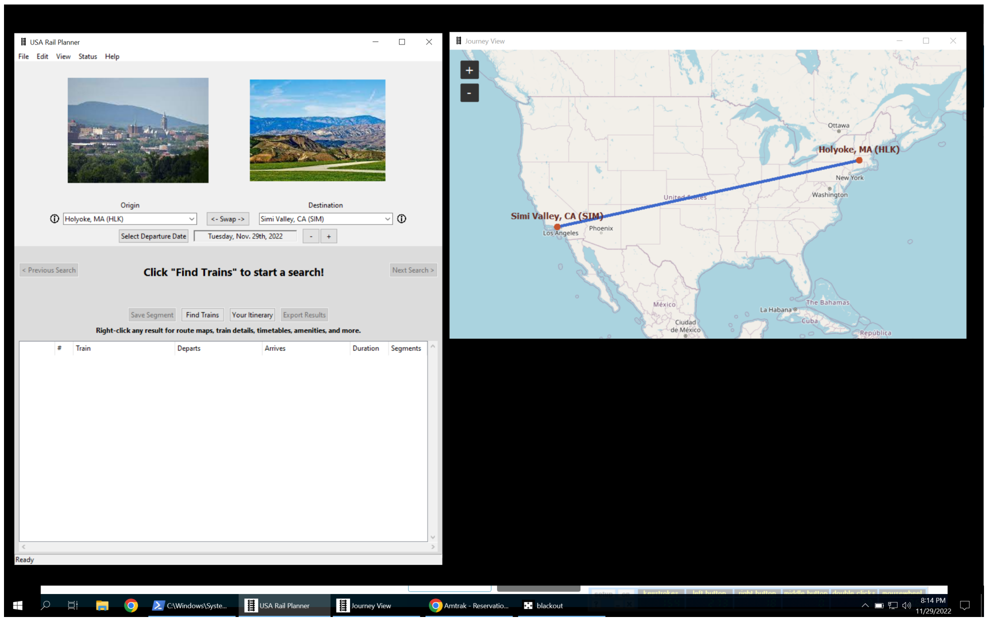


Figure 6.5: USA Rail Planner Task Setup in VM

Group 2 participants started with the USA Rail Planner method, shown in Figure 6.5. The USA Rail Planner main window and map are displayed, with the Blackout cover behind them. This obscures the web driver running in the background that searches the Amtrak website. A small sliver of the "Back" and "Continue" buttons can be seen at the bottom of the figure. The Windscribe VPN is also running in this task to prevent Amtrak from blocking the web driver usage. The Rail Planner is the only allowed application in this task, however, participants were informed that a web browser may open for some functions in the application (such as timetables or tourism information), and that this is considered part of the application. They were not allowed to interact with the web browser, the page was for viewing only. If a searching error occurred due to some problem with the web driver and/or Amtrak site, participants were told to try the search again. If that did not work, the researcher cleared the web driver cache and reset the Blackout cover.

At the end of a task, participants were allowed an intermission to get a drink or use the restroom. During that time, Mousotron metrics were recorded, and any open files were saved for later review. The second task would then be prepared, and the process would repeat. After both tasks were completed, a post-study questionnaire was provided to the participant, and they were told it was meant to gauge the effectiveness of the applications they encountered during the study. This questionnaire's content is listed in Table 6.4. Finally, after this was completed, participants were dismissed from the study.

6.4 Study Results

To analyze the Time to Complete Itinerary and Trivia (TTC) and Trivia Accuracy, a one-way ANOVA was used. The Shapiro-Wilk test was used to ensure the data was normal. For all other analysis, as the data was determined to not follow a normal distribution, both the Wilcoxon Signed-Rank test and Kruskal-Wallis test were used to establish statistical significance. Post-hoc analysis to review the effectiveness of counterbalancing was done with a paired two-tail t-test and the Kruskal-Wallis test for normal and non-normal data, respectively. For all statistical measures, $\alpha = 0.05$ was used.

Scoring for the task and trivia accuracy followed a set of rules that were applied evenly across the participant results. For trivia, empty answers were a zero, wrong answers were 0.25, partially correct answers or ones answered correctly but with the wrong methodology received 0.5. One example of the latter is responding that a train did or did not stop at a certain station, and they were correct, but did not verify the information empirically like checking the route map or timetable. Correct answers received one point. Task accuracy followed a similar system, but entire legs could

Table 6.4: Post-Study Questionnaire

Question		**Response**
1a/1b	How confident are you that you can plan a multi-city train trip now after using USA Rail Planner/Amtrak website and spreadsheet?	1-7 scale
2a/2b	To what extend did you feel frustrated while using USA Rail Planner/Amtrak website and spreadsheet?	1-7 scale
3a/3b	How much effort did it take to plan an itinerary using USA Rail Planner/Amtrak website and spreadsheet?	1-7 scale
4a/4b	How easy was it to learn and use USA Rail Planner/Amtrak website and spreadsheet?	1-7 scale
5a/5b	How confident are you that the answered trivia questions and your itineraries are correct for USA Rail Planner/Amtrak website and spreadsheet?	1-7 scale
6	How effective was the route map in the USA Rail Planner for visualizing a train route?	1-7 scale
7	How effective was the "Save Segment" button in indicating that the highlighted result would be saved to your itinerary?	1-7 scale
7a	Optional - Suggest an alternate name?	Write-in
8	Using what application, in your view, took you less time to complete the tasks assigned to you?	USA Rail Planner; Amtrak website and spreadsheet; About the same
9	After using both methods of itinerary planning, what is your preference?	USA Rail Planner; Amtrak website and spreadsheet; About the same
10	What improvements or changes, if any, would you make to the USA Rail Planner?	Write-in
11	Do you have any other comments or suggestions?	Write-in

Table 6.5: User Study Hypotheses

Hypothesis	
H1	The USA Rail Planner will help users create itineraries faster compared to the Amtrak + Excel method.
H2a	Users will more accurately create itineraries using the USA Rail Planner than Amtrak + Excel. [Task]
H2b	Users will more accurately gather information using the USA Rail Planner than Amtrak + Excel. [Trivia]
H3	Users will view the USA Rail Planner as more difficult to use than the Amtrak + Excel method.
H4	Users will prefer the USA Rail Planner over the Amtrak + Excel method.

be marked as zero if the wrong result was found, such as incorrect station selections or not following the listed stipulation for the task step. An additional point can be awarded for itineraries if they were in chronological order. It should be noted that task partial credit was only available for the Amtrak + Excel method, as this requires manual entry, and several individual attributes could be right or wrong. Compare this to the Rail Planner, where the application saves all information (and its attributes) automatically, so the correctness is solely dependent on the user saving the correct result.

There are four hypotheses for the study. These are listed in Table 6.5. The first theorizes that users will create itineraries faster using the USA Rail Planner, and for this, the time to complete the trivia and task combined was used as the metric. The second hypothesis originally predicted that accuracy would be improved, but this has been split into two, more granular hypotheses. Hypothesis 3, that the Rail Planner would be more difficult to use, is based on the potential unfamiliarity with the Rail Planner, and Hypothesis 4 states that users will still prefer it, despite the predicted difficulties.

