# Put It In Park


Picture from: http://geology.isu.edu

Miles Chandler, Joseph Engelhart, Ella Robertson, Levi Villareal, and Billy Vo

The University of Texas at Austin

10/04/2019
CS373 – Software Engineering
Glenn Downing

# Table of Contents

# 1 Motivation

This project aims to encourage Americans and tourists to go outdoors by highlighting National Parks and recreational activities in their area. This would serve to promote physical fitness and increase awareness of all that national parks have to offer. In doing this, we hope to answer three questions:

1.) What national parks are in close proximity to our users?
2.) What recreational areas are in close proximity to our users?
3.) What types of sites can visitors expect to see in each area and what activities can they participate in?

# 2 User Stories

Our developers are Food Cravings, and their website is https://foodcravings.net.

## 2.1 Phase 1

### 2.1.1 User Stories

#### 2.1.1.1 User Story 1

Include a description within the splash or "Home" page. User story completed; altered the splash page to match specifications.

#### 2.1.1.2 User Story 2

Include profile pictures, commits, and issues of each team member within the "About" page. User story completed; about page data was dynamically pulled from GitLab.

#### 2.1.1.3 User Story 3

Structure our models in a grid or table form for each of their respective pages. User story completed; structured models in a grid format.

#### 2.1.1.4 User Story 4

Include a link to our API documentation designed by Postman. Our API should be capable of returning a list of models, returning a list of models, returning attributes of the instances, and returning detailed list of models. User story completed; added link to API documentation on About page.

#### 2.1.1.5 User Story 5

Design our instance pages as stated. Three instance pages for each of the three models, five attributes from the model page, and one instance-specific media. This was completed by creating the three instance pages, and filling them with static data.

### 2.1.2 Requests to Developer

#### 2.1.2.1 Customer Story 1

As a user, I would like your website to include at least the project/website name in place of "Logo" at the top left corner. This is so I can know clearly what website I'm on and the purpose of the project. The font/styling can stay the same, I would just like the content to clearly reflect the website for now.

#### 2.1.2.2 Customer Story 2

As a user, I would like to see data on three instances for Recipes, Restaurants, and Grocery Items on their respective pages. This is so I could see what kind of instances would be included in each model, and get a preview of the types of information that will be shared in the final website. The data can be displayed in any visual format for this step; I just want it to appear on the page.

#### 2.1.2.3 Customer Story 3

As a customer, I would like the About Page to have stats dynamically derived from GitLab. This is so I can see how productive the developers I've hired are being. The statistics I would like derived from GitLab are

total number of commits, total number of issues, and number of unit tests (probably 0 tests so far, but I would still like it to appear on the page).

#### 2.1.2.4 Customer Story 4

As a customer, I would like the home page to display something other than lyrics. Please change the title of the home page to the title of the project, and the body of the web page to give a brief description of the project. This will help the customer to see what goal of the final version of the website is.

#### 2.1.2.5 Customer Story 5

As a customer, I would like the website to have a color theme other than black and white. This is to make the website more aesthetically pleasing to users. In order to begin this shift in color theme, could the nav bar color be changed from black to something else? Any other color will do - choose one that you feel will best fit with the feel of your final website.

## 2.2 Phase 2

### 2.2.1 User Stories

#### 2.2.1.1 User Story 6

Add pagination - It would be very helpful to be able to page the instances grid when clicking on a model, allowing users to see only a portion of the full list of instances for each model. User story completed. We used react-paginate in order to allow users to page through data on each model page.

#### 2.2.1.2 User Story 7

About page improvement - include what tools you are using, motivations for the website, and link to your GitLab repo on About page. Issue resolved; We restyled the pages and made the fetching of commits and closed issues work correctly, as well as adding the purpose, and links to the GitLab repository and postman documentation.

#### 2.2.1.3 User Story 8

Build and consume RESTful API. API must be expanded and configured to:

1. List of instances of a model with their 5 basic attributes
2. List of instances of a model
3. Detailed attributes of an instance Your API must pull the data from YOUR database hosted on AWS or GCP. Ensure your website is pulling the data from the API dynamically, and no data presented on the site for a model/instance is static.

