RecRangers
Final Project Report

MIMS 2017

Ankur Kumar, Proxima DasMohapatra, Rob Kuvinka, Reema Naqvi | Faculty Advisor: Robert J. Glushko

Table of Contents

EXECUTIVE SUMMARY 3

I. **INTRODUCTION** 3

II. **PRODUCT DESIGN** 3

III. **USER RESEARCH** 4

IV. **DATABASE DESIGN** 4

V. **SEARCH** 4

PRODUCT DESIGN 5

I. **INTRODUCTION** 5

II. **DESIGN PROBLEM** 5

III. **DESIGN DESCRIPTION** 5

VI. **TOOLS AND TECHNOLOGY** 7

1. Framework - Django 7

2. Styling - Materialize.css 8

3. Wireframes - Sketch 8

USER RESEARCH 9

I. INTRODUCTION 9

II. USER INTERVIEWS 9

III. PARTICIPANTS 9

IV. INTERVIEW DESIGN 10

V. FINDINGS 10

1. Location 10

2. Two Phases of the Search Process 11

3. Tags versus Paragraphs 11

4. *Comparison Tool is Key* 13

5. *Photographs* 13

6. *Weather* 13

VI. USABILITY TEST 13

DATABASE DESIGN – A MODEL FOR INFORMATION CLASSIFICATION 16

I. INTRODUCTION 16

II. DATA DICTIONARY – A TAXONOMIC APPROACH 16

III. ENTITY-RELATIONSHIP DIAGRAM 18

IV. RIDB 18

V. DATABASE VS. API INTEGRATION 19

VI. DATABASE DESIGN – USING DJANGO MODELS AND SQLITE 20

VII. CHALLENGES 22

SEARCH 24

I. INTRODUCTION 24

II. METHOD 24

III. CHALLENGES 27

PRODUCT DESIGN APPENDIX 29

TECHNOLOGY GLOSSARY 31

Executive Summary

I. Introduction

RecRangers is a capstone project for the Master of Information Management & System degree at the UC Berkeley School of Information. The project is a collaboration between team members Proxima DasMohapatra, Ankur Kumar, Rob Kuvinka and Reema Naqvi. Work was performed during the Spring 2017 semester.

RecRangers is developed to accomplish two goals. First, to create an application which allows users of both novice and expert knowledge a place to discover new recreation options on federal lands. Secondly, using the Recreational Information Database (RIDB) as the data source, we want to create a roadmap and showcase how open government data can be put to use. The four essential steps needed to complete these goals are product design, user research, database design and search.

The following reports details the four steps. The product design section outlines the design problem and details how the RecRangers' UI/UX addresses this. The user research section describes what we learnt from users' struggles and how it informed our approach to the problem. The database section describes the how and why of the content being used, stored and accessed through the application, and what design decisions were taken to improve the overall user experience. The search section describes our approach towards making the data source more accessible to users, and our design choices in organizing it for a faster search process.

II. Product Design

RecRangers is designed to address two different problems. First, the current offerings for federal land discovery (recreation.gov and google search) overwhelm users with non-helpful information, which often leads to cognitive overload. Using lessons from behavioral economics and choice architecture, RecRangers aims to tackle this problem by making design decisions which leave users more satisfied which their choices. This is largely accomplished by limiting search result sets, displaying only pertinent information (as identified by user research) and a search-select-compare workflow.

RecRangers goal is to sketch a roadmap on how open government data sets and APIs can be put to use. The main data source of data is the Recreational Information Database, which is an open data source maintained by several federal agencies. Using the API, RecRangers aims to build a product which harnesses the power of governmental data to fill a gap in the marketplace. Ideally, this process can be applied to many different domains in the future.

III. User Research

The user research was carried out with the goal of identifying what kind of information users need to make a decision about visiting a public land. User interviews were conducted in two rounds, an initial round to identify some areas of concern and secondary interviews that sought to answer the issues raised. We found that no other website in this space offers what people consider a good user experience, searching for recreation sites is a complicated, drawn-out process and identified those design elements in competitor sites that were helpful. All this informed our product design. Usability testing our website in the end highlighted the areas for improvement in our design solution.