Table 6.6: Counterbalancing Effectiveness for Quantitative Results

Metric	**Test**	**USA Rail Planner**	**Amtrak + Excel**
TTC	t-Test (two-tail)	$p = 0.679$	$p = 0.339$
Task Accuracy	Kruskal-Wallis	$H = 0.026 < p = 3.841$	$H = 3.103 < p = 3.841$
Trivia Accuracy	t-Test (two-tail)	$p = 0.296$	$p = 0.348$

Table 6.7: Quantitative Results Analysis

Metric	**Test**	**Result**
TTC	ANOVA	$F = 46.402, p <$ 5e-6
Task Accuracy	Kruskal-Wallis	$H = 3.853 > p = 3.841$
Trivia Accuracy	ANOVA	$F = 4.617, p < 0.05$
Keystrokes	Kruskal-Wallis	$H = 17.280 > p = 3.841$
Mouse Interactions	Kruskal-Wallis	$H = 14.301 > p = 3.841$

6.4.1 Quantitative Results

A variety of compelling quantitative results were obtained from the user study. However, before discussing these, the validity of the data must first be examined. As mentioned, the Shapiro-Wilk test was employed to determine if each metric's data followed a normal distribution, and only TTC and Trivia Accuracy fit this criterion. However, the effect of counterbalancing needed to be consulted first. If the counterbalancing did not work, and a learning effect was introduced, the validity of the claims being made would be called into question. However, for the three quantitative metrics that would be affected, counterbalancing was determined to be successful for all of them, and the results of the tests are displayed in Table 6.6. That is, there was no significant difference between the means of the two groups for each metric.

With the counterbalancing in place and the normality of the data determined, appropriate analysis was conducted and revealed significant benefits of using the Rail

Planner. Table 6.7 shows the results for one-way ANOVA and Kruskal-Wallis tests for each metric and application. TTC provides the clearest indication of statistical significance. The Rail Planner took 47% less time to complete the itinerary and trivia compared to the traditional method of using the Amtrak website with an Excel spreadsheet, shown in Figure 6.6. Additionally, the Rail Planner TTC spread was nearly half that of Amtrak + Excel as well. This may indicate that the Rail Planner is easier to pick up and learn regardless of skill level, since the finish time fell within less than 8 minutes on either side of the mean, compared to Amtrak + Excel's over 13-minute spread. Similarly, task and trivia accuracy had statistical significance, if only slightly, showcased on the left in Figure 6.7. Mean task accuracy for the Rail Planner was nearly 7% higher than Amtrak + Excel, and trivia accuracy had an even greater difference at just under 19% - a 32% increase.

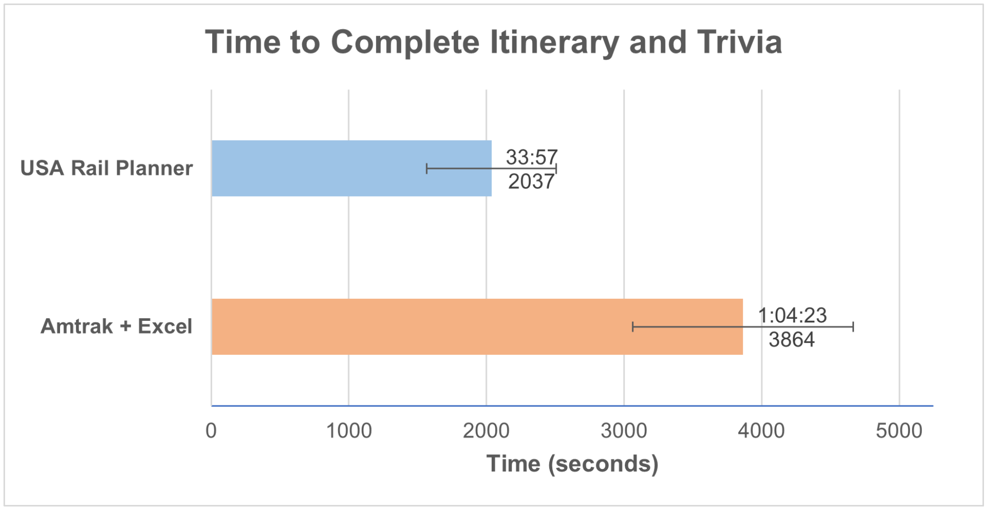
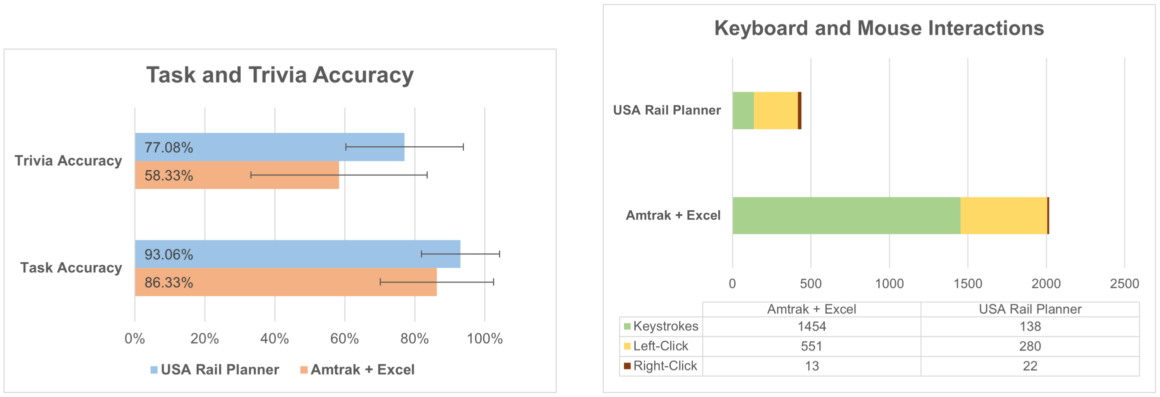


Figure 6.6: Mean Time to Complete

Secondary analysis was performed for keyboard and mouse interaction, displayed on the right in Figure 6.7. Both keyboard and mouse (left-click and right-click) interactions while using the Rail Planner were significantly lower than the Amtrak + Excel method, with keystrokes being reduced by 91%. One may conclude from this that, in addition to the lower TTC, significantly less input is required from the user to accomplish their goal of creating an itinerary.



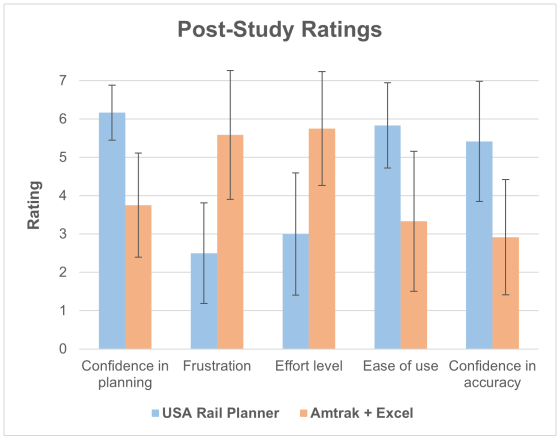
	Amtrak + Excel	USA Rail Planner
🟩 Keystrokes	1454	138
🟨 Left-Click	551	280
🟫 Right-Click	13	22

Figure 6.7: Mean Task/Trivia Accuracy and Keyboard/Mouse Interactions

6.4.2 Qualitative Results

The pre- and post-study questionnaires provided valuable participant data, not only for demographics, but also for application sentiments. Like the quantitative data, user ratings were put through a Shapiro-Wilk test, and it was determined that none of the samples were normally distributed. As such, all two-group rating results from qualitative analysis used a Kruskal-Wallis test to verify significance. For the single group ratings, it was found that, out of 12 participants, 10 thought that the Rail Planner took less time than the Amtrak + Excel method, and 11 preferred using the Rail Planner. This can be seen in Figure 6.8 on the right. Additionally, the found the route map and “Save Segment” button in the Rail Planner to be effective, with mean ratings of 5.9 and 5.2, respectively. However, this does not reflect user experiences with those two elements, covered in the later discussion section.