User story completed. We created an API that allows the consumer to get a list of all parks, recreational areas and states. It does so in a way where it gets 10 items at a time, in order to provide pagination. In addition, the consumer of the API can also get information by passing in a park ID, a recreational area ID, or a state ID, which allows the user to get data specific to a single instance.

#### 2.2.1.4 User Story 9

Highlight table row for model pages - When cursor hovers over an instance in model page highlight the table row. Also, make the entire table row clickable instead of just the name of the instance. Implement this functionality for all instances of every model page. Issue resolved; we altered the table to a grid format for all model pages (parks, states and recreation) and added an animation where the grid item pops up when you hover over it. In addition, we also styled these grids to provide a much more aesthetically appealing layout.

#### 2.2.1.5 User Story 10

Edit model pages - It would look nice if the instance pages were presented in a format other than a columnized list. Maybe some more color or a change in layout would look good, and could even help with loading the data dynamically to have set containers. A few changes could make these pages really pop, and would make the website that much more appealing. Levi changed the model pages from tables to grid, and updated the styles of the model and instance pages to make them much more appealing.

### 2.2.2 Requests to Developer

#### 2.2.2.1 Customer Story 6

Calorie rounding - As a user, I would like to have the calorie attribute altered a bit. Could you please round the calorie attribute to the nearest calorie? This is for ease of use and UI clarity.

#### 2.2.2.2 Customer Story 7

Pagination - As a user, I would like to have pagination implemented for each model's page. Once there are enough instances to implement pagination, please do so. The number of instances per page can range anywhere from 10 - 50, whatever you feel works best for your site.

#### 2.2.2.3 Customer Story 8

More instances - As a user, I would like to see each model page have a good deal of information. Please have at least 10 instances for each model. If for some reason you feel as though a model does not need 10 or more instances, we can discuss it and change the benchmark.

#### 2.2.2.4 Customer Story 9

Splash page pictures - As a user, I feel as though the transition between the pictures of food is a bit too long. It seems to take about 5 seconds, but I think 3 or 4 seconds would keep the website from feeling sluggish. If this change is implemented and you do not like the faster transition time, we can discuss it and potentially reverse it.

#### 2.2.2.5 Customer Story 10

Price range - As a user, I would like the "Price Range" attribute in the restaurants model to be more specific. Somewhere on this page, could you clarify what each dollar sign means? Ex: typical price for a single meal for "$" would be <$10, "$$" would be $10-20, "$$$" would be $20-30, etc. You can make the benchmarks whatever you like, but I feel like making this information available to the user somewhere on the page would be useful in allowing them to determine where to eat.

## 2.3 Phase 3

### 2.3.1 User Stories

#### 2.3.1.1 User Story 11

For this phase, we need searching across the website, as well as per model. Requirements: Ability to search from the splash page, should search the entire website. Each model page must have the ability to search within it, the returned search must highlight the terms searched for, and searchability must be google like.
Time estimated to complete: 3 days
Time taken to complete: 3 days
We used Fuse to facilitate global searching as well as searching per model. We get the user entered string and use that to find the closest matches. We then display the closest matches to the user on the site.

#### 2.3.1.2 User Story 12

We would like to see what tools were used on the about me page. A picture of each tool and a description about how each tool was used to build the website. The tools have to be shown in grid-like format and special focus on the tools that were not required.
Time estimated to complete: 2 days
Time taken to complete: 3 hours
We added a new ToolCard component which we used to easily build out all of our tools and display them uniformly on the site.

#### 2.3.1.3 User Story 13

You should be able to sort the parks by data they present. You can break this by multiple parameters including: price, name, activities they offer, etc. This feature should be on each section of the website, and should be done on the same page just reordering the presentation of data.
Time estimated to complete: 3 days
Time taken to complete: 1.5 days
Because we implemented our API using Flask Restless, this part was made very easy. Using the built in order_by query parameter, we could grab the user selected sorting options, and pass that to the API using order_by.

#### 2.3.1.4 User Story 14

As the website stands right now, the connections between one model to the next lead to a 404 error. This should be fixed and reimplemented to allow for cross-connections between one model page to one or more model pages of different types.
Time estimated to complete: 1 day

Time taken to complete: 11 hours
These errors showed up because we did not consume the API in the previous page. Once we began consuming the API, the errors went away.

