Applications in Object-oriented Programming and Databases:
Close By
A program to find places and get to them

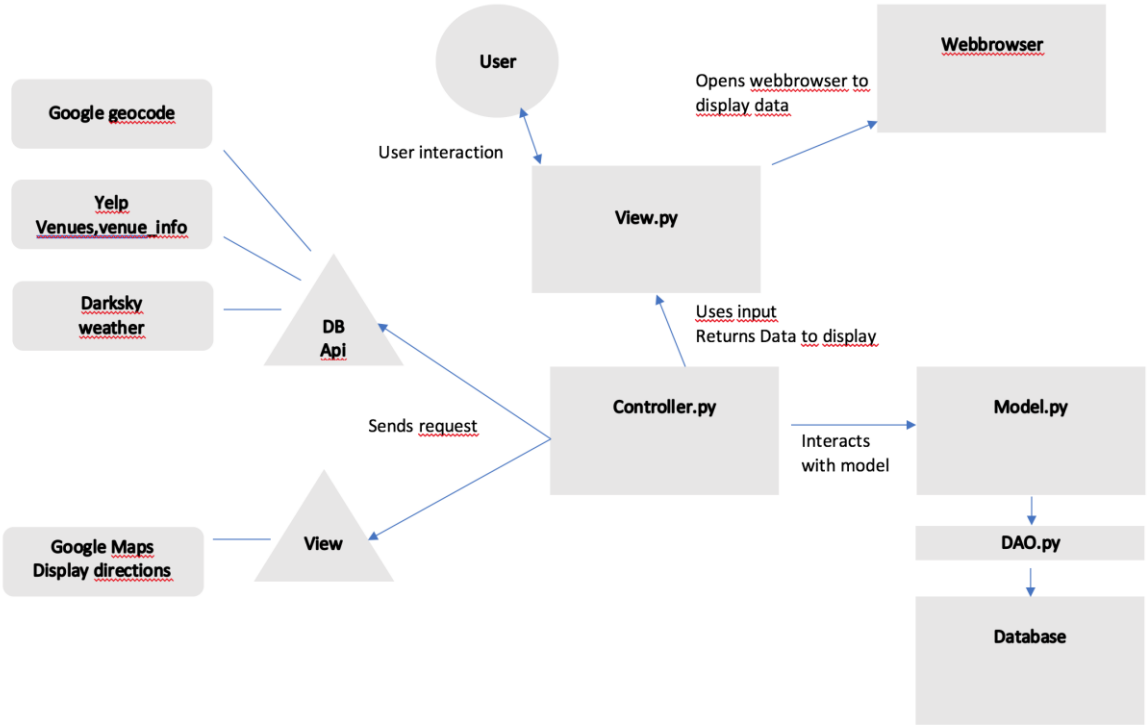
Professor:
Professor Binswanger, Johannes
johannes.binswanger@unisg.ch
&
Ruben Zürcher
ruben.zuercher@student.unisg.ch

Authors:
Emanuel Guggenheim
Emanuel.guggenheim@student.unisg.ch
Dominic Brotschi
dominic.brotschi@student.unisg.ch

Abstract

Our Project aims to create an application that allows the user to input any location in the world and get activities places and venues (restaurants,bars,clubs, etc.) corresponding to his requested location and a from the user specified category. The Data is gathered through the Yelp Fusion API. Furthermore, data about the current and future Weather Conditions and directions are added. Directions are displayed on Google Maps. Per Request the user can get a maximum 1000 activities, places and venues. In this paper we are going to discuss the structure of our application and it's underlying database. In the last part we will address our key learnings and possible improvements of our application.