The meat of the post-study questionnaire was the ratings when comparing the two methods. Users were asked five questions about their experience with both the USA Rail Planner and the Amtrak + Excel methods, each with their own ratings. The results of these questions are shown on the left in Figure 6.8 and listed in Table 6.8. Each of the differences in ratings between the two methods were determined to be statistically significant. Rail Planner scored higher than Amtrak + Excel for the participants’ confidence in their ability to plan a train trip, its ease of use, and their confidence in the correctness of their completed itinerary and trivia questions. Rail



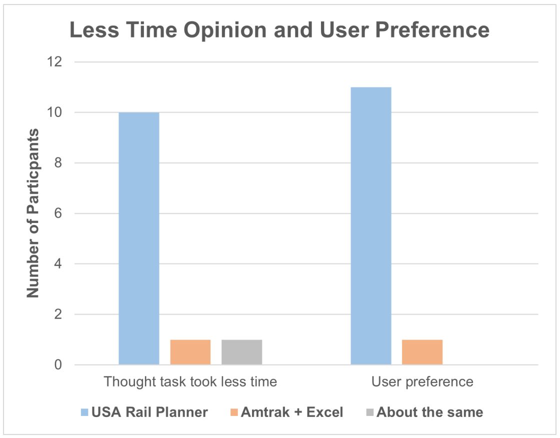


Figure 6.8: Mean Post-Study Ratings and User Time Perception/Preferences

Table 6.8: Qualitative Results Analysis

Metric	**Test**	**Result**
Confidence in planning	Kruskal-Wallis	$H = 14.083 > p = 3.841$
Frustration	Kruskal-Wallis	$H = 12.000 > p = 3.841$
Effort level	Kruskal-Wallis	$H = 9.363 > p = 3.841$
Ease of use	Kruskal-Wallis	$H = 9.363 > p = 3.841$
Confidence in accuracy	Kruskal-Wallis	$H = 9.901 > p = 3.841$

Planner also scored lower when participants were asked how frustrated they were while using it and the amount of effort required to complete the task. Clearly, in addition to the empirical evidence that Rail Planner is faster and more accurate than the Amtrak + Excel method, users viewed it more favorably.

Perhaps the most critical evaluation in the study was the participants' assessment of their confidence in planning a multi-city train trip. In the pre-study questionnaire, they were asked to rate their confidence in this matter, not having seen either method yet. Then, after the testing is concluded, the post-study questionnaire asks them about their confidence level after having used both methods. As seen in Figure 6.9, users were significantly more confident in their ability to plan a multi-city train trip after using the USA Rail Planner. The Amtrak + Excel method, while slightly higher, was not of any significance.

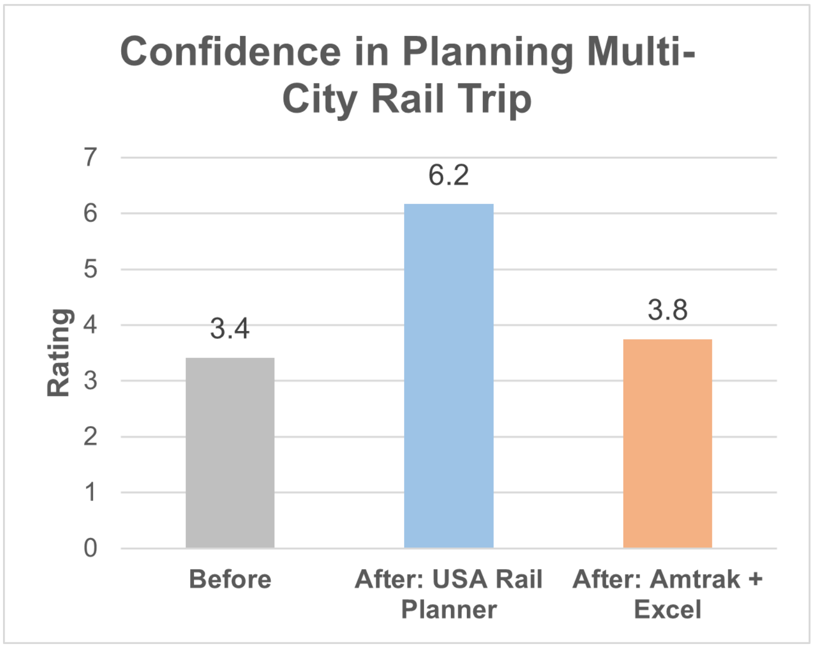


Figure 6.9: Confidence in Planning Before and After Study

Participant Suggestions

Finally, participants were given the opportunity to recommend improvements or changes to the Rail Planner. Many responses related to station selection, the "Save Segment" functionality, search duration, and interface clarity.

First, the autocomplete feature caused some issues for participants. One such example was in Step 1, wherein they needed to find a train to Raleigh, NC. Users often typed in exactly "Raleigh, NC" and no results were returned, as the actual station name was "Raleigh Union, NC" for its Union Station status.

Second, the "Save Segment" button caused a great deal of confusion. While the wording makes sense to a frequent Amtrak traveler, for those less familiar, it was unclear. Naming it "Save to Itinerary" or similar would likely have made it much clearer as to the functionality. Additionally, it did not reflect that anything had happened when the user clicked it, such as greying itself out or showing a success message. In most studies, participants would click the button twice (to be greeted with the "already saved" error message), view the Itinerary to verify the trip was added, or both.

Third, the duration of train searches was brought up frequently, however, they were unaware that it used Amtrak's website in the background to conduct the searches. They are right in one aspect, in that the results cannot be displayed in the Rail Planner until the page has finished loading, whereas the Amtrak + Excel method allows users to view results before the page completely loads.

Fourth, interface clarity was mentioned by about half of the 12 participants. This mainly related to the station Detail View (accessible via the "i" icon next to the dropdown menus or by clicking its name in the map window) and right-click train menu. Admittedly, the info icon is quite small, and, as one participant noted, blended in due to its black color. They remarked that had it been a different color, such as blue, it would have indicated more clearly it was an interactable element. The right-click menu was also hard to find for some in the study, despite the message below the "Find Trains" button that right-clicking any result will bring up more information (paraphrased). However, user experimentation was key in discovering these elements. Often, users would mistakenly click a station name in the map window but discover that it opened the station Detail View, containing potential answers to trivia questions. And, despite the Itinerary having no such indication that a right-click menu was available, participants still right-clicked on those results and were pleased to find the menu also existed there.

6.5 Discussion

Users were not satisfied with the Amtrak + Excel method. This is in part due to how the information is displayed (or not) on the search page and the laborious task of transcribing information to the spreadsheet. Participants in Group 1 (Amtrak + Excel first) frequently commented that the Rail Planner was much easier after realizing that results can be saved with the click of a button. Group 2 participants (Rail Planner first) sometimes became disappointed when they discovered that they would need to type in all the search results from the Amtrak website.

In terms of clarity, participants were at times confused by Amtrak's search results page, specifically what each element in a result represented. Issues were the train number (and sometimes its name), number of segments (unsure what “Direct” meant in that context), and the transfer stations. Compare this to the Rail Planner, which utilizes a table view with clearly labeled columns. However, the Rail Planner did have some column-related pain points. Despite the “Edit Display Columns” menu option, none of the participants utilized any menu options or bothered to check what was in them. This related to the task steps for deleting specific segments. Performing the deletion was not the issue, as the “Delete” button was clearly displayed in the Itinerary's buttons row, it was figuring out which train to delete. The origin and destination columns are not displayed by default, and the column settings affect both the search results table and Itinerary. Due to this, participants would sometimes delete the wrong train. Most, however, realized they could access the right-click menu and access the train's Detail View to verify the origin and destination. This was an unintended side-effect of the Itinerary design, but it was interesting to see how participants determined the appropriate train.

