Team Runtime Error Final Report

**Description**

The goal of *Where2go* is for users to plan trips in USA. Based on input cities and tags, user can easily get a recommendation for their travel route and a timeline view of the trip. Each tourist attraction comes with a brief overview as well. All of the features can be viewed on the website. User will also be able to check their previously saved plans and modify them.

**Usefulness**

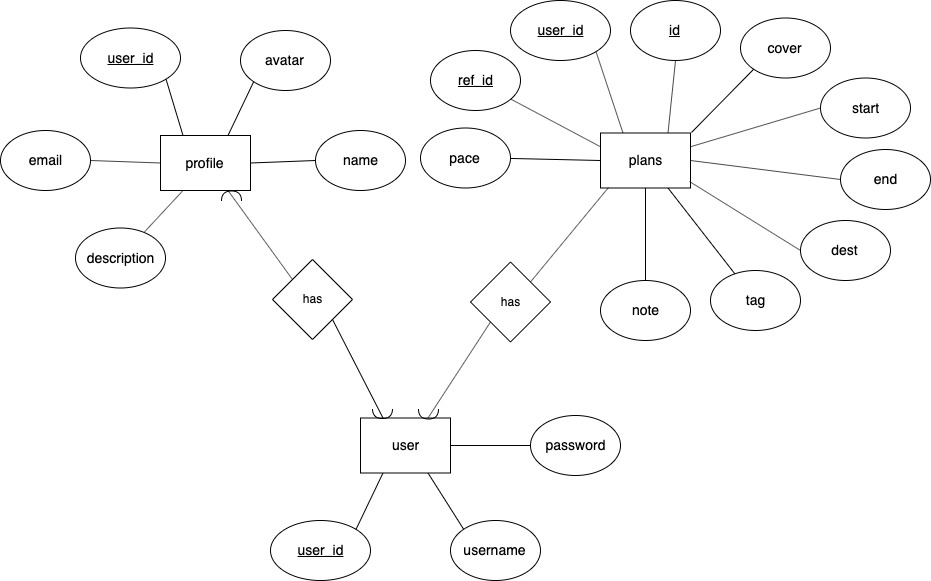
It is quite common that people spend days and weeks to make a travel plan. While planning the route, people would like to have one or more “tags” for the trip such as shopping, relaxing, or wildlife. People may also consider the pace, which can be viewed as the hours people would like to spend travelling each day. Users may also just use the recommendation to get an expectation about what the trip may look like.

Besides selected destinations and a start-end date, user will choose any number of preference tags such as shopping, beach, museum, etc. User can also describe their pace as slow-easy, medium(default), or fast-paced, and the route will change according to user preference. For now, we are just focusing on the travel recommendations in California.

**Data**

In our database, we store user’s information and their saved plans. That includes user’s avatar’s url, id, password, name, email, description. For the plans, we have start date, end date, a list of destination cities, a list of tags, cover’s url, note, pace, and reference id.

**ER Diagram and Schema:**



users(user\_id, username, password)

profile(user\_id, email, description, avatar, name)

plans(id, user\_id, start, end, dest, tag, cover, note, pace, ref\_id)

**Where we get the data**

We crawl the information of all tourist attractions in California from tripadvisor.com, including their rating and overview. Then we use data mining to estimate the duration for each spot.

Google api is used to find the longitude and latitude of each tourist attraction, so that we can calculate the great-circle distance between each pair of tourist attractions. We only focused on California because the amount of geoinformation data is too large.

**NoSQL**

We use Neo4J to store cities and tourist attractions’ information and itinerary information. Attractions are stored as nodes with relation –[:Near]-> to the city or region it is close to, while cities are stored as individual nodes. Trip plans are stored as linked list, with a head node labeled with the ID of the plan.