IV. Database Design

In order to create an application which would be aimed at solving the pain-points of the users, it is always important to have a robust data-management system. The main goal of the database design in this project was to create a content layer which would enable the creation of features and improve the usability of the complete application.

It was very important that this layer readily avail data depending upon the need of the user, and as requested by the application. This report describes, in detail, what and how the design decisions taken allowed us to successfully cater to this need.

V. Search

User research prior to the development of the web application highlighted some common frustrations with the usability of the data in the RIDB website, which was affecting the usability of the vast amount of information present on the website recreation.gov. The goal of search in the RecRangers website was to aid users in their search process for outdoor activities on federal lands by introducing a platform that shares many features with the most popular search applications (Google, Bing, etc.) such as suggest-as-you-type, smart results that show relevant results according to a user's location, suggestions in case the user's search query did not return any results, etc.

Product Design

Rob Kuvinka

I. Introduction

RecRangers aims to provide users seeking to discover new federal recreation areas a way to search, learn about and compare. Market and user research found current offerings such as recreation.gov and google overwhelm users with both novice and expert domain knowledge with too many options, non-helpful information and insufficient comparison tools. RecRangers tackles this by creating an interactive application using data from the Recreation Information Database API to facilitate discovery and decision making. Fundamental requirements include activity/location search, a limited search result candidate list from search, ability to discard unwanted options and a comparison tool. A few of the major design decisions where a search-filter-compare workflow, duel-field Elasticsearch and cards stylized with materialize.css to battle information overload. Design evaluation was verified by building a working version of the application and doing task-based user research to determine if the requirements had been satisfied.

II. Design Problem

Federal lands are some of the premier recreation areas in the country. Unfortunately, visiting new locations, particularly when in unfamiliar locations, can be intimidating. Challenges include discovering new areas, crowds concern and finding activity information about them. All these problems that can dissuade people from visiting these places. RecRangers' main mission is to address this burden.

In a different context, many technologies and systems used by governmental agencies are outdated. RecRangers is also focused testing feasibility and creating a rough roadmap on if/how these legacy systems might be updated to modern industry standards. This requires not only technology implementations, but people based methodologies like user research. Thinking through how to use agile software development to address needs for a broad range of users (which is typical of a governmental project such as this) was a big piece of this project.

III. Design Description

The high-level structure of RecRangers is a three-page flow; search → filter → compare. User research found this basic 3-part flow to have significant advantages over recreation.gov in keeping users on task and quickly accomplishing discovery and selection goals.

The search page (Fig. 1) is designed under several requirements. We the page must allow search by location/activity (or both), clearly articulate the purpose of the application (the purpose of the application) and introduce a styling theme that will remain consistent throughout the UX. Search was implemented by using a dual search

box, breaking up the location and activity.1 This marks the major departure from the current search on recreation.gov where only one search box is given. The intention of this design change is that by entering both the location and activity, users will be more specific in their search criteria, yielding in higher quality results. However, if a piece of information is missing (either location or activity), the user is still able to get results. Displaying a brief mission statement and contextualizing the application with some text gives the users a nudge to begin searching. Using Django as our framework, along with materialize.css, we are able to create a consistent styling across the application.