The trivia questions were often a pain point for participants. Usually, they would disregard the document or forget it was there until completing all the task steps. Only then would they realize that they needed to complete it simultaneously and would go back and re-search for the trains involved in the questions. With the second task, they corrected this error and would consult the trivia more often. Thanks to the counterbalancing, these errors did not appear to have any influence on the overall accuracy. The Rail Planner was meant to alleviate this task by including easily accessible route maps and train details from the Itinerary and search results, and this was used for the most part. Of note was Step 7a for both tasks. The wording was chosen (“those results”) to inspire users to use the search navigation features in the Rail Planner, however, only a few participants realized this was an option, the rest opted to perform another search.

Interestingly, despite never having used the Rail Planner before, most participants almost immediately discovered and used the autocomplete feature in the station dropdown lists. Some expected the results to auto-populate as they typed, but realized they needed to hit the Enter key or click the dropdown again. In some cases, the station name was typed in but not selected from the list. It was hoped users would realize their error as the city image and map markers did not update, but some still performed a search. The search heading also listed the previously (properly) selected stations but eventually the participants realized their mistake. One very interesting thing some of the participants did related to Rail Planner's Step 4, where they must find a train to New York, NY. A station name was not specified, and there are two entries in the list for "New York": New York Penn, NY (NYP), which is the correct station, and New York State Fair, NY (NYF). One participant did start a search to the latter location but realized their error when no train results were returned (no service was available between the WAS and NYF). More commonly, they would select one result or the other and consult the map window to see where the marker was placed. They could see New York (the city) on the map, and since New York State Fair was in upstate New York state, they switched their destination to New York Penn, which placed a marker squarely in downtown New York City.

6.5.1 Hypotheses

There were four hypotheses presented in Section 6.4 in Table 6.5. First, it was theorized that the Rail Planner would take less time to create an itinerary than the Amtrak + Excel method. Based on the significant time savings presented in the quantitative results, the null hypothesis (they take the same amount of time) can be safely rejected and H1 can be accepted. Second, accuracy in task and trivia was predicted to be better with the USA Rail Planner than Amtrak + Excel. Once again, based on the results, H2a and H2b are accepted. Third, the Rail Planner was thought to be more difficult to use for the participants. This was a surprising result, as based on qualitative evidence, participants found the Rail Planner easier to learn and use than the Amtrak + Excel method. Thus, the null hypothesis, that users will not find the Rail Planner more difficult to use than Amtrak + Excel, cannot be rejected. Consequently, the third hypothesis H3 cannot be accepted. However, this is a positive outcome. Fourth, and finally, it was predicted that users would prefer using the USA Rail Planner to plan itineraries over the Amtrak + Excel method. Over 91% of participants (11 out of 12) did prefer it. Based on this, H4 can be accepted.

Chapter 7

Conclusions and Future Work

In this final chapter, the application's contributions and impact are discussed in Section 7.1. This section covers key points from the thesis. Finally, Section 7.2 rounds off the thesis and covers possible future work. There are multiple avenues of expansion presented, with varying ranges of complexity and use.

7.1 Conclusions

The application described in this thesis, the Rail Planner, has provided an efficient way to plan a multi-city train trip and explore multiple potential routes. Due to issues with existing solutions, particularly with saving search results, the application presents itself as a breakthrough for rail trip planning in the United States. With a web scraping solution underneath, the user is never taken out of the app while the hard work of searching is done in the background. The application is built in Python, using Selenium for scraping and Tkinter for the user interface, allowing for cross-platform compatibility.

The research provides numerous contributions in the field of itinerary and trip planning. Integrating the web driver to perform searches keeps users in one place for finding information. Results are parsed and able to be saved to a user's itinerary, which can be exported to a file, thus eliminating the need for manual transcription of any information discovered. Saved (past) search results allow for an easy review of travel options without the need to perform a search again, a significant time-saving improvement over existing methods. Route maps with intermediate stops provide helpful context as to the train's journey and can provide ideas for destinations along the way, something not nearly as prevalent as with a list. Train details provide a wealth of information, from broad to granular, about a selected search result, including information users could not normally view otherwise, such as available seats. Station details provide users with valuable information about connecting routes and local transit options. Minor contributions are also present in the application in a variety of areas. Timetables, something traditionally difficult to find since Amtrak stopped producing digital versions, are included in the View menu or from individual results. Picture displays of selected locations provide additional context for the user, while also adding some flair and color to the main window. Finally, saving and loading progress allows users to work at their own pace without losing any of their past research. While the application is an Amtrak website searcher underneath, its additional features bring it from an interface for the website to a novel, feature-packed assistant for train travel planning.

7.2 Future Work

The USA Rail Planner has interesting avenues for expansion. Highly requested features from survey respondents relate to station and train information. For example, station information could include details about the surrounding area, such as distance to the city center, available lodging or parking, places to eat, or amenities at the station. In this way, the application could expand beyond a rail planning service and turn into a trip planning service, with the distinction coming from the inclusion of city information and lodging. However, more complex features can be added.

7.2.1 Improved Search Functionality

Perhaps the most critical improvement that could be made to the application is the removal of the web scraping requirement for searches. It is the weakest link in the application, and, as discussed, could fail if the Amtrak website undergoes significant changes. As such, methods to remove or mitigate this necessary functionality could be iteratively implemented. First, the application would need to store route data in a database of some kind, which would be packaged with the application. Route data, in this context, refers to specific train/bus trips. Each trip is provided a number by Amtrak, and it corresponds to specific trains or buses with specific routes and departure/arrival times. Of course, this can succumb to data becoming out-of-date, so this could be implemented in conjunction with a web service to pull updated data. Routes and their stops are included in the application, but schedules are not. GTFS data is perfect for this kind of implementation, although the amount of information provided would need to be examined closer. However, if GTFS could be used with the application, it could provide route map data, timetables, and other critical information. Second, a routing algorithm would be necessary to provide accurate results. While it can't be guaranteed that it will exactly match what Amtrak would provide for the same search, it is reasonable to assume they would use a "shortest-path/time" algorithm when providing search results. Finally, with routes pre-loaded and a routing algorithm in place, the true potential of the application could be realized: automatic trip generation. Given a date range and a list of destinations (ordered or unordered), it would generate the itinerary for the user without any other input. Specifying how many days to spend in a certain location could be specified as well, to allow for breaks in travel. The ease of creating an entire itinerary in seconds would be greater than that of searching the Amtrak site or using the application as-is.

Alternatively, the search speed could be increased by using Amtrak's Travel Planning Map search form. In preliminary tests, the results were found much quicker using this method compared to searching from the main Amtrak site and would serve as a useful speed increase in the interim period before implementing the aforementioned route database and search algorithm.

