DayTripper

George Dunnery, Bradford Egan, Jake Feinbaum, Vedanth Prakash

Northeastern University, Boston, MA, USA

Abstract

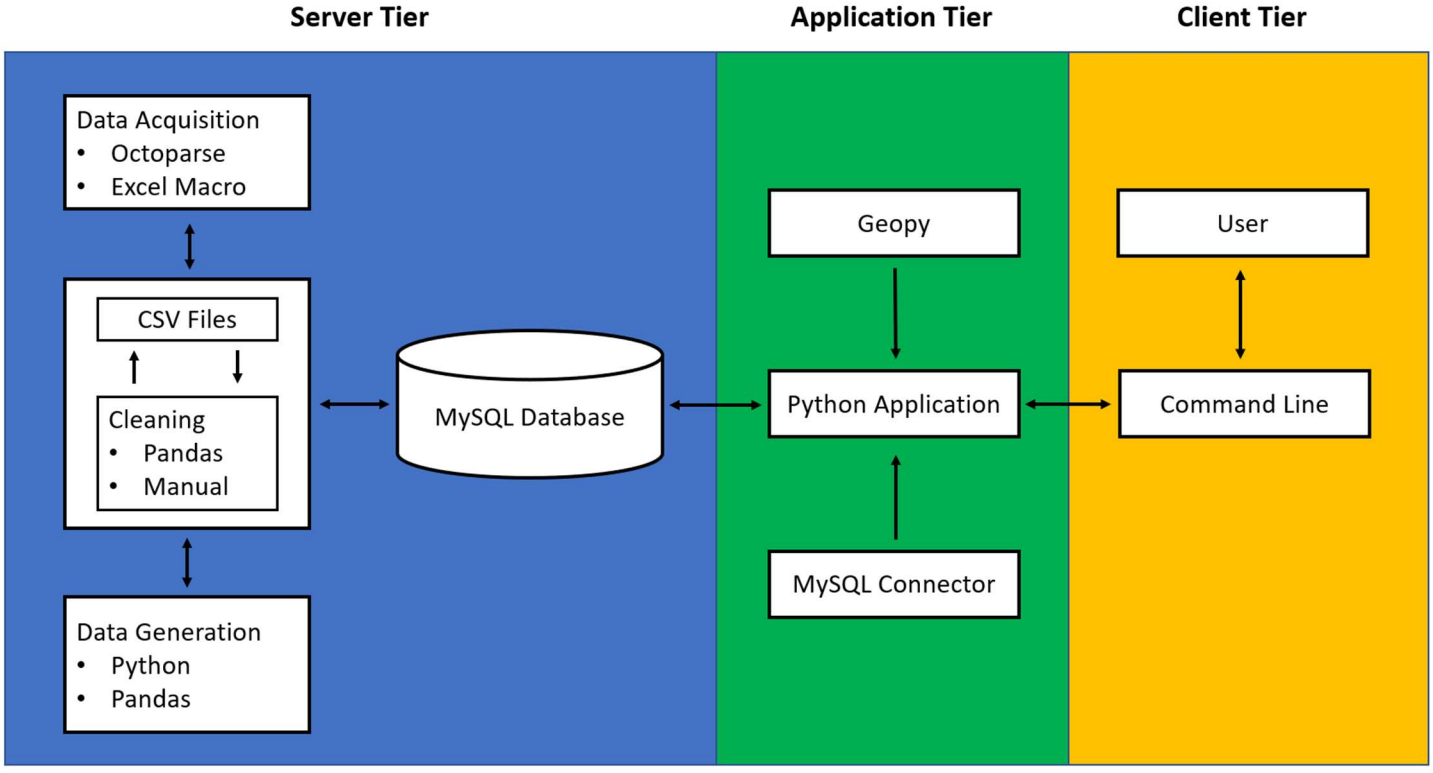
“DayTripper” is an itinerary building application written in Python that relies on a MySQL relational database to assist the end user in planning a trip. The goals of this project included the collection of data relevant to trip planning, the creation of a comprehensive database of activities and their features, the completion of a cross platform application, and to provide an itinerary building service to the user. DayTripper was successfully implemented as a command-line application which supports user registration, itinerary recommendation from a questionnaire, and the storage and retrieval of previously generated itineraries.