#### 2.3.1.5 User Story 15

Currently, all the models have duplicate instances. It would be nice to have more instances without duplicates.
Time estimated to complete: 1 day
Time taken to complete: 5 hours
Similar to the last story, these errors showed up because we did not consume the API in the previous page. Once we began consuming the API, the errors went away.

### 2.3.2 Requests to Developer

#### 2.3.2.1 Customer Story 11

Searching capability - As a user, I would like to easily search for the information I'm looking for. Please add searching capabilities to each model page, as well as on the splash page in order to search the website as a whole. Please place the search bar somewhere near the top of each web page.

#### 2.3.2.2 Customer Story 12

Sorting capability - As a user, I would like to easily sort the instances on each model page. Please add sorting capabilities that involve the instance attributes. It can be sorted by thing such as name, calorie count, etc.

#### 2.3.2.3 Customer Story 13

Create a logo - As a user, I think it's appealing for a website to have a logo to distinguish it. Could you please create a simple logo that relates to your content in some way? This could be displayed in the nav bar.

#### 2.3.2.4 Customer Story 14

Color scheme - As a user, I would like to see a more aesthetically pleasing color scheme. Right now, it seems to be just black, white, and gray. Could you think of another color scheme that adds more interest to the website?

#### 2.3.2.5 Customer Story 15

Fonts - As a customer, the fonts used for the nav bar and headings seem pretty standard. Would it be possible to find a new font that would add more interest to the user? Perhaps Google Fonts or https://femmebot.github.io/google-type/ could assist in this, although it is not a requirement to use these resources.

# 3 RESTful API

APIs currently utilized:

- National Park Service (NPS) API - Used to pull information about national parks.
- GitLab API - Used to pull members' profile pictures, amount of commits, and amount of issues.
- Recreation Information Database (RIDB) API - Used to pull information about recreational activities.
- U.S Census – 2018 Population API - Used to pull information about population for all U.S States
- Bing – Bing Image Search API – Used to find photos for recreational areas.

Our API (PINP API):
Postman documentation: https://documenter.getpostman.com/view/9011044/SVtR19mz
Base URL: https://flask-backend-dot-potent-retina-254722.appspot.com/api

- Requests of models return a list of all parks/recreation/states depending on the URL specified in the request

- Requests of instances return specified park/recreation/state given respective id's as a parameter.
- Requests of models specifying a park/recreation/state id return a list of instances of that model that offer or exist in that park/recreation/state.

How to construct a query using Flask Restless https://flask-restless.readthedocs.io/en/stable/searchformat.html

- This allows for easy sorting and filtering

# 4 Models

## 4.1 National Parks

The model "National Parks" represents the 49 parks throughout the contiguous United States. This model connects to the "States" model as location plays a large part in park visitation, and it connects to the "Recreational Activities" model as National Parks often include outdoor recreation.

### 4.1.1 Examples of filtered/sorted attributes

1.) Park Code
2.) Park Name
3.) Location
4.) Number of related recreational areas (if any)
5.) Fees
6.) Annual visitors

### 4.1.2 Examples of searchable attributes

1.) Park Code
2.) Park Name
3.) Location
4.) Related Recreational Area IDs
5.) Description of Park
6.) Park-ID
7.) Weather Info

## 4.2 Recreational Activities

The model "Recreational Activities" represents the estimated 500+ recreational activity opportunities throughout the United States. This model connects to the "States" model as different recreational activities are available based upon location, and it connects to the "National Parks" model as National Parks often include outdoor recreation.

### 4.2.1 Examples of filtered/sorted attributes

1.) Recreational area ID
2.) Reservable
3.) Number of activities
4.) Parent org ID
5.) Stay limit

#### 4.2.2 Examples of searchable attributes

1.) Recreational area name
2.) Location
3.) Related National Park (if any)
4.) Activities
5.) Description

### 4.3 Locations

The model "Locations" represents the states within the United States, plus Washington D.C. This model connects to the "Recreational Activities" model as different recreational activities are available based upon location, and it connects to the "National Parks" model as National Parks are visited heavily based upon their location and are tied to the ecosystem of the land they exist on.