7.2.2 Interface Additions

There are three primary potential additions to the application that would further enhance the user experience, at the cost of some increased complexity. First, thruway bus destinations (ex. Las Vegas – has not had passenger train service since 1997 [46, 47], replaced with bus bridges) could be included in the stations list. However, with the numerous bus stops, it may introduce unnecessary clutter in the list, and a distinction would need to be made between types of stops. An option to show or hide these stops may be a good compromise. Second, a list of direct connections from any given station could be added, possibly in the station detail view. While connecting trains are provided, *destinations* are not. A user may see that the Coast Starlight is a connecting route from the Sacramento Valley Station but would not know where it went. Providing a list of connecting stations from the station would provide users ideas for non-stop trips. This is technically possible with the current application as stations are linked to specific route files, so matching connecting routes to connecting stations would not be difficult. However, this may make the station details list very long. A simpler solution would be adding a label in the station selection area when two selected stations are direct connections, or by a toggle to show non-stop destinations from the origin only. Third, station information, such as amenities, parking, and nearby stations, could be included in the station Detail View. This information can be retrieved from Amtrak's Travel Planning Map and would provide useful context for travelers.

There are four secondary possible additions to the application that are not necessarily increasing the complexity but would still provide a better user experience. First, cleaning up the exported itinerary would be useful for later viewing. It is saved as a CSV file currently, which does not contain information about column widths. This means the user would have to manually adjust column widths in their spreadsheet viewer of choice to view all the content in a given column. Saving as an Excel (XLSX) file could mitigate this issue. Additionally, legs of the journey with multiple segments should be listed as such. A leg with three segments should be represented as three rows, not one condensed row. This would increase clarity for what trains/buses comprise a leg of the journey. Second, the itinerary allows for reordering of segments, but does not consider departure and arrival times. Effectively, a train earlier in the journey may have a departure date after one of the later legs. Highlighting “problem” entries in the table would be a safeguard against out-of-order itineraries. Third, the date increment buttons (-/+) are fairly small, and the text size could be increased substantially so they are more prominent. Fourth, and finally, is the inclusion of sorting in the results table. Users may want to choose a train with the shortest duration or the latest departure time, and, in a table of 20 or more results, they may spend too long identifying the correct train. By including a "sort” method, users could more quickly identify and save the trains they want. Alongside this could be the inclusion of filtering, wherein users could remove results from the table that do not meet certain criteria, such as mode of travel, WiFi access, or ADA boarding. This would also save time that would otherwise be spent checking the Detail View for results.

7.2.3 Time-to-Travel Map

One of the more complex ideas is including an interactive map that would show travel time to any point from a starting location. It could be color-coded based on the number of hours, such as green/blue for 1-3 hours and into orange and red as time increases. For the application, this would require the improvements relating to improved searches and the data that comes with it. Creating a map UI for it would be another challenge, but not impossible. A similar project in this domain is the Chronotrains map [48], mapping how far one can travel in five hours from a given station in Europe.

Bibliography

[1] *Select your trip.* URL: `https://www.amtrak.com/tickets/departure.html` (Retrieved 04/30/2022).

[2] *Multi-Ride & Rail passes.* URL: `https://www.amtrak.com/tickets/departure-rail-pass.html` (Retrieved 04/30/2022).

[3] *Freight Rail Facts & Figures.* 2022. URL: `https://www.aar.org/facts-figures` (Retrieved 10/01/2022).

[4] *Amtrak tickets, schedules and train routes.* URL: `https://www.amtrak.com/home` (Retrieved 04/30/2022).

[5] *Multi-city tickets.* URL: `https://tickets.amtrak.com/itd/amtrak/complexrail` (Retrieved 09/27/2022).

[6] *Amtrak Travel Planning Map.* URL: `https://www.amtrak.com/plan-your-trip.html` (Retrieved 11/25/2022).

[7] *Plan your rail trip in Europe.* URL: `https://www.eurail.com/en/plan-your-trip` (Retrieved 09/27/2022).

[8] *Discover how to get anywhere.* URL: `https://www.rome2rio.com/` (Retrieved 09/27/2022).

[9] *Search & Compare Cheap Bus and train tickets.* URL: `https://www.wanderu.com/en-us/` (Retrieved 09/27/2022).

[10] Piero Maddaleni. *How I created an NPM package to access Amtrak's semi-private API.* 2021. URL: `https://blog.replit.com/amtrak` (Retrieved 09/28/2022).

[11] Michael K. Svangren, Mikael B. Skov, and Jesper Kjeldskov. "Passenger trip planning using ride-sharing services". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018). DOI: `10.1145/3173574.3174054`.

[12] Chang-Shing Lee, Young-Chung Chang, and Mei-Hui Wang. "Ontological recommendation multi-agent for Tainan City Travel". In: *Expert Systems with Applications* 36.3 (2009), 6740–6753. DOI: `10.1016/j.eswa.2008.08.016`.

[13] Lanyun Zhang and Xu Sun. "Designing a trip planner application for groups". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (2016). DOI: `10.1145/2851581.2892301`.

[14] Zhong-Ren Peng and Ruihong Huang. "Design and development of interactive trip planning for web-based Transit Information Systems". In: *Transportation Research Part C: Emerging Technologies* 8.1-6 (2000), 409–425. DOI: `10.1016/s0968-090x(00)00016-4`.

[15] Avichal Garg. *Public transit via google.* 2005. URL: `https://googleblog.blogspot.com/2005/12/public-transit-via-google.html` (Retrieved 09/27/2022).

[16] Christopher Cherry, Mark Hickman, and Anirudh Garg. "Design of a map-based transit itinerary planner". In: *Journal of Public Transportation* 9.2 (2006), 45–68. DOI: `10.5038/2375-0901.9.2.3`.

[17] Chris Harrelson. *Happy trails with google transit.* 2006. URL: `https://googleblog.blogspot.com/2006/09/happy-trails-with-google-transit.html` (Retrieved 09/27/2022).

[18] *Mobility database.* URL: `https://database.mobilitydata.org/`.

[19] *Welcome to roadtrippers.* 2022. URL: `https://roadtrippers.com/` (Retrieved 09/27/2022).

[20] *Furkot: Free road trip planner: Map your route.* URL: `https://trips.furkot.com/ui` (Retrieved 09/27/2022).

[21] Thomas Adler. "Intercity Transit Trip Planning Web Application". In: *Transportation Research Board* (2014). URL: `https://trid.trb.org/view/1352933` (Retrieved 09/28/2022).

[22] Lisa Jo Elliott et al. "Guidelines and best practices for open source transit trip planning interfaces". In: *Advances in Intelligent Systems and Computing* (2016), 163–172. DOI: `10.1007/978-3-319-41685-4_15`.

[23] Konstantinos Zografos, Vassilis Spitadakis, and Konstantinos Androutsopoulos. "Integrated Passenger Information System for multimodal trip planning". In: *Transportation Research Record: Journal of the Transportation Research Board* 2072.1 (2008), 20–29. DOI: `10.3141/2072-03`.

[24] Samaneh Navabpour et al. "An Intelligent Traveling Service Based on SOA". In: *2008 IEEE Congress on Services - Part I.* 2008, pp. 191–198. DOI: `10.1109/SERVICES-1.2008.44`.

[25] Senjuti Basu Roy et al. "Interactive itinerary planning". In: *2011 IEEE 27th International Conference on Data Engineering.* 2011, pp. 15–26. DOI: `10.1109/ICDE.2011.5767920`.

[26] Sascha Witt. "Trip-based public transit routing". In: *Algorithms - ESA 2015* (2015), 1025–1036. DOI: `10.1007/978-3-662-48350-3_85`.

[27] Mihai Gheorghe, Florin-Cristian Mihai, and Marian Dârdală. "Modern techniques of web scraping for data scientists". In: *International Journal of User-System Interaction* 11.1 (2018), pp. 63–75.

[28] A. Hernandez-Suarez et al. *A Web Scraping Methodology for Bypassing Twitter API Restrictions.* 2018. DOI: 10.48550/ARXIV.1803.09875. URL: https://arxiv.org/abs/1803.09875.

