Daily Planner

Members: Will Zhang, Pierre Boerkoel, Estella Wang

Daily Planner helps with indecision

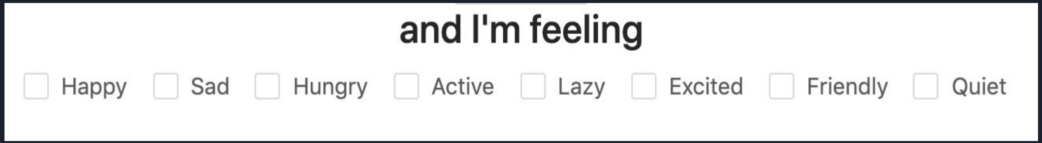
Purpose

Users input their current 'feelings', availability, and the maximum distance they are willing to travel, then we return a recommended 'plan' to that user


DEMO TIME

Achieved Goals

- Frontend to enter feelings *(minimal goal)*

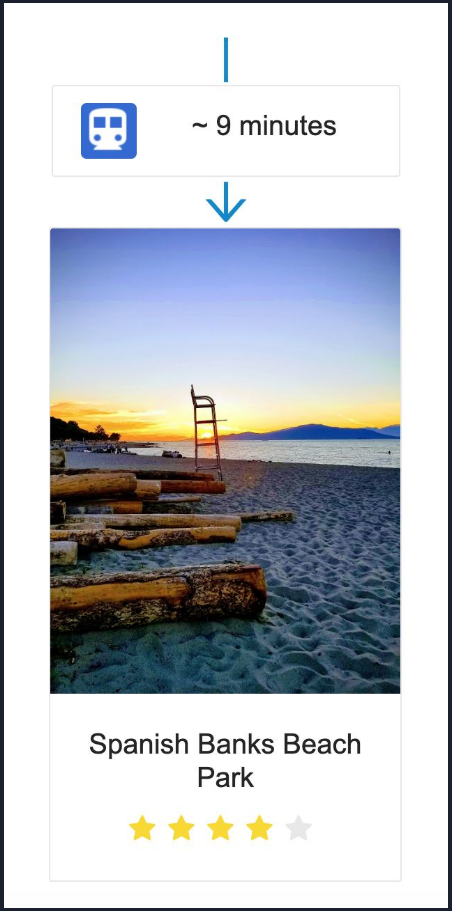


- Association of these feelings to tags in order to lookup places using Google Places API *(minimal goal)*

```
const feelingsDictionary = {
    "Happy": [
        "bar",
        "restaurant",
        "book_store"
    ],
    "Sad": [
        "park",
        "cafe"
    ],
    "Hungry": [
        "restaurant",
        "cafe",
        "supermarket"
    ],
    "Active": [
        "park"
    ],
    "Lazy": [
        "restaurant",
        "spa"
    ],
    "Excited": [
        "bar"
    ],
    "Friendly": [
        "bar",
        "park"
    ],
    "Quiet": [
        "book_store",
        "library"
    ]
};
```

Achieved Goals

- Creation of some recommended activities for the user *(minimal goal)*
- Ability to take into account the distance from one place to another *(standard goal)*
- Ability to create a plan that takes into account the time a user is available *(standard goal)*
- Mobile-friendly UI design *(stretch goal)*
- Ability to create a commute plan between places *(standard goal - partially achieved)*



Unachieved Goals

- Allow a user to download the plan to their calendar
- Use machine learning to better understand users' preferences

Issue - Millions of Google API calls if scaling up

Places Photo	Up to 28,000 calls	$7.00	$5.60
Places - Nearby Search + Basic Data + Contact Data + Atmosphere Data **Total cost:**	Up to 5,000 calls	$32.00 $0.00 $3.00 $5.00 ------------- $40.00	$25.60 $0.00 $2.40 $4.00 ------------- $32.00
Places - Text Search + Basic Data + Contact Data + Atmosphere Data **Total cost:**	Up to 5,000 calls	$32.00 $0.00 $3.00 $5.00 ------------- $40.00	$25.60 $0.00 $2.40 $4.00 ------------- $32.00

Solution - store the data in our database

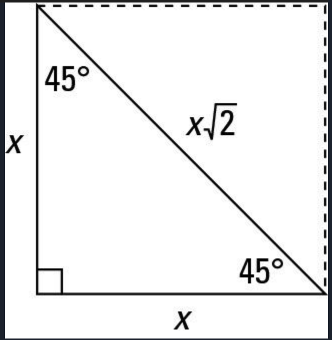
```
function updateAreaCollection(areaCollectionName) {
    console.log("Updating Area Collection: " + areaCollectionName + " at " + Date.now());
    // update database manager to reflect the update
    dbManagerCollection.remove({ areaCollectionName: areaCollectionName }, err => {
        if (err !== null) {
            console.error(err);
        }
    });
    dbManagerCollection.insert({ areaCollectionName: areaCollectionName, lastUpdateTimestamp: Date.now() });
    // create area collection or clear its previous data
    if (areaDatabase[areaCollectionName] === null || areaDatabase[areaCollectionName] === undefined) {
        console.log("Area collection doesn't exist before --- Creating new area collection: " + areaCollectionName);
        areaDatabase[areaCollectionName] = new Mongo.Collection(areaCollectionName, { _driver: db });
    } else {
        areaDatabase[areaCollectionName].remove({});
    }
    // fetch places data from google at the center of this area
    let centerGeoPoint = areaCollectionNameToCenterGeoPoint(areaCollectionName);
    let places = fetchPlacesFromGoogle(centerGeoPoint);
    for (let place of places) {
        areaDatabase[areaCollectionName].insert(place);
    }
    return places;
}
```

Super-cool feature

- Our ability to calculate the distance from one location to another
- Our estimated commute time

```
function estimateCommuteTime(place, geoPoint, commute) {
    // return an estimated commute time between two geo points (using the given commute type)
    let distance = measure(geoPoint.lat, geoPoint.lng, place.geometry.location.lat, place.geometry.location.lng);
    // estimateLongestPath = perimeter of the isosceles right angle - distance
    let estimateLongestPath = Math.sqrt(2) * distance;
    return estimateLongestPath / avgCommuteSpeed[commute];
}
```

```
function measure(lat1, lng1, lat2, lng2) {
    // Ref: https://en.wikipedia.org/wiki/Haversine_formula
    var R = 6378.137; // Radius of earth in KM
    var dLat = lat2 * Math.PI / 180 - lat1 * Math.PI / 180;
    var dlng = lng2 * Math.PI / 180 - lng1 * Math.PI / 180;
    var a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
        Math.cos(lat1 * Math.PI / 180) * Math.cos(lat2 * Math.PI / 180) *
        Math.sin(dlng / 2) * Math.sin(dlng / 2);
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    var d = R * c;
    return d * 1000; // return the distance in meters
}
```



Isosceles triangle

What technologies did we use?

HTML CSS JS

React connect() Redux

HEROKU

node JS

METEOR

mongoDB