#### 4.3.1 Examples of filtered/sorted attributes

1.) FIPS code
2.) State name
3.) Mail code
4.) Number of national parks (if any)
5.) Number of rec activities (if any)
6.) Population

#### 4.3.2 Examples of searchable attributes

1.) FIPS code
2.) State name
3.) Mail code
4.) Names of national Parks (if any)
5.) Recreational area IDs

## 5 Tools

- React JS - JavaScript library, namely using react routing
- Node - JavaScript run-time environment that executes JavaScript code outside of a browser
- Express - Web application framework for Node.js, designed for building web applications and APIs
- GCP - Used to deploy and host the web application
- Bootstrap - free and open-source CSS framework directed at responsive, mobile-first front-end web development
- Postman - Used to document our API and output in HTML format.
- Gitlab - Used to host our code and provide CI/CD environments
- Pixabay - Used to find royalty-free images for the static site
- NameCheap - used to provide website URLs
- Mocha – Used as our JavaScript test framework
- Enzyme – Used to more easily test our React components
- Selenium – Used to automate acceptance tests
- Chrome Webdriver – a controllable webdriver for Selenium

- SCSS – Used to make styles more flexible and readable
- Docker – Used to provide images for frontend and backend development
- Cloud SQL Proxy – Allowed to connect to the database locally using GCP
- Flask – Lightweight python web framework, used for our backend API
- MySQL – Relational database management system used to store our data
- Fuse.js – Used as a fuzzy search to find the closest match to each

# 6 Hosting

URL: putitinpark.xyz (and putitinpark.me)

Website URLs were obtained using NameCheap, and the site itself is hosted using Google App Engine. Details to deploy the app using app engine can be found here <https://cloud.google.com/appengine/docs/standard/nodejs/building-app/deploying-web-service>, and are also listed below.

- Clone the repository onto your local machine.
- Navigate to the cloned repository directory.
- Install the gcloud command-line tool.
- Run 'gcloud init' in the repository directory, then select the appropriate project from the list provided (You must be invited to the correct gcloud project).
- When ready to deploy, run 'gcloud app deploy'.

The site is able to support HTTPS because Google Cloud Platform provides a free SSL certificate to sites hosted using its platform. In order to get this working, we had to add DNS records to our NameCheap account, to provide proof that we had access to the website.

# 7 API

Note: currently, our base api's url is https://flask-backend-dot-potent-retina-254722.appspot.com/api instead of putitinpark.xyz/api.

## 7.1 National Parks

GET putitinpark.xyz/api/nationalparks

Returns all instances of National Parks

GET putitinpark.xyz/api/nationalparks?page=<page_number>&results_per_page=<results_per_page>

Returns instances of National Parks on page <page_number> with specified results per page

GET putitinpark.xyz/api/nationalparks?q=<JSON_query>

Returns instances of National Parks of which satisfies the JSON query object

## 7.2 Recreational Activities

GET putitinpark.xyz/api/recreations

Returns all instances of recreational activities

GET putitinpark.xyz/api/recreations?page=<page_number>&results_per_page=<results_per_page>

Returns instances of recreational activities on page <page_number> with specified results per page

GET putitinpark.xyz/api/recreations?q=<JSON_query>

Returns instances of National Parks of which satisfies the JSON query object

### 7.3 Locations (States)

GET putitinpark.xyz/api/location

Returns all instances of locations (US states and D.C.)

GET putitinpark.xyz/api/location?page=<page_number>&results_per_page=<results_per_page>

Returns instances of locations on page <page_number> with specified results per page

GET putitinpark.xyz/api/location?q=<JSON_query>

Returns instances of locations of which satisfies the JSON query object

## 8 Pagination

### 8.1 Frontend Support

On the frontend, we used an external library component, react-paginate, in order to facilitate pagination. This component provided a basic HTML pagination element, which we styled to make more accessible and match the theme of our website. With this component, we were also able to define a handlePageClick method, which handled page redirects whenever a pagination element was clicked.

### 8.2 Backend Support

On the backend, the API supported calls which took a number that corresponded to a pagination number. This would then return an object containing 12 instances, so that on the frontend, we would only display 12 instances at a time, which was the goal of the paginator.