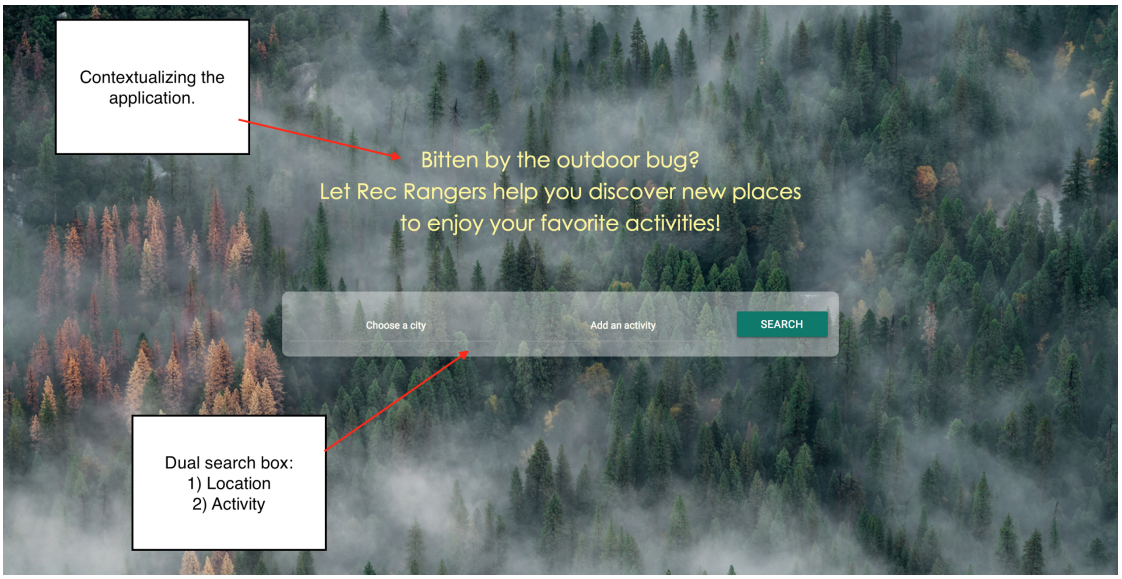


Fig. 1: Annotated screenshot of the search page

The crux of RecRangers' purpose is the results page (Fig. 2). Design decision for this page came largely from academic studies in choice architecture. Choice architecture is about the impact of presentation on decision making. There is a common misconception that providing users with all choices and information will allow them to make the best decisions. However, studies in cognitive overload have shown this is not optimal, because after a certain number of options, users feel less motivation to make a decision, and also feel less satisfaction when a decision is reached. On the other hand, the single option aversion is a bias in which users presented with only one option experience an increased desire to search for alternates.

Requirements for the results page are minimizing cognitive overload (while considering the single option aversion), displaying the only most relevant information components for facilitating choice (as identified from user studies) and the allowing the users to create a candidate set of favorite areas which will later be used in the compare page. Cognitive overload issues were tackled by only displaying three results, each on separate cards, at any given time. Information components we

[1] Inspiration for the dual search box came from kayak.com and tripadvisor.com

included on our results page are name of the rec area, a photograph, activity icons, textual descriptions (tooltip). Buttons were placed to 1) add the area to the candidate set and 2) discard for another option. Additionally, a color-coded geo-located marker of each rec area is displayed on dynamic map. If no location is entered, the default location become location of the IP address. An annotated screenshot of the results page below:

