TRIP PLANNER APP

Team Members:
Bochao Wang
Ly Ly Dang
Gurjas Singh
Yesenia Canales

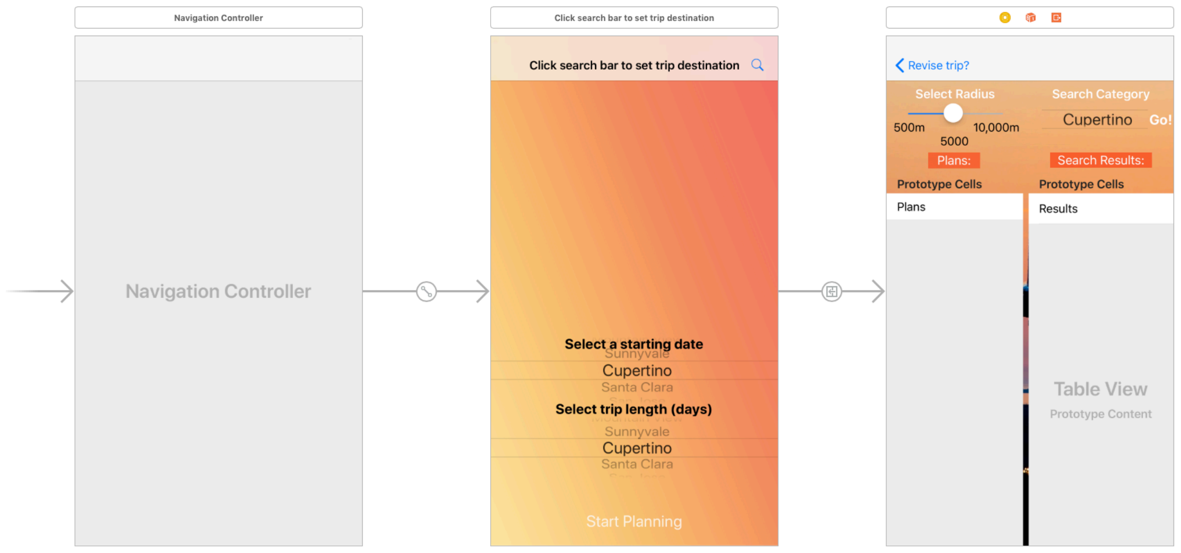
Contents

I. Purpose of the Trip Planner

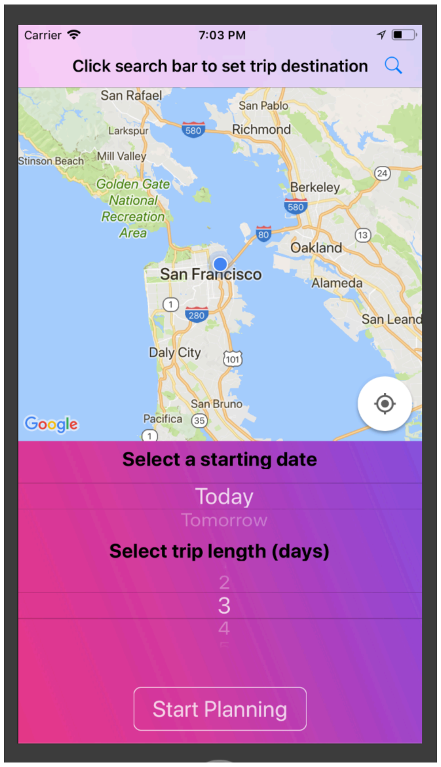
There are a number of trip planning apps in the App Store, but they are all either a traveler's guide or To-Do styled reminder app (like our list app covered class), or an itineraries tool for collecting air tickets, hotel booking.

What we want to do is to create a flexible and handy trip planner with a fully integrated weather forecast that covers your entire trip, a quick search with a preferred destination, the ability to change the radius and the categories to focus on the places worth the visit, along with an easy drag-and-drop feature to create and modify the order of your planned visits.