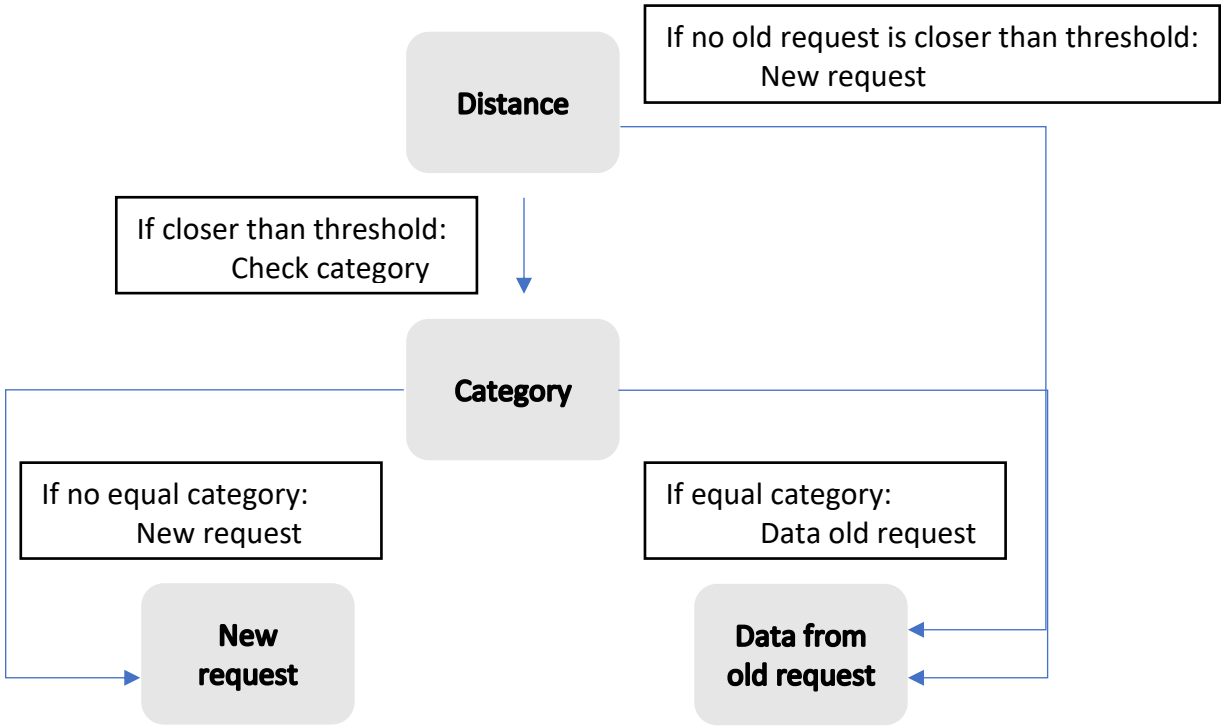
Application Overview



Structure

Our application follows the MVC pattern with a data access object between the model and the database. In a first step the user is asked in the command-line to input his current location. The view returns the raw statement into the controller who then sends an API request to Google Geocoding to get the exact coordinates. In a second step 22 pre-saved categories are retrieved from the database and displayed in the command-line. The user

now has to choose a category. The category, the longitude and latitude chosen by the user are getting saved in the database next to a request id in a table created to save all user requests made to the application. In a next step the controller has to decide whether he has to send an API request or use already stored entries. Through the model the controller retrieves all the requests that have been made. If there are no requests, it sends a new api request. If there are requests it calculates the distance between the current userlocation and all the requests that have already been made. If the distance between current userlocation and one of the old request is smaller than a certain threshold the controller checks further if the old request was made with the same category request as the new request, if yes, there is no need for the controller to send out a new Api request. The above is shown in illustration one. We implemented this functionality mainly for efficiency reasons as getting 1000 entries from yelp takes up to 30 seconds.



Ilustration one: Api request

In of both cases, the controller sends out a request for the weather data and the model then stores it in the database.

New request

The controller sends the request and receives a json object. The Controller then parses it into a list containing all the key-values that need to be saved. After parsing the controller calls the model to insert the data through the dao into the database.