**Relational and non-relational**

Geographic data used in the project can be stored more efficiently as graphs (more specifically, trees). Since each itinerary contains a list of tourist attractions, it is better to store the list structure in a graph database instead of relational database. Therefore, geographic information and itinerary information are both stored in Neo4j for high efficiency queries. User profiles, login information and plan details are stored in relational database (MySQL) to utilize queries with joins among different tables. Each plan entry in the relational database has a reference id that points to the corresponding plan stored in the non-relational database.

**Feature specs:**

1. Register account: On the login page, the user can sign up with username, email, password.

2. Change account’s avatar, id, password, name, email, description: In user profile, the personal information can be updated by click “modify”.

3. Delete account: In user profile, user can click “delete” to delete the account.

4. Recommend trip plan: User can select cities and tags to create a travel route.

5. Modify plan’s start date, end date, list of cities, list of tags, cover’s url, note, pace: The algorithm will modify the route recommendation according to changes in date, cities, preference and pace.

6. Delete plan: The user can delete plan in profile.

7. View plan in timeline view: After click on the plan, the user can see a timeline view which shows the travel route day-by-day on a minute basis.

8. You can enjoy California’s scenery in the background.

**One basic function**

One of our basic function is to create /modify /delete the user data. The user data was store in two sheets (profile & user), the user sheet only stores the userid, username and password which is mostly used for login and register. The profile sheet contains the user’s email, name, avatar and description. When deleting user’s account, we need to join tables to find user’s plans, and delete them as well. To prevent ‘ghost users (who has record in user but no data in profile)’, we used transaction when create/delete user data.

**Code snippet for SQL:**

INSERT INTO profiles (user\_id, email, description, name, avatar)

VALUES ($uid, '$email', '$description', '$name', '$avatar')

ON DUPLICATE KEY UPDATE email = '$email',

name = '$name',

description = '$description',

avatar = '$avatar';"

**Code snippet for NoSQL:**

MATCH (c:City)

WHERE c.name = {city}

CREATE (p:Place), (c)<-[:Near]-(p)

SET p.name = {name}, p.duration = {duration}, p.url = {url}, p.rating = {rating}, p.tag = {tag}, p.address = {address}