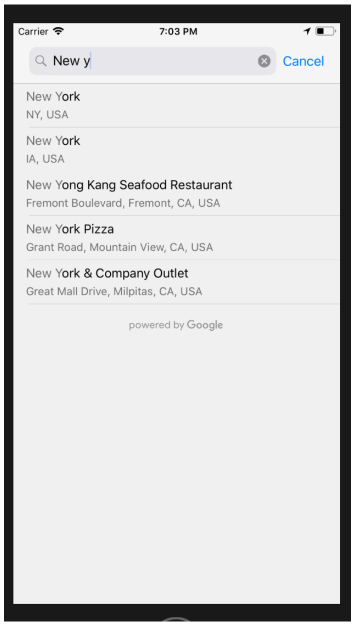
II. Navigation of the app

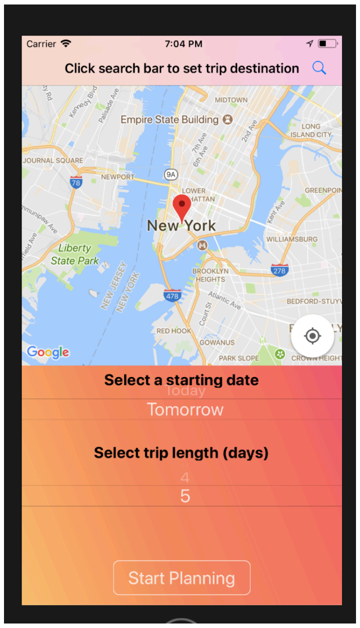


The screen shot above is the main storyboard of our app. The navigation of the app is simple and the navigation controller helps navigate back and forth between the main view (controlled by ViewController.swift) and the search view (controlled by SearchViewController.swift).



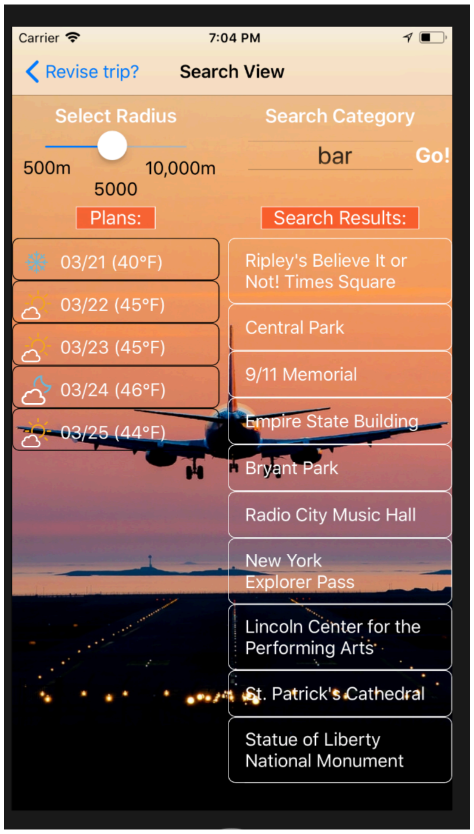
When the app launches, the user will be asked to allow location usage, then the user's current location will appear on the screen (blue dot). San Francisco is set to be the user's default location if a preferred destination isn't searched for.

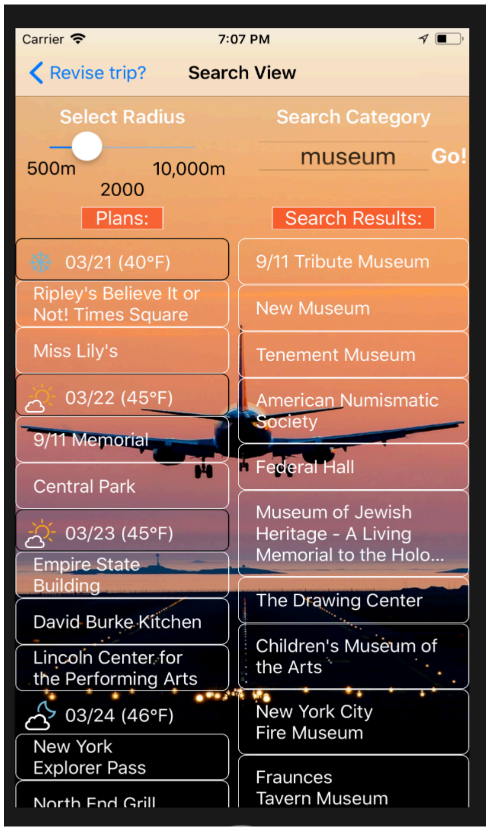

Carrier
7:03 PM
New y
Cancel
New York
NY, USA
New York
IA, USA
New Yong Kang Seafood Restaurant
Fremont Boulevard, Fremont, CA, USA
New York Pizza
Grant Road, Mountain View, CA, USA
New York & Company Outlet
Great Mall Drive, Milpitas, CA, USA
powered by Google



Of course, we provide an easy searching function via Google Maps and the Places API to help user find the destination of the trip. You simply tap on the search icon to start the search view which allows you to find a place by its coordinates, city, country, or a keyword. Here for example, I searched for “New York” and once user selects the destination, the main view will show the new destination with a red Place Picker within updated GMSMapView.