[29] Arif Himawan, Adri Priadana, and Aris Murdiyanto. "Implementation of Web Scraping to Build a Web-Based Instagram Account Data Downloader Application". In: *IJID (International Journal on Informatics for Development)* 9.2 (2020), 59–65. DOI: 10.14421/ijid.2020.09201. URL: http://ejournal.uin-suka.ac.id/saintek/ijid/article/view/09201.

[30] Daniel Delling, Thomas Pajor, and Renato F. Werneck. "Round-based public transit routing". In: *Transportation Science* 49.3 (2015), 591–604. DOI: 10.1287/trsc.2014.0534.

[31] Peter Hart, Nils Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), 100–107. DOI: 10.1109/tssc.1968.300136.

[32] *List of Amtrak stations.* 2022. URL: https://en.wikipedia.org/wiki/List_of_Amtrak_stations (Retrieved 10/07/2022).

[33] John Sharp. *Amtrak official: Gulf Coast Service starting in 2022.* 2021. URL: https://www.al.com/news/2021/02/amtrak-official-gulf-coast-service-starting-in-2022.html (Retrieved 10/09/2022).

[34] *Amtrak • Operator details.* URL: https://www.transit.land/operators/o-9-amtrak#routes (Retrieved 11/01/2022).

[35] Python Software Foundation. *Python.* URL: https://www.python.org/ (Retrieved 11/03/2022).

[36] *Tkinter - Python interface to TCL/TK.* URL: https://docs.python.org/3.9/library/tkinter.html (Retrieved 11/03/2022).

[37] *Tkcalendar.* URL: https://pypi.org/project/tkcalendar/ (Retrieved 11/03/2022).

[38] TomSchimansky. *Tomschimansky/TkinterMapView: A python tkinter widget to display tile based maps like OpenStreetMap or Google satellite images.* URL: https://github.com/TomSchimansky/TkinterMapView (Retrieved 11/03/2022).

[39] *Selenium.* URL: https://www.selenium.dev (Retrieved 11/03/2022).

[40] SergeyPirogov. *Sergeypirogov/webdriver_manager.* URL: https://github.com/SergeyPirogov/webdriver_manager (Retrieved 11/03/2022).

[41] Ultrafunkamsterdam. *Ultrafunkamsterdam/undetected-chromedriver: Custom selenium chromedriver: Zero-config: Passes all bot mitigation systems (like distil / Imperva/ Datadadome / Cloudflare IUAM).* URL: https://github.com/ultrafunkamsterdam/undetected-chromedriver (Retrieved 11/03/2022).

[42] *Beautifulsoup4.* URL: https://pypi.org/project/beautifulsoup4/ (Retrieved 11/03/2022).

[43] URL: https://windscribe.com (Retrieved 11/29/2022).

[44] Ana Marculescu. *Download blackout 2.0*. 2013. URL: https://www.softpedia.com/get/Multimedia/Graphic/Graphic-Capture/AB-Blackout.shtml (Retrieved 11/29/2022).

[45] *Mousotron : Mouse and keyboard activity monitor*. URL: https://www.blacksunsoftware.com/mousotron.html (Retrieved 11/29/2022).

[46] Richard Simon. "Gone With the Wind: Demise of Desert Train Ends L.A.'s Amtrak Link to Las Vegas". In: *Los Angeles Times* (1997), VYA3–VYA33. (Retrieved 11/13/2022).

[47] Robert Johnston. *FRA launches passenger long-distance study site*. 2022. URL: https://www.trains.com/trn/news-reviews/news-wire/fra-launches-passenger-long-distance-study-site/ (Retrieved 11/24/2022).

[48] Benhamin Td. URL: https://www.chronotrains.com/ (Retrieved 07/29/2022).

Appendix A

Map Utilities

Listing A.1: Map Utilities Source Code

```
import requests
from bs4 import BeautifulSoup
from lxml import etree
from tkintermapview import convert_address_to_coordinates

def amtrakAddressRequest(stationCode: str) -> list[str]:
  """
  Given a station code, returns address of the station from a web
     ↪ request to Amtrak.

  Parameters
  ----------
  stationCode : str
      Amtrak station code (three letter string)

  Returns
  -------
  list[str]
      [Street number and name, City/State/Zip] or None if nothing was
          ↪  found.
  """
  # Find station page
  # Parse for address
  try:
    webinfo = requests.get(f"https://www.amtrak.com/stations/{
       ↪ stationCode.lower()}")
    soup = BeautifulSoup(webinfo.content, "html.parser")
    dom = etree.HTML(str(soup))
    try:
      addr1 = dom.xpath("//*[@class='hero-banner-and-info__card_block
        ↪ -address']")[-2].text
      _addr2 = dom.xpath("//*[@class='hero-banner-and-
        ↪ info__card_block-address']")[-1].text
    except IndexError:
      _addr2 = dom.xpath("//*[@class='hero-banner-and-
        ↪ info__card_block-address']")[1].text
    addr2 = _addr2.replace('␣␣','').replace('\r\n', '␣')
    return [addr1, addr2]
  except:
    return None

def getCoords(code: str) -> list[float]:
  """
  Finds coordinates of an Amtrak station.

  Parameters
  ----------
  code : str
      Amtrak station code, three letter string.

  Returns
  -------
  list[float]
      [Latitude, Longitude] or None.
  """
  address = amtrakAddressRequest(code)
  try: coords = convert_address_to_coordinates(f"{address[0].strip()
      ↪ },␣{address[1].strip()}")
  except (TypeError, IndexError):
    coords = None
    print(f"Could␣not␣find␣coordinates␣for␣station␣{code}:␣{address}"
        ↪ )
  if coords == None:
    try:
      _isCA = None
      canadianStates = ['ON', 'QC', 'NS', 'NB', 'MB', 'BC', 'PE', 'SK
        ↪ ', 'AB', 'NL']
      for state in canadianStates:
        if state in address[1]:
          _isCA = state
          break
      if _isCA != None:
        address[1] = address[1].split(_isCA)[0]+_isCA
        coords = convert_address_to_coordinates(f"{address[0]},␣{
           ↪ address[1]},␣Canada")
      else:
        coords = convert_address_to_coordinates(f"{address[1]},␣
           ↪ United␣States")
    except (IndexError, TypeError):
      coords = None
      print(f"Could␣not␣find␣coordinates␣for␣station␣{code}:␣{address
         ↪ }")
  return coords
```