Old request

If the controller detects that there is an old request saved close to the user location and with the same search category he will then return the request id of this request and pass it on to the model. The model will then get all the data gathered with this request.

In both cases the user will be asked how many entries he would like to see, if he would rather like to order the data by distance or by rating and if the data should be ordered ascending or descending. After answering these questions the view returns the input statements. The controller passes the returned statements to the model which joins the tables and selects the rows according to the user input. The controller then returns the

selected rows into the view who then parses the data into html adds a css and opens the browser to display a dataframe. The user then has to choose one entry, for this entry he will get directions displayed on google maps. He chooses the entry by entering its unique venue id into the command-line. The view returns the id and the controller passes it on to the model. The model returns the selected row to the controller who then parses the coordinates. The coordinates of the selected venue and the coordinates of the userlocation will be sent to google maps and the browser will pop up showing the directions on google maps. In a second tab of the browser the user will find the weather data for his current location. Finally, the user will be asked if he wants to make a new request or quit the application.

Database

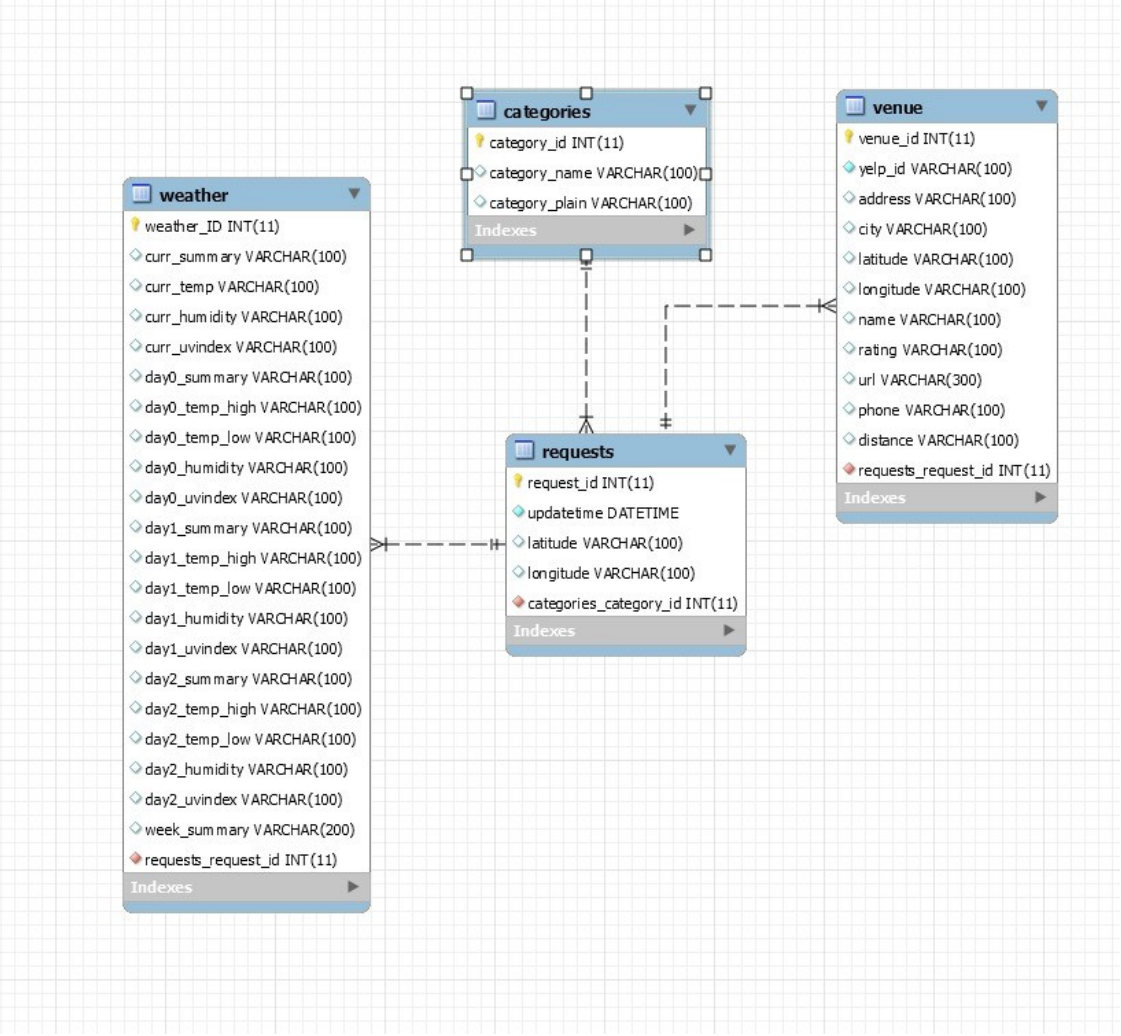


Illustration 2: Database EER

Our Database contains 4 Tables. The heart is the requests table. In this table all the user requests are getting saved. In the venue table the information received from the yelp api is saved. The categories contain all the parent categories from the yelp api and the weather table contains the weather data gathered form the darksky api.
The gathered data is linked to its corresponding requests. All the tables except categories do have a foreign key pointing to the requests table. They have a many to one relationship with the requests table. Due to the fact that our application needs to be able to retrieve data from former requests.
When initializing the application, the database is checked for requests older than 12 hours. These requests together with the corresponding data in weather and venue are then deleted as they are out of date through cascading the foreign keys. The categories table is not affected due to the fact that it doesn't contain a foreign key pointing to the request table. The request table contains a foreign key pointing to the category id. This is necessary because the application needs to check with each user request if a similar request has been made distance wise and category wise. The weather and venue table do allow NULL entries as requests are sent with coordinates and the answers of these requests sometimes do not contain all the entries. The categories table does not allow any null values because of the fact that the categories have been pre-inserted. The application is not allowed to alter the categories table.