Then user can scroll through the two UIPickers to select a starting date and the trip length (days). After completion, the user may click “Start Planning” to begin navigating the Search View.


Carrier 7:04 PM
Revise trip? Search View
Select Radius Search Category
bar Go!
500m 10,000m
5000
Plans: Search Results:
03/21 (40°F)
03/22 (45°F)
03/23 (45°F)
03/24 (46°F)
03/25 (44°F)
Ripley's Believe It or Not! Times Square
Central Park
9/11 Memorial
Empire State Building
Bryant Park
Radio City Music Hall
New York Explorer Pass
Lincoln Center for the Performing Arts
St. Patrick's Cathedral
Statue of Liberty National Monument


Carrier 7:07 PM
Revise trip? Search View
Select Radius Search Category
museum Go!
500m 10,000m
2000
Plans: Search Results:
03/21 (40°F)
Ripley's Believe It or Not! Times Square
Miss Lily's
03/22 (45°F)
9/11 Memorial
Central Park
03/23 (45°F)
Empire State Building
David Burke Kitchen
Lincoln Center for the Performing Arts
03/24 (46°F)
New York Explorer Pass
North End Grill
9/11 Tribute Museum
New Museum
Tenement Museum
American Numismatic Society
Federal Hall
Museum of Jewish Heritage - A Living Memorial to the Holo...
The Drawing Center
Children's Museum of the Arts
New York City Fire Museum
Fraunces Tavern Museum

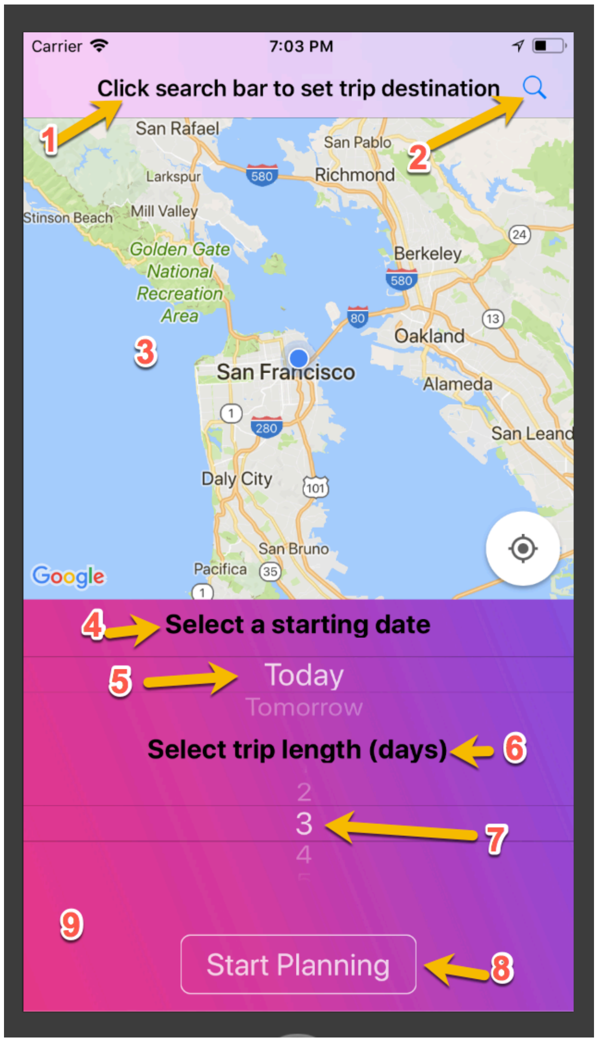
The search view comes with the city’s weather forecast results on the left side and the “things-to-do” search results on the right. The weather forecast data is retrieved from the Weather API, which will be further discussed, covering all of the 5 days selected.