Appendix B

Web Scraping Code Example

Listing B.1: Searching and Parsing Code Snippet

```
import time
import json

from random import randint

from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By

from .driver import Driver
from traintracks.train import Train
from views import config as cfg

USE_TRAIN_CLASSES = True

def __checkEveryPage(self, area, pages: int, isScrape: bool=True) ->
   ↪  None:
  """
  Checks every page of search results. If there is one, it does not
     ↪ loop.

  Parameters
  ----------
  area : WebElement
      Search area at the bottom of the page where the page links are
         ↪ located.
  pages : int
      Number of pages of results.
  isScrape: bool, Optional
```

```
        Whether to use the old webscraping method (True) or the new
            ↪ JSON method (False), by default False
    """

    def scrapingMethod():
      for page in range(1,pages+1): # Starts at 1 (page 1) and goes up
          ↪ to pages
        self.__updateStatusMessage(f"Searching␣-␣checking␣page␣{page}",
            ↪  29./pages)
        self.driver.execute_script("window.scrollTo(0,␣document.body.
            ↪ scrollHeight)") # Puts page links in view

        # Loads next page and waits until elements load
        area.find_element(By.XPATH, f".//*[text()='{page}']").click()
        WebDriverWait(self.driver, 5).until(EC.
            ↪ presence_of_element_located((By.XPATH, f".//a[text()='{
            ↪ page}']//ancestor::li[@class='pagination-page␣page-item␣
            ↪ active␣ng-star-inserted']")))
        time.sleep(1)

        searchResultsTable = self.driver.find_element(By.XPATH, "//div[
            ↪ contains(@class,␣'trigger-searchList')]") # Search area
        trainList = searchResultsTable.find_elements(By.XPATH, ".//div[
            ↪ contains(@class,␣'trigger-searchItems')]") # List of
            ↪ train results
        self.__findTrainInfo(trainList)

    if isScrape:
      scrapingMethod() # We don't need this.
    else:
      if self.__processTrainJson() != True:
        scrapingMethod()

def __processTrainJson(self, file: dict=None) -> bool:
    """
    New method to get train search results from session storage JSON.

    Parameters
    ----------
    file : dict, optional
        For testing onlt, specify a results dictionary to use, by
            ↪ default None
```

```
Returns
-------
bool
    True, if the data was able to be parsed.
"""

try:
  if file == None:
    j = json.loads(self._getSessionStorage("searchresults", True))
  else:
    j = file
    self.__updateStatusMessage('Test', 22)

  _journeySolutionOption = j["journeySolutionOption"]
  _journeyLegs = _journeySolutionOption["journeyLegs"][0]
  _journeyLegOptionsMultiple = _journeyLegs["journeyLegOptions"] #
     ↪ All segments

  for index, opt in enumerate(_journeyLegOptionsMultiple):
    self.__updateStatusMessage(f"Processing␣results", 28./len(
       ↪ _journeyLegOptionsMultiple))
    try:
      if len(opt["reservableAccommodations"]) > 0: # Not sold out

        # Basic Data Gathering // Compare to __findTrainInfo()
        segmentType = len(opt["travelLegs"])
        if opt["isMultiSegment"] == True: #Multi-segment
          number = "N/A"
          name = "Multiple␣Trains"
          hasTrain = False
          hasBus = False
          for leg in opt["travelLegs"]:
            if leg["travelService"]["type"].upper() == "TRAIN":
              hasTrain = True
            elif leg["travelService"]["type"].upper() == "BUS":
              hasBus = True
          if hasTrain and hasBus: name = "Mixed␣Service"
          elif hasBus and not hasTrain: name = "Multiple␣Buses"

        elif opt["isMultiSegment"] == False: #Single segment
          number = opt["travelLegs"][0]["travelService"]["number"]
          name = opt["travelLegs"][0]["travelService"]["name"]
```

```
origin = opt["origin"]["code"]
destination = opt["destination"]["code"]

departure = opt["origin"]["schedule"]["departureDateTime"]
arrival = opt["destination"]["schedule"]["arrivalDateTime"]
travelTime = opt["duration"]

coachPrice = opt["coach"]["lowestPrice"]
businessPrice = opt["business"]["lowestPrice"]
sleeperPrice = opt["rooms"]["lowestPrice"]

# Initial data update
outputDict = {
  "Number":number,
  "Name":name,
  "Origin":origin,
  "Departure":departure,
  "Travel␣Time":travelTime,
  "Destination":destination,
  "Arrival":arrival,
  "Coach␣Price":coachPrice,
  "Business␣Price":businessPrice,
  "Sleeper␣Price":sleeperPrice,
  "Segments":segmentType,
  "Raw":opt
}

# Advanced Data Gathering
try:
  extra = {}
  for index, seg in enumerate(opt["segments"]):
    _thisAmenities = []
    for amenity in seg["travelLeg"]["travelService"]["
        ↪ amenities"]:
      _thisAmenities.append(amenity["name"])

    _thisNum = seg["travelLeg"]["travelService"]["number"]
    _thisName = seg["travelLeg"]["travelService"]["name"]
    _thisType = seg["travelLeg"]["travelService"]["type"]

    _thisDest = seg["travelLeg"]["destination"]["code"]
    _thisArrive = seg["travelLeg"]["destination"]["schedule"
        ↪ ]["arrivalDateTime"]
```

```
            _thisOrigin = seg["travelLeg"]["origin"]["code"]
            _thisDepart = seg["travelLeg"]["origin"]["schedule"]["
                ↪ departureDateTime"]

            _thisDuration = seg["travelLeg"]["elapsedTime"].replace(
                ↪ 'P','').replace('T','␣').replace('H', 'H␣')
            _thisSeatsAvailable = opt["seatCapacityInfo"]["
                ↪ seatCapacityTravelClasses"][index]["
                ↪ availableInventory"]

            extra[seg["travelLegIndex"]] = {
              "Name": _thisName,
              "Number":_thisNum,
              "Type":_thisType,
              "Origin":_thisOrigin,
              "Destination":_thisDest,
              "Departure":_thisDepart,
              "Arrival":_thisArrive,
              "Duration":_thisDuration,
              "Available␣Seats":_thisSeatsAvailable,
              "Amenities":_thisAmenities
            }

          citySegments = opt["citySegments"]

          outputDict.update({
            "City␣Segments":citySegments,
            "Segment␣Info":extra})

        except (KeyError, IndexError) as e:
          print(e)

        if USE_TRAIN_CLASSES:
          self.thisSearchResultsAsTrain.update({self.
              ↪ numberTrainsFound : (Train(outputDict))})
        else:
          self.thisSearchResultsAsDict[self.numberTrainsFound] =
              ↪ outputDict
        self.numberTrainsFound += 1
        self.__updateNumberTrainsLabel()
    except Exception as e:
      print(e)
```

```
  except (KeyError, TypeError) as e:
    print(e)
    return False
  if self.numberTrainsFound > 0: # Interim
    return True
  else: return False

def __enterStationInfo(self, area) -> None:
  """
  Fills in origin and destination fields.

  Parameters
  ----------
  area : WebElement
      Area that contains input fields.
  """
  # Entering departure station info
  fromStationSearchArea = self.driver.find_element(By.XPATH, "//div[
      ↪ @class='from-station␣flex-grow-1']")
  fromStationSearchArea.find_element(By.XPATH, "//station-search[@amt
      ↪ -auto-test-id='fare-finder-from-station-field-page']").click
      ↪ ()
  inputField1 = fromStationSearchArea.find_element(By.XPATH, "//input
      ↪ [@id='mat-input-0']")
  inputField1.clear()
  inputField1.send_keys(self.origin)
  WebDriverWait(self.driver, 5).until(EC.presence_of_element_located
      ↪ ((By.XPATH, "//button[contains(@aria-label,'From')]"))) #
      ↪ Station name autofilled
  time.sleep(randint(5,100)/100.)

  # Entering destination station info
  toStationSearchArea = self.driver.find_element(By.XPATH, "//div[
      ↪ @class='to-station␣flex-grow-1']")
  toStationSearchArea.find_element(By.XPATH, "//station-search[@amt-