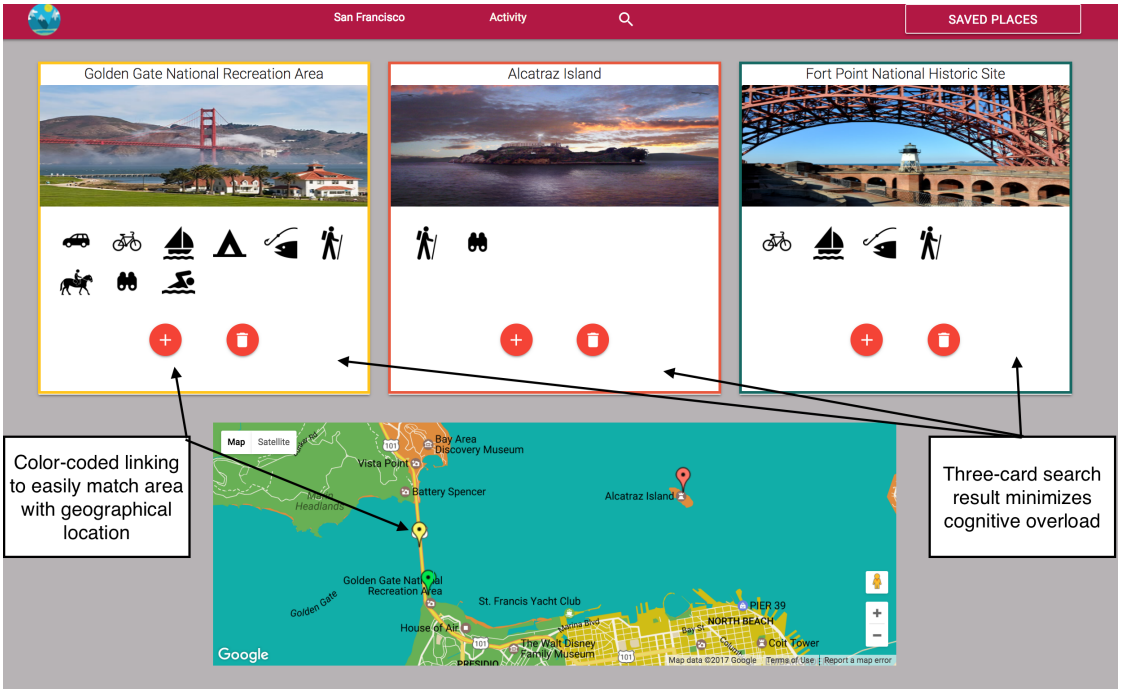


Fig. 2: Annotated screenshot of results page

After building a candidate list, the comparison page is designed to provide further rec areas details and allow the user to compare and contrast different information components. Additional components not on the results page include address, link to recreation.gov site, reservation spark-line, permitted activities and weather.

Importantly, the comparison page also allows the user to take personalized notes and has a print feature to let the user take the rec area details on the go!

VI. Tools and Technology

A. Framework - Django

We chose to use Django for several reasons. Its free, open-source and based in Python (which all group members have experience in). Django provides a robust set of built-in programs which easy front-end connection to other parts of the project such as databases and search.

B. Styling - Materialize.css

Materialize is responsive CSS framework that is built on Google's material design scheme. Its purpose is to speed up the development process by providing some default styling and being easy to work with.

C. Wireframes - Sketch

Because time is limited, wireframes allowed us to do initial user testing without need to code up the design. Sketch allows you to make a flexible workflow quickly. See Appendix for sketch/hand-drawn wireframes.

User Research

Reema Naqvi

I. Introduction

The goal of our project was to redesign recreation.gov and display the information contained in it such that it supported users in finding recreation sites within given geographic parameters. Such a site also needed to support whatever activity the user wanted to engage in (e.g. wildlife viewing near San Francisco).

In order to arrive at a design that makes picking a recreation site an easy, intuitive and straightforward process, we conducted user research to understand what the most critical pieces of information users need are. What elements crowd a web page without adding any value? What were other websites that users like using for this purpose? These are some of the questions that helped us in carving out our solution. Ultimately, we performed usability testing with users to see what elements on RecRangers could be improved and what features they hoped to see in future iterations.

II. User Interviews

The first stage of our user research focused on open-ended generative interviews with each one of our 4 team members to identify some of the important issues and questions to study. We discovered that planning an excursion to federal lands was a complicated task, even for the most experienced of outdoor recreation researchers among us. Although our team was comprised of people from different parts of the world with varying levels of outdoor survival skills, experience and familiarity with the natural landscape, not one of us could point to any one website that enabled users to perform this task well.

Once we had some basic findings, we then drilled down further into those questions and areas of concern and conducted 5 more interviews to understand what the process of planning a trip to a national park looked like for those users.

III. Participants

Even though our project idea was conceived with the intention of lowering the barriers to entry for people who were largely new or inexperienced in planning outdoor trips, we purposely also interviewed people who did not fit this criterion i.e. those who frequently explored and felt comfortable in navigating public lands. The reason for doing so was to learn from the tried-and-true trip planning method developed by them, and figuring out how this could be adapted to suit the needs of newcomers in the space. We also wanted to see if there were challenges both

categories of people faced, and the workarounds the experienced ones had come up with.