The “things-to-do” results are based on the user’s selected destination and are ranked by prominence within a 10km radius via Google API. User may select another radius by moving the UISlider and another category from UIPickerView such as a museum, restaurant, bar, etc. to retrieve new results.

The user may use the convenient useful drag-and-drop feature to grab a result cell on the right and drop it into the preferred section on the left side to add to your trip plan. If the user changes their mind about a certain place in their trip plan, they may simply swipe left to delete the place. The user can scroll through the two independent tables to create a customized trip plan with ease.

III. Customization of Controls

A. Main ViewController



The main view is controlled by the "ViewController.swift", customization of the 9 controls are shown here.

1. UINavigationItem: Write sentences for the title and back button to prompt user to search for a trip destination.

2. UIBarButtonItem: Select the search icon to display it. Then write func “openSearchBar” to invoke Google AutoComplete API on the new AutoCompleteView, which lets the user pick a place from the search results. Afterwards, the location manager will update the location and the UIPicker in the GMSMapView (No. 3).

3. GMSMapView: Set the name to be “googleMapView”, set defaults (delegate, camera, coordinates, etc.), then enable “current location” to be used in the app and view current location as blue dot. Function “locationManager” will update and show the map with the returned result from No. 2 control with predefined zoom, camera, then stop updating.

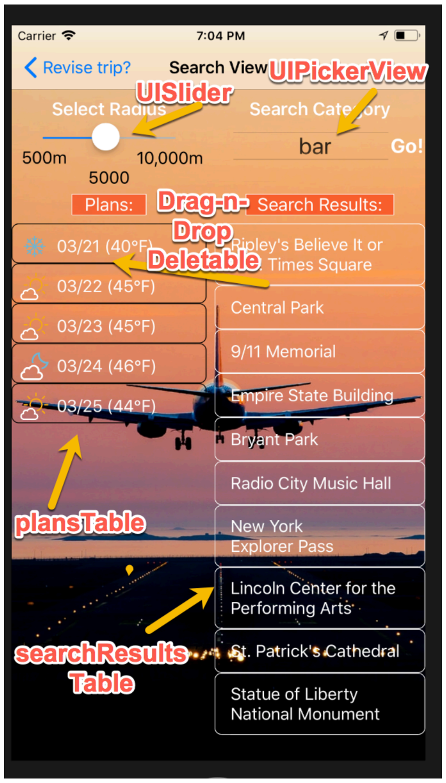
4. and 6. UILabel: Write a sentence, didn’t customize.

5. and 7. UIPickerView: called “datePickerView” and “lengthPickerView” respectively. Defined an array for each view, e.g. let startDate = ["Today", "Tomorrow"] to show all choices for user to pick from. For each pickerView, set title to be the corresponding row, set numOfRows to be the array count, save the choice to “begin” and “days” by using tags (tag 1 for datePickerView, tag 2 for lengthPickerView), and set the foregroundcolor (text color) of both pickerView to white.

8. UIButton: named as “navigationButton”, customize its border color, width and corner radius for better visual effect. Function “prepare” links the data from main view via “ShowSegue” segue to the search view, including the coordinates of the destination, starting date and length of the trip.

9. gradientView and animatedBackgroundColor(): To set the background of the app to change color, we actually add a gradient image called gradientView and then call the function animatedBackgroundColor() to animate the gradientView to run from left to right and reverse.

B. SearchViewController



The SearchViewController of the application contains two UITableViews corresponding to the plansTable and the searchResultsTable. To add these tables to the view, the controller was extended to include the UITableViewDataSource and UITableViewDelegate classes.

Data is passed by prepare function and “ShowSegue”. Both Weather API and Google Place API will execute the search query and return the results to corresponding table and display in tableView.

UISlider lets user to choose a radius from 500m to 10km with indicator on current radius, we just customize the slider scale to increase the radius by 500m. UIPickerView works in the same way as former customized UIPickerViews to let user choose a category from the list and pass the selection to search query. Here we just use 8 categories for demonstration, actually Google Places API provides dozens of Places type to be chose from. https://developers.google.com/places/android-api/supported_types

We set up an URL link with keywords and embedded radius & category choice, set up the session and send via Google places API. The data returned in JSON format, we use one powerful tool called SwiftyJSON to easily fetch the name from each object and store in searchResults array. DispatchQueue with asyncAfter predefined seconds is adopted to make sure all results are returned before reload the data.