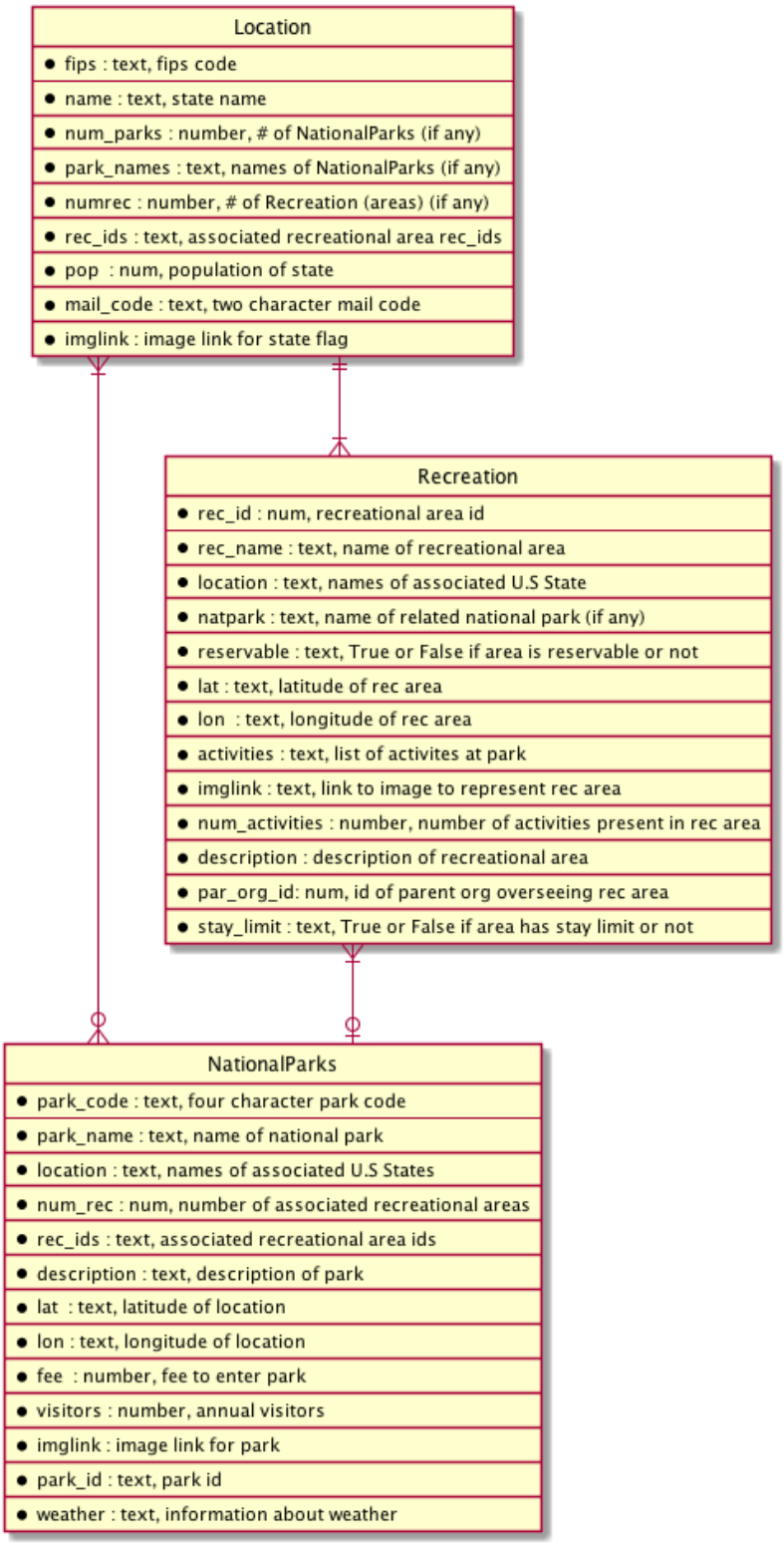
## 9 Database

### 9.1 Data Sources

Data was pulled from external APIS including the National Parks API (https://www.nps.gov/subjects/developer/api-documentation.htm), Recreation Information Database API (https://ridb.recreation.gov/docs), Bing Image Search API (https://azure.microsoft.com/en-us/services/cognitive-services/bing-image-search-api/), and the US Census API (https://www.census.gov/data/developers/data-sets/acs-1year.html). These APIS were called using scripts that can be found within misc/scripts, and information retrieved from them is stored within a MySQL instance on GCP.