Introduction

Many applications and websites centralize metadata about venues (e.g. restaurants, museums, parks, etc.) or give comprehensive reviews and ratings, but planning a daily itinerary is generally left up to an end user to figure out for themselves. The goal of DayTripper is to provide an interactive itinerary builder as a service to an end user. This project is significant because no service currently exists to centralize the process of planning an itinerary by keeping track of all the minutia – it is an uncontested niche! First, the user will register as a user of the application. Then the user can enter key information (questionnaire) to have a trip recommended to them or search for other itineraries that have been saved to the database by other users. The questions will seek to uncover details about the user's group (family, professional, couple, casual, etc.), budget, destination, and general interests to narrow down the relevant activities. Furthermore, the application will dynamically account for details like the travel time between activities and hours of operation for the selected venues.

System Architecture

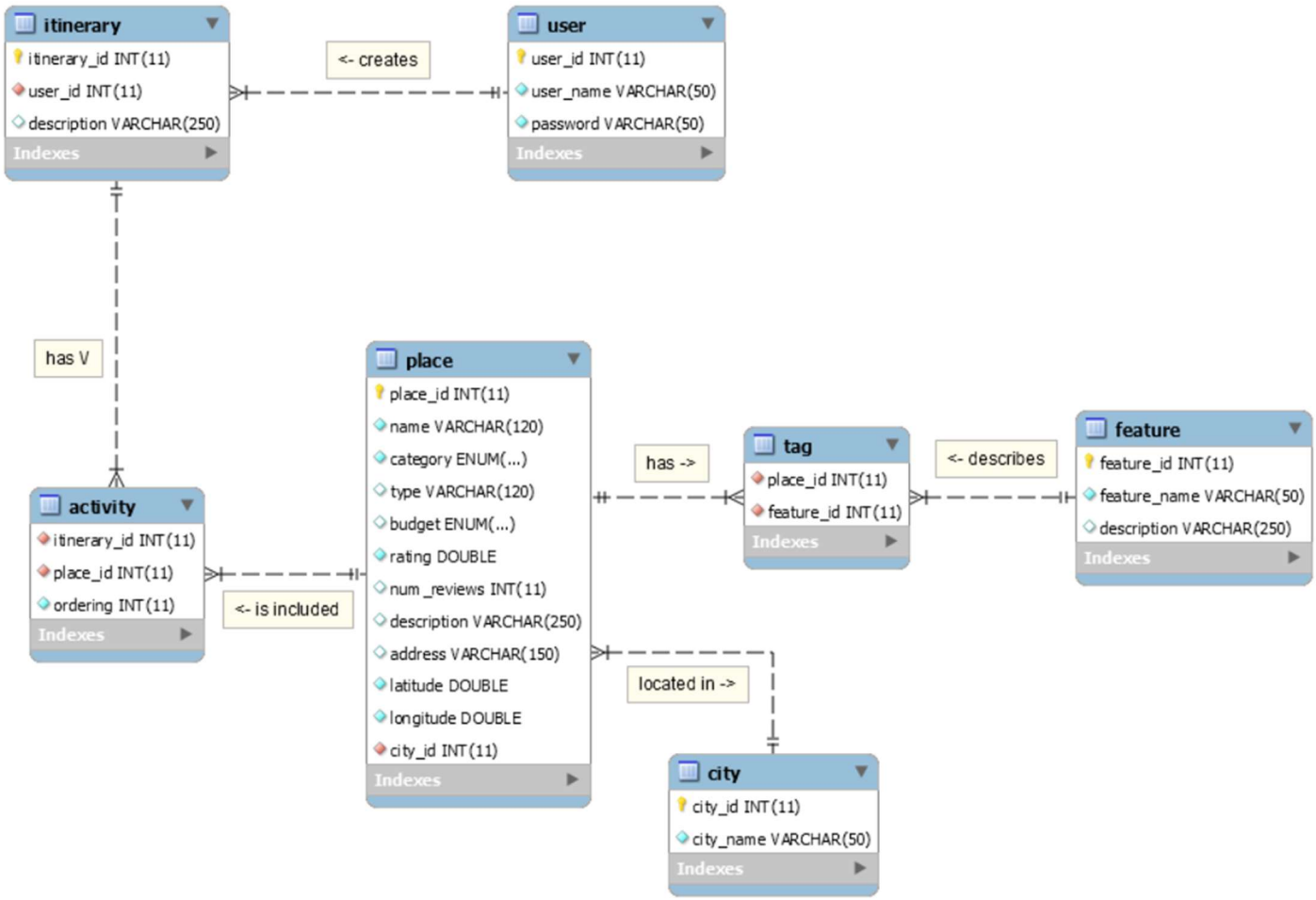
The system architecture is based around two main components: a MySQL relational database and an application written in Python. The database has been populated with data that was indirectly scraped from Google Maps using a tool called Octoparse (see Data Acquisition section). The end user will interact with a Python driven command-line application which relies on API functions to communicate with the database. The connection to the database is made possible by the MySQL connector package, and the travel times are estimated based on speed of transit and the Euclidean distance between two coordinates calculated by Geopy.