In the UITableViewDataSource, customizations were made to the following functions: numberOfRowsInSection, numberOfSections, cellForRowAt, canEditRowAt, and commitEditingStyle. Each of these functions will return different values depending on which UITableView is passed in. For the plansTable, the number of sections is the length of the trip in days. The number rows in a section is the number of activities that is planned for that day. For the searchResultsTable, the number of sections is always one and the number of rows is fixed to 10 to only display the first 10 results from the search query. The cellForRowAt function is customized to return the cell information for the appropriate cell identifier and also has customization for the cell UI. For the plansTable, the canEditRowAt and commitEditingStyle functions were customized to allow deletion of a row when the user swipes left.

In the UITableViewDelegate, customizations were made to the following functions: viewForHeaderInSection and heightForHeaderInSection. These functions are customized to modify the header of the section for the plansTable. The section customization does not apply to the searchResultsTable.

In the viewDidLoad function of the SearchViewController, the delegate and datasource properties need to be set for both the plansTable and the searchResultsTable.

C. Weather API

In the app, the forecasted weather data comes from the weather API of the Darksky.net website. The API requires users to have an account in which a key is given for that account. The key allows a number data fetches per account per day.

To use the API, we only need to pass the location in longitude and latitude to the website to request forecasted data. The API will return the current weather forecast for the next week. A weather structure was implemented to request the forecasted data as well as parsing the data returned from the API. The weather structure includes a weather summary, a weather icon and the temperature data.

When the SearchViewController is loaded, the getWeatherForLocation function is called with the location specified by the picked location which returns an array of weather objects for the next week for that location. This data is stored in the forecastData variable that will be used later to populate the “Plans” table of the Search View. Note that code was added to remove the extra forecasted days that are not needed by the user in planning their trip.

The plansTable is a UITableView within the SearchViewController. The number of sections for the plansTable is customized by the length of the trip. On initialization, the number of rows for the plansTable is zero. This changes as the user inputs trip activities for that day. The section header was customized to show the date, forecasted temperature along with a weather icon.

D. Drag and Drop Feature

In the SearchViewController, there are two tables corresponding to the “Plans” (plansTable) and “Search Results” (searchResultsTable). The plansTable is described in the “Weather API” section. The searchResultsTable is a UITableView which contains results from the search of surrounding places.

The purpose of the drag and drop feature of the app is to drag the results from the search that the user wants to include in their plan to the “Plans” section for that day. To implement this, the SearchViewController class was extended to include the following classes: UITableViewDropDelegate and UITableViewDragDelegate.

For the UITableViewDragDelegate, the itemsForBeginning and the itemsForAddingTo functions were customized to locate the item at the selected index for dragging. The return type for these two functions is a UIDragItem. The selected item for dragging is a value from the seachResults array that eventually gets set as the UIDragItem.localObject. This localObject will be used in the UITableViewDropDelegate.

For the UITableViewDropDelegate, the performDropWith function was customized to properly perform the drop functionality. Upon a “drop”, the destination index is determined first. This is the corresponding row and section indices for where the user wants to drop the dragged object. Next, the loadObjects function is called to get the “localObject” that was set in the UITableViewDragDelegate. After the object is loaded, an insert of the object into the plansTable is invoked.

In the viewDidLoad function of the SearchViewController, the enable flags for the drag and drop delegates need to also be set for use. The searchResultsTable needs the dragInteractionEnabled property to be set to true while the plansTable needs the dropInteractionEnabled property to be set to true.

IV. Data Persistence Mechanism and Schema

We use Google Maps and Google Places API to search and display the chosen destination and places of interest according to radius and category. The API key is written to AppDelegate to make sure it'd working properly. The things-to-do results are ranked by prominence within 10km radius, so the results are stable for a relative long time. JSON reader is written to retrieve the results in given order.

Arrays for selection and variables have default value to avoid null, and arrays of search results are predefined to be empty and updated after each search, which would reduce the possibility of app crash to enhance robustness.

Async and wait for several seconds is adopted to make sure all search results are returned in given time to avoid data corrupt or bad user experience.