DayTripper

Logan Noel
Carlos Alvarado
Tom Jarosz

Project Goal

Develop a software product that builds a custom one-day itinerary of popular sites/attractions in a city

- Includes a route-optimized list of attractions based on the user's schedule, preferences and mode of transportation as well as attractions' locations throughout the city

Primary Data Sources:

~~GooglePlaces API~~, Foursquare API, GoogleMaps API, Geopy API

First user interaction

User Preferences/Getting the Data

Collect user input (city, categories, time constraints, etc.)

- Check if we have city, else, get it from Geopy
- Check if we have places for given city/category, else, query FourSquare
    1. Get list of places for a given category (for each city, about 100~300 places)
    2. (Multi)process those places
    3. Get place information and store in DB
- Get photo and description for top 10 places
- Display top 10 results to user

Second user interaction

User Ranking/Optimization Algorithm

For top 10 places, collect user input (ranking)

- Run routing algorithm
- Display itinerary results
  - Suggested times
  - Ordering
  - Google Map

Project Demo

1. City already in DB
2. City not in DB
3. “I don’t know” city
4. Revealed preferences for places
5. Multiprocessing vs. single processing

Management Tools

- Categories populator
- Data refreshing

What we've learned

- ORMs are great!
- Multiprocessing helps reducing time (even with single core!)
  - Care needed when making parallel transactions with DB
  - Debugging difficult
- API limits are huge bottleneck if deploying App to real life
- TSP approximations are not very good

DayTripper

Logan Noel
Carlos Alvarado
Tom Jarosz

User cases

Only “required” input is city

1. Sends empty form
    a. Displays error
2. Writes a city name that is not in our DB
    a. Search for that string using **geopy**
    b. If success, save city and continue; otherwise, display error
3. Selects a city that is already in our DB
4. Selects “I don’t Know”
    a. Selects random city from our DB and continue

Front-end: Added some JS

- We used some JQuery widgets to make our form more user-friendly
  - Autocomplete field, with cities loaded from our DB
  - Calendar field
  - Time frame slider

Django: URLs / Views

- Unique URL

- Three main views:
  - No user input
  - User input as POST
  - User input as GET

Django: Templates

- One main template with “child” templates associated with each view
- For this, we used Django’s Template Inheritance (http://djangobook.com/templates-in-views/)

Django: Files structure

- We tried to keep our code clean; for this reason we used Django's recommended file structure:

- Project Root
  - Home
    - Home templates
    - Django settings
  - Itinerary
    - Management
      - Load_categories
      - Refresh_db **TO BE IMPLEMENTED, NEED TO MODIFY MODELS.PLACE**
    - Itinerary app templates
  - Utilities
    - Main scripts

Places pseudo-code

Collect places(city, categories)

- Check if we have city/category combination in our DB
  - If so, return those places
- Get list of places for a given category (for each city, about 100~300 places)
- **(Multi)process** those places
  - Get place information and store in DB
- Return places

Select places(user_query)

- Places = collect_places(city,categories)
- Return top 10 places

Optimize pseudo-code

Determine how many nodes (places of interest) to include in itinerary

- Branch-and-bound algorithm determines approximate time required to traverse n nodes
- Include user-prioritized nodes first

Find distance minimizing route through selected nodes (TSP with heuristics)

- Start user as close to predefined starting location as possible
- Calculate total time required to traverse calculated nodes, considering user's mode of transportation, variable traffic conditions, variable time spent at different types of locations (dynamically programmed)
- Offer alternative mode of transit if user's selected type is particularly inefficient for a given edge (e.g., takes too long to walk from node C to D)
- Order nodes to maximise the number of nodes open (e.g., 3pm-8pm) when user arrives/leaves