Database Design

The database originally consisted of several key tables which stored data about locations of interest: dining, park, hotel, museum, landmark, hotel, and entertainment (which includes venues such as bars, casinos, clubs, zoos, theaters, etc.). Each of these tables specified the name of the establishment and included a foreign key which referenced the city it's located in. The feature table had many-to-many relationships using join tables with each "main category" table. Features can be thought of as hashtags on Twitter. For example, a park may have features like "swimming pool", "playground", "hiking trails", and "fountain".

The design of the database was dramatically updated (after the presentation) during the process of integrating user registration and the storage of itineraries that could be used as templates in the future. The problem was that an itinerary table would require several join tables – one to each category of activity – to support the use of foreign keys. Supporting variable amounts of each type of activity introduced the need for more join tables or composite itinerary keys avoid violating the first normal form (lists). Therefore, the database design was revised with a greater focus on simplicity and scalability.



The solution to the problem is pictured above. A standard set of columns to support all categories of places was identified such that every place had the same attributes. Each of the main category names became an attribute along with the subcategories referred to as types. The budget column caused problems with null values since it had been defined as an enum of abstracted, "dollar-sign" style brackets ("$", "$$", "$$$"). To overcome this, a default value for budget was created as an empty string, which indicates that there is no budget since the activity is free (for example, visiting a park or landmark). In the future it will be much easier to include new places.

The features table serves the purpose of adding extra descriptive tags to the places stored in the database, and was not changed by the update. The tag table is a simple join table pairing the primary key of a place to the primary key of a feature. Between the main category and the list of features, it became apparent that the true type of venue could become lost among the other feature tags. The type attribute is meant to resolve this problem by acting as a subcategory that clearly represents the purpose of the venue. For example, a casino venue is considered 'entertainment' and may include many features (like restaurants, shopping malls, swimming pools, concert halls, etc.) that make it difficult to identify the specific type of venue.

To wrap up the database design section, the relationships between the entities will be enumerated. A user can create, store and retrieve itineraries. Itineraries are lists of at least one place, and places can be included in itineraries. The join table Activity supports the many-to-many relationship between Itinerary and Place. Places must be located in a particular city. Places are tagged with features, and features describe a place. The join table Tag supports the many-to-many relationship between Place and Feature.

Data Acquisition, Preparation and Generation

Data was initially intended to be acquired by scraping directly from Google Maps [1] using Python scripts and the Beautiful Soup package. This scraping technique failed because of how Google Maps [1] serves its information – data is not embedded in a readable format such as HTML. While there is an API that can be used for free for a limited number of queries, an alternative tool with no restrictions was found and used instead.

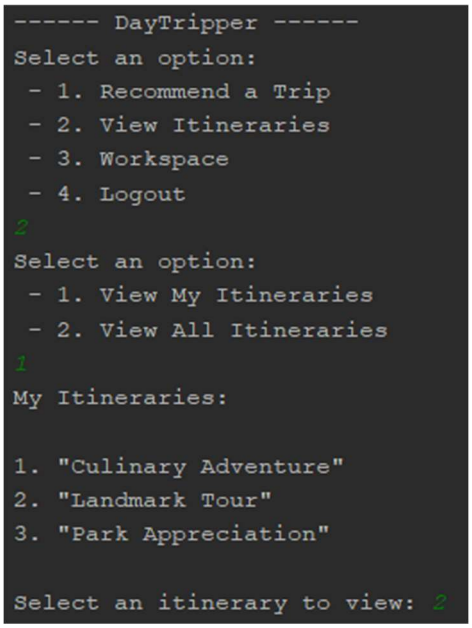
The data included in the final database was scraped using a program called Octoparse [2]. To gather data using Octoparse [2], a procedure must be defined within the application. For this project, the program is directed to repeatedly scrape the text from selected sections on the results of a Google Maps [1] search like "Restaurants in Boston" to generate the data for each of the attributes. Then the element which calls the next subpage of the search must be identified. The data was then mapped to a CSV file by Octoparse [2]. To support travel time estimation, all places in the database were subjected to an Excel Macro that gathered its latitude and longitude coordinates.