IV. Interview Design

The interviews were very open-ended and conversation like. Users were tasked with deciding upon a public land to visit during an upcoming break. The only constraint was that it had to be a completely new place for them i.e. they had not visited it before and it had not been recommended to them by somebody else. They were encouraged to browse any websites they wanted to help make the decision. Other than this direction, we allowed the users to walk us through their thought process and observed them browse different websites to find what they needed to accomplish the task. They were requested to state out loud what bits of information they were trying to/hoping to find and what they liked/disliked about the websites they used for this purpose.

The interviews were loosely structured as seen below:

1. *Person A*, imagine you have a week off from work/school coming up. Think of an activity you feel like doing (e.g. skiing, camping) and plan a trip to a public land to do it at.
2. Consult as many websites as you need.

   After users make their pick:

   a. Why did you pick place *x* over *y* or *z*?
   b. What do you think you still don't know or are unsure about?
   c. If you called the ranger tomorrow, what more questions would you have for them?
   d. Would knowing about weather/traffic etc. be helpful?
   e. Do you think a printable booklet with all the relevant information about this location would be useful?

V. Findings

1. Location

The most useful findings for our purposes was discovering that while people struggled to think of an activity (e.g. rock climbing, snorkeling) that they wanted to do, everybody had certain geographic parameters in mind. All our interviewees started the interview with something along the lines of "I've been thinking of visiting Illinois/South Carolina". It appeared that even though they had not started doing any research around this geographic area, knowing roughly *where* they wanted to go was something none of our interviewees spent much time thinking

about. For inexperienced outdoor goers, thinking of an activity proved to be challenging i.e. they just want to go somewhere but were not sure exactly what they wanted to do there.

For our design, this meant that location could be a good starting point i.e. first the user inputs where they want to go, and all other constraints such as dates and activity may be asked but are not required input fields.

2. *Two Phases of the Search Process*

The search process is a long process that users engage in over several days. Rarely is the decision to go explore a new recreation area made on the spot. Given this, the search process can be divided into two stages: the initial shortlisting phase, and the advanced analysis phase.

In the first stage of initial shortlisting, users query a geographic area for recreation sites and end up with a large number of options, e.g. 20 places that they can visit. The first thing they do is try to quickly cut down this number to get a smaller, more manageable subset of options. This process is based on first impressions and on whatever information users can digest at a glance. The time spent per option is roughly between five seconds to 30 seconds.

Users initiate the advanced analysis phase once they have narrowed down to 3-5 options. This stage involves a more detailed analysis of each recreation site. Here, users are interested in reading about other people's experiences, looking at many more photographs, checking out the types of accommodation available, looking up the weather forecast for their travel dates etc. Since they go into this level of detail for every location, narrowing down to a very small (3-5) number of options in this phase is critical.

What this meant for purposes of our design was that furnishing every possible detail about a site and displaying it on the results page was unnecessary and counterproductive. We could therefore be sparing with the amount of information we displayed as long as we provided the details at a later stage when they were needed.

3. *Tags versus Paragraphs*

People find reading through large chunks of text (See Fig. 3) to glean insights really tiresome in the initial stage of their search. However, the same level of detail is very useful when they enter the advanced stage of search. For the initial stage, since users are looking to make quick decisions and appreciate those websites that had tags (See Fig. 4) such as "dog-friendly", "camping" i.e. words that succinctly conveyed the information without sandwiching it between two

paragraphs of an aspiring writer's prose. We did not want to totally discard the rich information because without it, people could not make the optimal decision without compromise.