      ↪ auto-test-id='fare-finder-to-station-field-page']").click()
  inputField2 = toStationSearchArea.find_element(By.XPATH, "//input[
      ↪ @id='mat-input-1']")
  inputField2.clear()
  inputField2.send_keys(self.destination)
  WebDriverWait(self.driver, 5).until(EC.presence_of_element_located
      ↪ ((By.XPATH, "//button[contains(@aria-label,'To')]"))) #
      ↪ Station name autofilled
  time.sleep(randint(5,100)/100.)

def __enterDepartDate(self, area) -> None:
  """
  Fills in departure date.

  Parameters
  ----------
  area : WebElement
      Area that contains the departure date input field.
  """
  departsArea = area.find_element(By.XPATH, "//div[@class='departs-
      ↪ container␣w-100']")
  departsArea.click()
  inputField3 = departsArea.find_element(by=By.XPATH, value="//input[
      ↪ @id='mat-input-2']")
  inputField3.clear()
  inputField3.send_keys(f"{self.departDate}\t") #Depart Date
  #searchArea.find_element(by=By.XPATH, value="//input[@id='mat-input
      ↪ -4']").send_keys("03/27/2022") #Return Date
  time.sleep(randint(5,100)/100.)

def _getSessionStorage(self, key: str, beCareful: bool=False) ->
    ↪ dict:
  """
  Retrieves an item from session storage.

  Parameters
  ----------
  key : str
      Item to retrieve.
  beCareful : bool, optional
      If True, do NOT refresh the page to get the results, by default
          ↪ False

  Returns
  -------
  dict
      Session storage item.
  """
  time.sleep(1)
  _tc = self.driver.execute_script("return␣window.sessionStorage.
      ↪ getItem(arguments[0]);", key)
```

```
  if _tc == None and beCareful == False:
    self.driver.refresh()
    time.sleep(2)
    _tc = self.driver.execute_script("return␣window.sessionStorage.
       ↪ getItem(arguments[0]);", key)
  return _tc

def oneWaySearch(self, isScrape: bool=False) -> dict:
  """
  Performs a search for Amtrak trains on a one-way journey.

  Parameters
  ----------
  isScrape : bool
      If True, use the old scraping method, otherwise use the JSON
         ↪ method.

  Returns
  -------
  dict or str
      If the search is successful, returns a dict of trains,
         ↪ otherwise a string of the error message.

  Raises
  ------
  Exception
      Catch-all for any failed searches, returns the error message.
  """

  # Resets/clears elements to begin new search
  self.numberTrainsFound = 0
  self.thisSearchResultsAsTrain.clear()
  self.thisSearchResultsAsDict.clear()

  try: # Loading the page
    self.__updateStatusMessage("Searching␣-␣loading␣page", 1)
    if (self.driver.current_url != cfg.SEARCH_URL) or self.
       ↪ returnedError: # If not at the page or an error occurred
       ↪ last time, reload
      self.driver.get(cfg.SEARCH_URL)
    self.returnedError = False
    self.driver.execute_script("window.scrollTo(document.body.
       ↪ scrollHeight,␣0)") # Reset after previous search
```

```
# Make sure the page loads and the New Search button is available
   ↪  to us
WebDriverWait(self.driver, 5).until(EC.
   ↪ presence_of_element_located((By.XPATH, "//button[starts-
   ↪ with(@class,␣'am-btn␣btn--secondary')]")))

try: # Clicking the new search button
  self.__updateStatusMessage("Searching␣-␣opening␣input␣fields",
     ↪ 4)
  newSearchButton = self.driver.find_element(By.XPATH, "//button[
     ↪ starts-with(@class,␣'am-btn␣btn--secondary')]")
  newSearchButton.click()
  time.sleep(1)

  try: # Entering to/from stations
    self.__updateStatusMessage("Searching␣-␣entering␣stations",
       ↪ 2)
    searchArea = self.driver.find_element(By.XPATH, "//div[@id='
       ↪ refineSearch']")
    searchArea = searchArea.find_element(By.XPATH, "//div[starts-
       ↪ with(@class,␣'row␣align-items-center')]")
    self.__enterStationInfo(searchArea)

    try: # Entering departure date
      self.__updateStatusMessage("Searching␣-␣entering␣travel␣
         ↪ dates", 1)
      self.__enterDepartDate(searchArea)

      # Wait until "Find Trains" button is enabled, then click it
      self.__updateStatusMessage("Searching␣-␣starting␣search",
         ↪ 2)
      WebDriverWait(self.driver, 5).until(EC.
         ↪ presence_of_element_located((By.XPATH, "//button[
         ↪ @aria-label='FIND␣TRAINS'␣and␣@aria-disabled='false']
         ↪ ")))
      searchArea.find_element(By.XPATH, "//div[starts-with(@class
         ↪ ,␣'amtrak-ff-body')]").click() # Get calendar popup
         ↪ out of the way
      time.sleep(randint(5,100)/100.)
      searchArea.find_element(by=By.XPATH, value="//button[@aria-
         ↪ label='FIND␣TRAINS'␣and␣@aria-disabled='false']").
         ↪ click() # Click search button
    # Search has been completed, but there is no service
    try:
      self.__updateStatusMessage("Searching␣-␣starting␣search",
          ↪ 2)
      time.sleep(2)
      self.__updateStatusMessage("Searching␣-␣loading␣results",
          ↪ 3)
      potentialError = self.driver.find_element(By.XPATH, "//div
          ↪ [starts-with(@class,␣'alert-yellow-text')]").text
      print(potentialError)
      self.returnedError = True
      self.__updateStatusMessage("Error")
      return potentialError

    # Search has been completed, but we didn't find any trains
        ↪ on that day
    except NoSuchElementException:
      try:
        potentialError = self.driver.find_element(By.XPATH, "//
            ↪ div[@amt-auto-test-id='am-dialog']")
        newDate = potentialError.find_element(By.XPATH, ".//div[
            ↪ starts-with(@class,␣'pb-0␣mb-5␣ng-star-inserted')]
            ↪ ").text
        newDateError = '\n'.join(newDate.split("\n")[0:2])
        print(newDateError)
        self.returnedError = True
        self.__updateStatusMessage("Error")
        return newDateError

      # Search has been completed, and we found a train(s)
      except NoSuchElementException:
        try:
          self.__updateStatusMessage("Searching␣-␣parsing␣
              ↪ results␣page", 5)
          WebDriverWait(self.driver, 3).until(EC.
              ↪ presence_of_element_located((By.XPATH, "//div[
              ↪ contains(@class,␣'trigger-searchList')]")))
          searchResultsTable = self.driver.find_element(By.XPATH
              ↪ , "//div[contains(@class,␣'trigger-searchList')]
              ↪ ") # Table of results
          nextPage = searchResultsTable.find_element(By.XPATH, "
              ↪ .//ul[starts-with(@class,␣'pagination␣
                ↪ paginator__pagination')]") # Page links area
              numberSearchPages = int((len(nextPage.find_elements(By
                ↪ .XPATH, ".//*"))-4)/2) # Find out how many pages
                ↪  exist
              self.__checkEveryPage(nextPage, numberSearchPages,
                ↪ isScrape)
              self.__updateStatusMessage("Done", 50)

              #print(json.dumps(self.thisSearchResults, indent=4))
              return self.thisSearchResultsAsTrain

             except Exception as e: # Search parsing
              print("There␣was␣an␣error␣retrieving␣the␣search␣data."
                ↪ )
              print(e)
              self.returnedError = True
              return e
        except Exception as e: # Entering departure date
          print("There␣was␣an␣error␣entering␣the␣departure␣date.")
          self.returnedError = True
          return e
      except Exception as e: # Entering to/from stations
        print("There␣was␣an␣error␣entering␣in␣station␣info.")
        self.returnedError = True
        return e
    except Exception as e: # Clicking the new search button
      print("There␣was␣an␣error␣clicking␣the␣search␣button.")
      self.returnedError = True
      return e
  except Exception as e: # Loading the page
    print("There␣was␣an␣issue␣with␣loading␣the␣search␣page.")
    self.returnedError = True
    return e
```