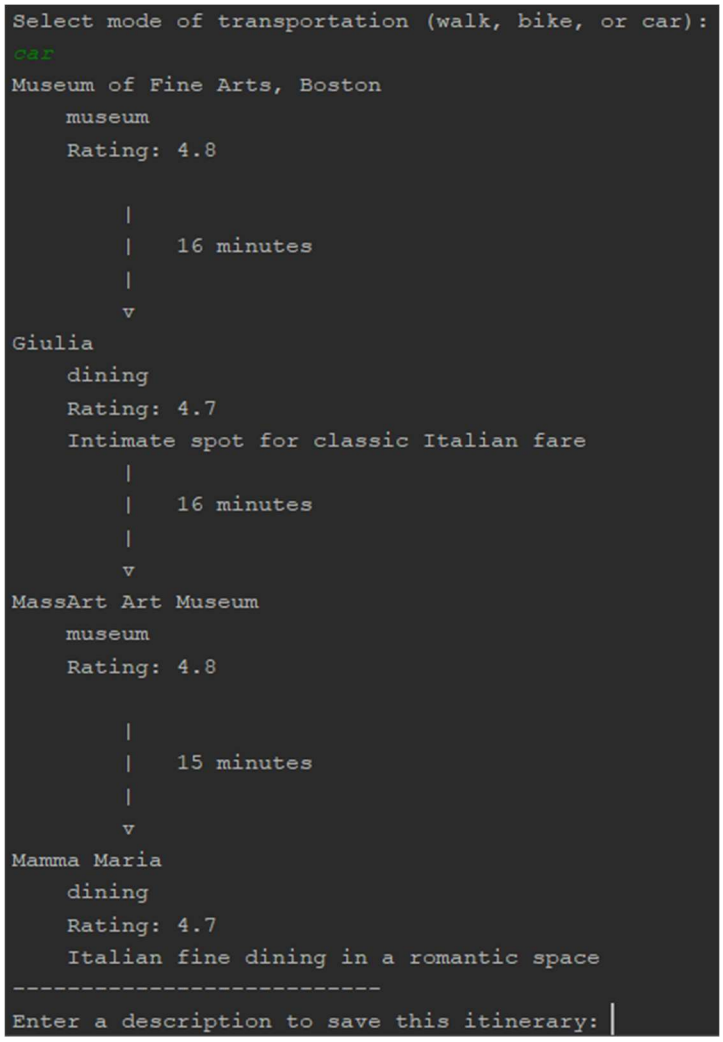
Data preparation (cleaning) was required before it could be imported into the database. Entries that were clearly wrong or otherwise unusable were deleted manually. The number of ratings each place received had been stored as a string with commas for readability, which caused problems when importing that attribute as an integer. To fix this, Pandas data frames were used to remove all commas from the rating counts.

Data generation was required for the purposes of having a complete database. The features table was populated with 74 features that were copied from Google Maps [1] into a CSV file. The tag table was generated by randomly assigning several randomly selected tags to each entry in the place table using Python and the random package. For places with missing budget values, Pandas was used again to fill in null values with a default average budget rating of '$$'.

User Interface

DayTripper has a command line interface. When the user opens DayTripper, they can choose to register as a new user or login with an existing username and password. Then the user is presented with a main menu consisting of 4 options: recommend a trip, view itineraries, workspace, and logout. Choosing "recommend a trip" will initiate the questionnaire in which the user provides information that is used to generate an itinerary. After completing the questionnaire, the full itinerary is printed out with estimated travel times. The user can then enter a description for the itinerary before it is saved to the database (the travel times depend on mode of transit and are therefore not stored in the database with the itinerary). The itinerary is also saved to a .txt file so the user has a local copy.