### 9.2 Schema

There are three tables, each based on a model: location, nationalparks, and recreation. The schema for each table and the relationship between the tables can best be summarized by the following UML chart:


Location
fips : text, fips code
name : text, state name
num_parks : number, # of NationalParks (if any)
park_names : text, names of NationalParks (if any)
numrec : number, # of Recreation (areas) (if any)
rec_ids : text, associated recreational area rec_ids
pop : num, population of state
mail_code : text, two character mail code
imglink : image link for state flag
Recreation
rec_id : num, recreational area id
rec_name : text, name of recreational area
location : text, names of associated U.S State
natpark : text, name of related national park (if any)
reservable : text, True or False if area is reservable or not
lat : text, latitude of rec area
lon : text, longitude of rec area
activities : text, list of activites at park
imglink : text, link to image to represent rec area
num_activities : number, number of activities present in rec area
description : description of recreational area
par_org_id: num, id of parent org overseeing rec area
stay_limit : text, True or False if area has stay limit or not
NationalParks
park_code : text, four character park code
park_name : text, name of national park
location : text, names of associated U.S States
num_rec : num, number of associated recreational areas
rec_ids : text, associated recreational area ids
description : text, description of park
lat : text, latitude of location
lon : text, longitude of location
fee : number, fee to enter park
visitors : number, annual visitors
imglink : image link for park
park_id : text, park id
weather : text, information about weather

# 10 Testing

## 10.1 Frontend Unit Testing

On the frontend, we used Mocha and Enzyme in order to facilitating the JavaScript which we wrote for the frontend. Currently, the tests test basic functionality of the project, such as rendering different components, and ensuring that the correct number of components are displayed on the models page. These tests were integrated into the CI/CD pipeline, so that whenever someone tried to make a new commit on the repository, GitLab would send an email if the new code did not pass the tests.

## 10.2 Frontend GUI Testing

In order to do acceptance testing and ensure that our code worked from a real world scenario, we wrote GUI tests using Selenium. The tests spin up Chrome Webdriver and perform browser actions, and then validate those actions to see that the correct behavior was produced.

## 10.3 Backend Testing

Our backend uses Flask to serve up data from the MySQL database to the frontend. In order to test this functionality, we used Python unittest to ensure that the data that we were passing onto the frontend was what we expected.

## 10.4 Postman Testing

Postman tests check the status number of the response, as well as if the json responses from the API endpoints contain the necessary and specific properties (attributes/fields) based on the model. Newman runs Postman tests located in the collection in the CI.

# 11 Filtering

## 11.1 Tools Used

In Phase 3, we implemented our API using Flask Restless, which has a lot of built in functionality to support filtering. We built out a UI with dropdowns so that the user could select as many filters as they wanted to. Then, when the user applies the filters, the preferences are grabbed, and set along as query parameters to the Flask Restless backend, which will only return results that match all of the given filters. With Flask Restless, we can check any column value, and use the operations >, <, <=, >=, ==.

## 11.2 Filterable Attributes

### 11.2.1 Locations

- Number of recreational areas
- Number of parks
- Population

### 11.2.2 National Parks

- Number of recreational areas
- Admission price
- Number of annual visitors

#### 11.2.3 Recreation

- Reservable (T or F)
- Stay limit (T or F)
- Number of activities

# 12 Sorting

## 12.1 Tools Used

Sorting was very similar to filtering, in the fact that we took advantage of the tools already provided by Flask Restless in order to facilitate sorting. We built out a UI that allows the user to choose which attribute to sort by, and whether they want it in ascending or descending order. Once they apply the sorting, their choice (if any), is passed along as a URL parameter to the API, and the result is displayed to the user.

## 12.2 Sortable Attributes

#### 12.2.1 Locations

- Location name
- Number of parks
- Number of recreational areas
- Population
- Mail code

#### 12.2.2 National Parks

- Park name
- Location name
- Number of recreational areas
- Fee price
- Number of visitors

#### 12.2.3 Recreation

- Recreation area name
- Recreation area ID
- Location name
- National park name
- Number of activities

# 13 Searching

## 13.1 Tools Used

To facilitate searching, we used Fuse.js, which provides a ‘fuzzy search` that rates each instance on how much they match the search query. This allows for there to be some ambiguity and misspellings in the user query, and Fuse.js will still find matches. For the global site-wide search, we do the same things as on the model page, except that we run the Fuse.js query on all of the models, and display the results from all of the models.