Complications and conclusion

Firstly, due to certain circumstances, our group got reduced to 2 members. This and the fact that neither one of us has ever written a line of object oriented code, led to a variety of complications, clearly visible upon first sight of our code. Only half-way through we understood that we should set up our classes vastly differently, that many ideas we had were impossible to implement, atleast for us and that OOP is not easy after all. The learning curve, as mentioned by Mr. Zürcher in the class, has been hard-felt by us both, even more so than with R.
Initially, we had a very hard time understanding the concept of MVC when applied to our theoretical idea. However, this is definitely something that has drastically improved over the course of this project, albeit too late to be fully implemented into our code. We now realize, that our controller should be set up fundamentally different.

Most of our time was spent solving for errors and using the correct syntax.
In hindsight, we don't even know what we would have done differently. We learned an amazing amount, certainly more than for any other 2 credits project. We also believe our next attempt at such a program would already be drastically better, although still far from proficient.
A program like Close By allows for so many added functionalities, converging towards our initial goal of basically an assortment of APIs, including redundancies such as multiple venue providers (Google Places, Foursquare). Although we were close to such an implementation, we had to realize that if we were to continue with this starting point, we could've never delivered a working product due to time constraints. Furthermore, we fully prioritized the controller and model compared to the view, as we both couldn't manage integrating a UI without losing further functionality of the main program.
While we enjoyed most of the creation of this project, we came to the realization that we had spent an immense amount of time for the amount of credits awarded and while we would gladly do it again, we do hope that our complications will be considered.
We have basically tried to speak Spanish, without having ever learned it, but do now understand the basics.

Appendix

Prerequisites

- Python 3
- MySQL
- Packages:
    - Pymysql
    - Requests
    - Webbrowser
    - Math
    - Pandas
    - Os
    - Urllib.request

In the dao.py and category.py files, the MySQL login info must be changed accordingly.

After having set up the DB from the dumps, you can run the main.py and start the program.