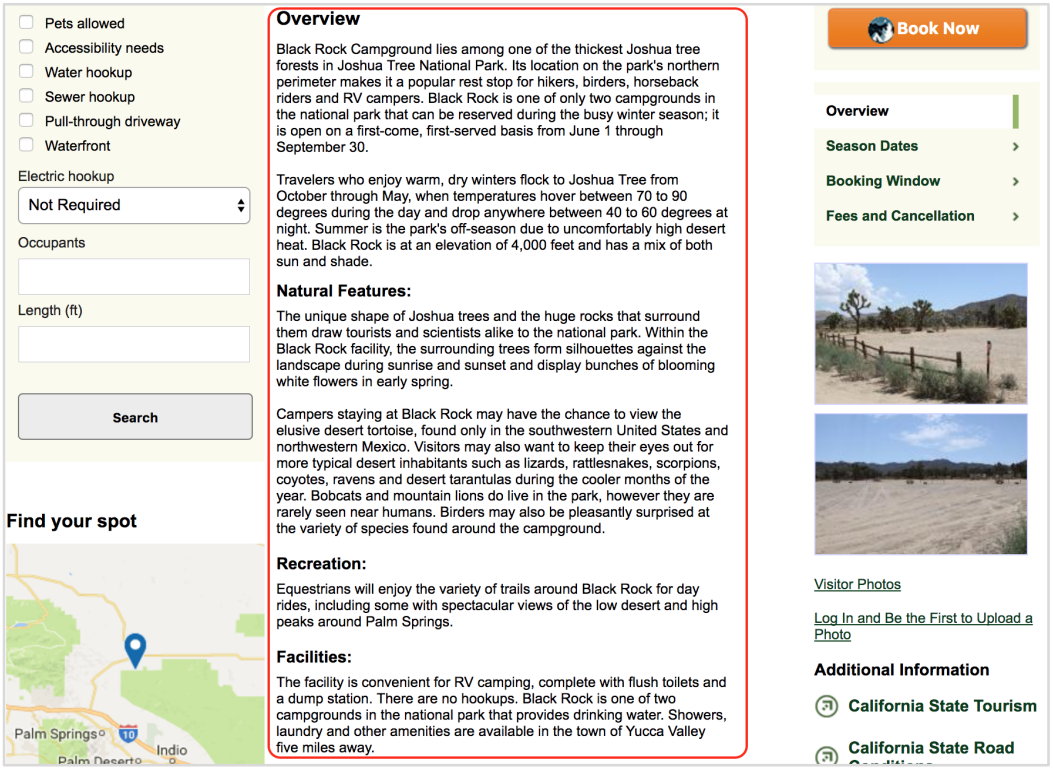


Fig. 3: Screenshot from recreation.gov showing a large amount of text

Fig. 4: Screenshot from recreation.gov showing examples of helpful tags

4. *Comparison Tool is Key*

To find recreation options, users usually enter a very generic search query in Google's search bar and open the first few links that come up. These websites range from recreation.gov to obscure blogs which people open and browse in parallel.

Reading across different sites, each with vastly differently formatted information, then trying to remember what one has just read and comparing options across different browser tabs is an extremely cumbersome and irritating process for most. However, without comparisons, users cannot make a decision. This is how we came up with the idea of a 'Comparison' page for our website to shift the burden from the user to the application.

5. *Photographs*

The selection process is heavily influenced by high quality, well-shot photographs. Blogs that do not feature photographs of the recreation sites are found to not be very helpful while an aesthetically appealing photograph can inspire the user to shortlist that site. This is why we decided that featuring photographs was not an option, but a top priority.

6. *Weather*

Knowledge of the predicted weather conditions for a recreation site for the duration of stay is a critical determinant for the choice of accommodation. If it is expected to be too cold, people choose to skip camping and book a hotel or Airbnb instead. Similarly, rainy weather means people cannot go rock climbing. While our current website does not integrate a weather API, we recognize that providing users with the weather forecast will improve their user experience and benefit their decision-making process.

VI. Usability Test

Usability tests on our website highlighted the shortcomings of different features our website plans to fix. They are as follows:

1. *Changing background pictures*

The homepage background randomly loads 1 of 6 images each time the page is refreshed. While this is aesthetically pleasing, the photographs have different color palettes while our default text's (website name, tagline, input fields) color remains consistent. This means that the text is not clearly visible on some of the images so our text's color needs to be fixed.

2. *Saved Bucket Counter*

At present, users can save a location by pressing the add button on the location card[2] to later compare sites using the comparison tool. But, there is no status message as to where that card is going or how many they have saved already. Users would like to see a counter next to the 'Saved Places' button in the header, like Amazon's shopping cart.

3. *Limit on Saved Place*
   Users recommend fixing a maximum number of cards that can be compared at a time. This is because comparing 10 would defeat the purpose of a meaningful comparison. Plus, there needs to be a message popup once that limit is reached so users know they can no longer save and compare cards.