Choosing “view itineraries” allows the user to view previous itineraries that they have saved or to see a random selection of itineraries saved to the database by other users. This makes it easy to find old itineraries or find inspiration trips that other users went on. The third option for the workspace is not complete at the time of writing. The vision of the workspace is to allow a user to add, remove, and rearrange the activities in an existing itinerary, which can then be saved to the database as a new itinerary. Finally, the last option “logout” provides an option to exit the program.

```
------ DayTripper ------
Select an option:
 - 1. Recommend a Trip
 - 2. View Itineraries
 - 3. Workspace
 - 4. Logout
1
Please answer the following questions.
--------------------------
Select a city:
Boston
Selected city: Boston
--------------------------
Describe budget on scale $ (cheap) to $$$$ (expensive):
$$$
Selected budget: $$$
--------------------------
Include dining activities? (y/n)
y
Including dining activities.
--------------------------
Select a type, enter 'done' to move on or 'sample' for ideas:
sample
('American', 43)
('Restaurant', 43)
('Italian', 27)
('Seafood', 25)
('Pizza', 18)
--------------------------
Select a type, enter 'done' to move on or 'sample' for ideas:
American
Types: ['American']
--------------------------
Select a type, enter 'done' to move on or 'sample' for ideas:
```

```
------ DayTripper ------
Select an option:
 - 1. Login
 - 2. Register
1
Username: George
Password: daytripper
Logged in as George

------ DayTripper ------
Select an option:
 - 1. Recommend a Trip
 - 2. View Itineraries
 - 3. Workspace
 - 4. Logout
```

Use Cases

The core use case for DayTripper is to assist an end user in the creation of an itinerary for a trip. Typical trip planning referencing a plethora of resources and manually keeping track of all the small details. The first option in DayTripper is inspired by exactly this problem: instead of having an end user spend hours searching for exciting activities, they can spend less than 5 minutes filling out a form to generate an itinerary that is personally relevant to them. This tool can be used by parents to plan a family trip, college students to plan a night out, tourists to see all the historical grandeur a city has to offer, or an HR manager to plan a memorable day of team-building activities.

The ability to view saved itineraries provides a foundation for the concept of creating and modifying template trips. Currently, the user can simply view their own saved itineraries and those saved by other users to the database. If the development of the app were to continue, these itineraries could be imported into the workspace so that the user could modify a template to make it their own.

Conclusions

The project was ultimately a success. DayTripper is functionally implemented as an interactive itinerary building which makes use of information stored in a MySQL relational database. A command-line interface allows users to generate a personalized itinerary based on their answers to a questionnaire, view their past itineraries, and view itineraries created by other users as templates. Users can also register and login under the same credentials later.

The four main problems encountered during development were overcome. First, data scraping directly from the web proved to be a dead end without paying for access to the Google API. An alternative method of data acquisition was discovered using Octoparse, which populated the database with real-world entries for the Boston area. Second, the gathered data needed to be cleaned line-by-line to fill in missing budget values and remove the commas from review counts. Pandas data frames were used to programmatically clean the large dataset (which at the time was spread across 5 CSV files). Third, calculating travel times between destinations had to be coded manually since directly scraping Google Maps was not a viable option. Excel Macros were used to add latitude and longitude values for every place in the database, with which the Geopy package could calculate the Euclidean distance between two locations thus allowing the estimation of travel time based on speed of transit. Fourth, supporting the storage of itineraries in the database revealed that the design needed to be revisited to avoid becoming cluttered with join tables. As is detailed in the database design section, the database was completely redesigned to simplify itinerary support, include users, and set a more scalable foundation.

Author Contributions

All authors contributed to the proposal document, regularly attended meetings for project planning, worked on developing the original conceptual model, presented the project to the class and created part of the slideshow shown during the presentation.

George Dunnery wrote the status update and final report, manually cleaned some of the data, lead group meetings by setting an agenda and defining clear goals, and programmed the core backend / frontend of the application using Python (and refactored it after the database was redesigned). Jake Feinbaum wrote and updated the logical model script, debugged the database using default values and constraints, created the feature table entries, and expanded the application to support user registration, calculating travel times, and the storage / retrieval of itineraries. George Dunnery and Jake Feinbaum also redesigned the conceptual and logical models. Vedanth Prakash came up with the idea for the project and its novel factor (keeping track of the small details), acquired all data used in the database using Octoparse, and retroactively gathered the latitude and longitude values for all venues. Bradford Egan cleaned the data using Pandas data frames to replace missing values and properly format columns to correct problems encountered when importing the data to MySQL. Bradford Egan combined the old category tables into the mono-table 'Place' after the database redesign, and generated the Tag table with over 4000 entries.

References

1. Google. *Google Maps.* Retrieved from: https://www.google.com/maps

2. Octopus Data Inc. *Octoparse.* Retrieved from https://www.octoparse.com