RETURN p

**Dataflow**

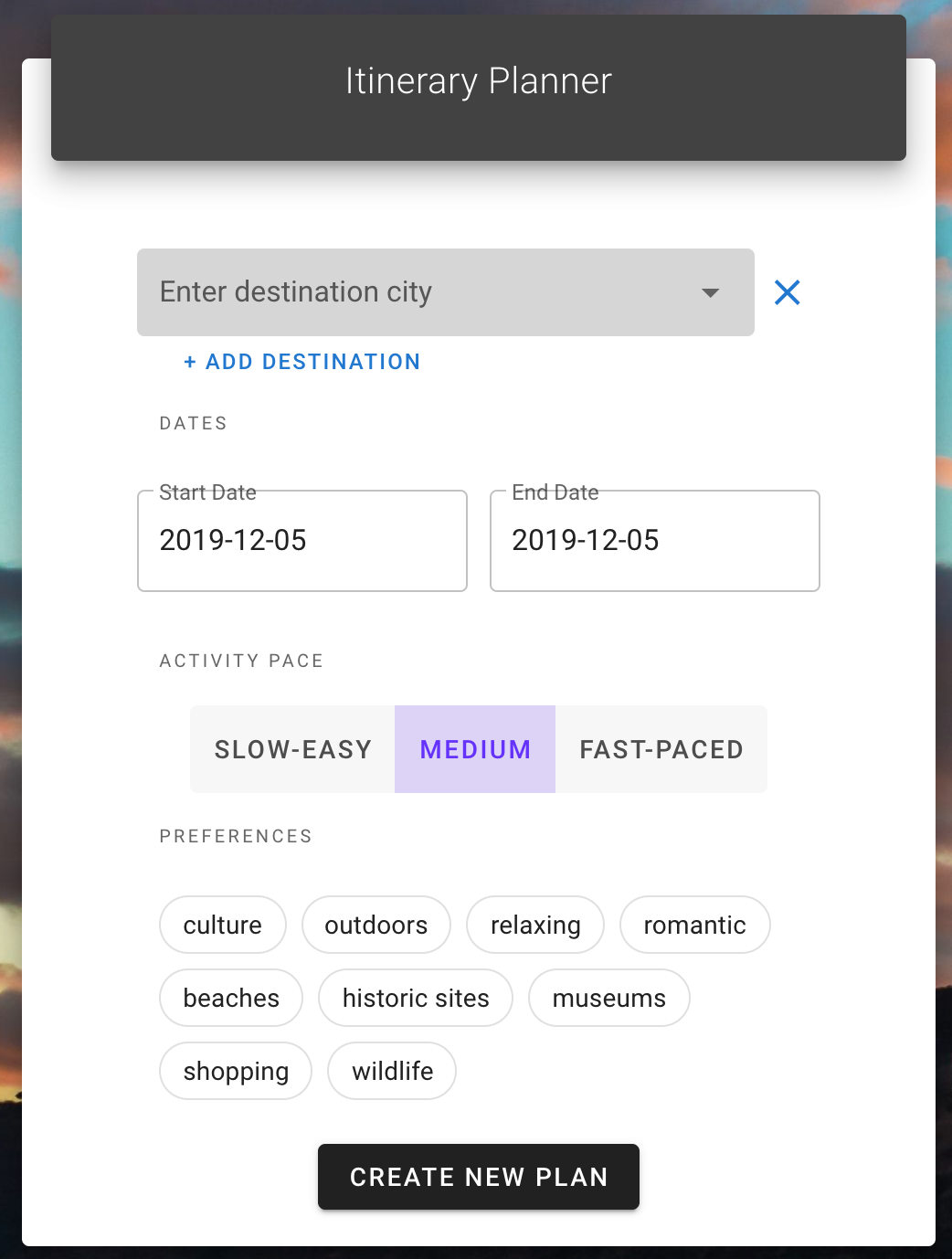
Alan

**Advanced function 1**

Our advanced function 1 (AF1) is to intelligently recommend user a travel plan based on user preferences. Users can have any number of plans stored in profile. Each plan can be viewed in a day-by-day timeline view.

To maximize tourists' experiences and travel efficiency, we referenced an *improved personal preference recommendation* (IPRR) algorithm from journal “A travel route recommendation algorithm with personal preference”. It takes user's personal preferences, user types, the real-time traffic condition of the tourism region, the historical statistical data real-time network congestion conditions, the popularity preference, and some more features into account to optimize travelling distance and travelling time.

We generate the travel route using a simpler version of such planning algorithm. Our recommendation considers user's preferences, pace, and rating of each tourist attraction, besides number of days and planned cities. Finally, we can run the algorithm to recommend travel route to user.



This problem is an advanced algorithm problem. We considered that it as a multidimensional constrained post man problem. We use the technology mentioned in the IPRR algorithms to compress the distance, preference, attraction rating into weight, while this weight is updated every time one node is reached. And we used Bellman-Ford algorithms under the limit of pace and time to generate the route.

Other things equal, our algorithm will offer a very different recommendation to the user selecting fast paced, from the user selecting slow-easy. Each tourist attraction also has one or more attributes. A dictionary is built to relate each attribute to one or more of the nine preference tags provided to the user.

**Technical challenge**

Apache lost

**What’s changed**

Originally, we thought about hotel recommendation. Later, we figured out that it might not be a good idea to include hotel information. The first reason is there is already very good and mature applications that does this job, and there is no need for us to do it again. Also, the time required for advanced function 1 is more than expected. And our server crashed once during development. It takes hours for us to resolve this problem.

**Division of labor:**

Shuhan Wang: php backend and database

Mingjie Zhao: front-end and communication between front and back end

Hanxuan Chen: algorithm and python backend

Zongqi Chen: 🚣