4. *Remove button in saved places*
   Once the sites are saved and appear in the sidebar, users have no way of removing them from the sidebar without refreshing the page and losing all changes. A delete button in the sidebar will solve this issue.

5. *Long Titles*
   The title of the location cards expands into the second line if it exceeds roughly 40 characters. This changes the size of the particular card and makes it look inconsistent compared to the other two. Setting a condition where titles longer than 40 character cut off the last three characters and display an ellipsis[3] at the end instead is the solution to this.

6. *Description Tooltip*
   Hovering over the card's image displays a small description of the site in a popup to the side of the card. However, the popup disappears if the mouse is moved off the card which means users cannot scroll down the popup, or even copy paste that text if they so desire.

7. *Map is below the scrollbar*
   The map on the results page is not entirely visible without scrolling down. This is a rather huge disadvantage especially because every location dynamically adds a location marker on the map. Having to stop, move the cursor and scroll to see the marker makes the user experience not seamless. We are working on resizing all the content on the page to fit the map better.

8. *User Notes in Compare Page*

[2] A location card refers to the individual search result item of which there are 3 on the results page.

[3] Three dots to indicate omission of one or more words

A feature identified by users as potentially helpful is a text field in the compare page for every site where users can write down their own comments.

9. *Save User Session*
   Currently, the website does not accommodate creating user accounts and storing the data past their current session. If users spend an hour picking our sites, they want the ability to come back to it later and not have to start from scratch.

Database Design – A Model for Information Classification

Ankur Kumar

I. Introduction

Before we start tackling the questions around how the database was designed, including the techniques used, and the whole thought-process behind it, let's take a step back and analyze what sort of information are we talking about, including in what form, and how it is available to us. Doing so will help us explore multiple options towards what technical approach we could take, and then work towards selecting the best possible method given the constraints of technical complexity, usability, and time.

II. Data Dictionary – A Taxonomic Approach

Recreation.gov provides a gateway to allow individuals explore the American Outdoors. It functions as a one-stop shop for trip planning, information sharing, and reservations from 12 different Federal Partners.

It has a gigantic wealth of information of 2,500 Recreational Areas, containing more than 60,000 Facilities, with millions of complementary records.

In order to ensure compatibility with the information on the current website, we have maintained the same taxonomy when developing our new interface. So, let's first go through a few definitions regarding what type of information is present with Recreation.gov.

1. Organizations

The organizations represent the different federal agencies that act as partners in the Recreation.gov initiative. All the Recreational Areas, and the Facilities are associated with one of these Organizations. The data-set is the combination of all the information provided from all of the agencies.

2. Recreational Areas

Recreational areas are large tracts of federal land which are made available to the public to be used for leisure. Recreation areas have many distinguishing characteristics (attributes) ranging from addresses, organizations, links, media links, activities, etc. Recreation areas can contain one to many child facilities.

3. Facilities

Facilities are points of interest within recreations areas. They can also "stand on their own" without a parent recreation area. Facilities may also be comprised of one or more child facilities. Examples are a ranger station, hotel, campground or trail. Facilities have many distinguishing characteristics (attributes) ranging from addresses, organizations, links, media links, activities, etc. We also are provided with historical booking information for the facilities.

4. Campsites

Campsites are associated with the Facilities. The current data-set only contains the campsites which are available for reservation. First-come-first-serve campsite information has not yet been included.

We are also provided with information regarding what equipment is permitted for each of these campsites, as well as their historical booking information.

5. Permit Entrances

These are associated with Parent Facilities. Permit Entrances contain information regarding different types of permits required for different campsites, or other activities in a certain facility.

6. Tours

These represent the tours that are available for the public. These are associated to parent facilities, and can also be a part of the parent tour packages, as denoted by 'member tours'.

7. Activities

Activities are associated to recreation areas and facilities. Examples are climbing, boating, and camping.

8. Events

Events are associated to either recreation areas or facilities. Examples are Salmon Festivals, Halloween in the Park, Balloon Festivals, Environmental Fairs, and Fishing tournaments.

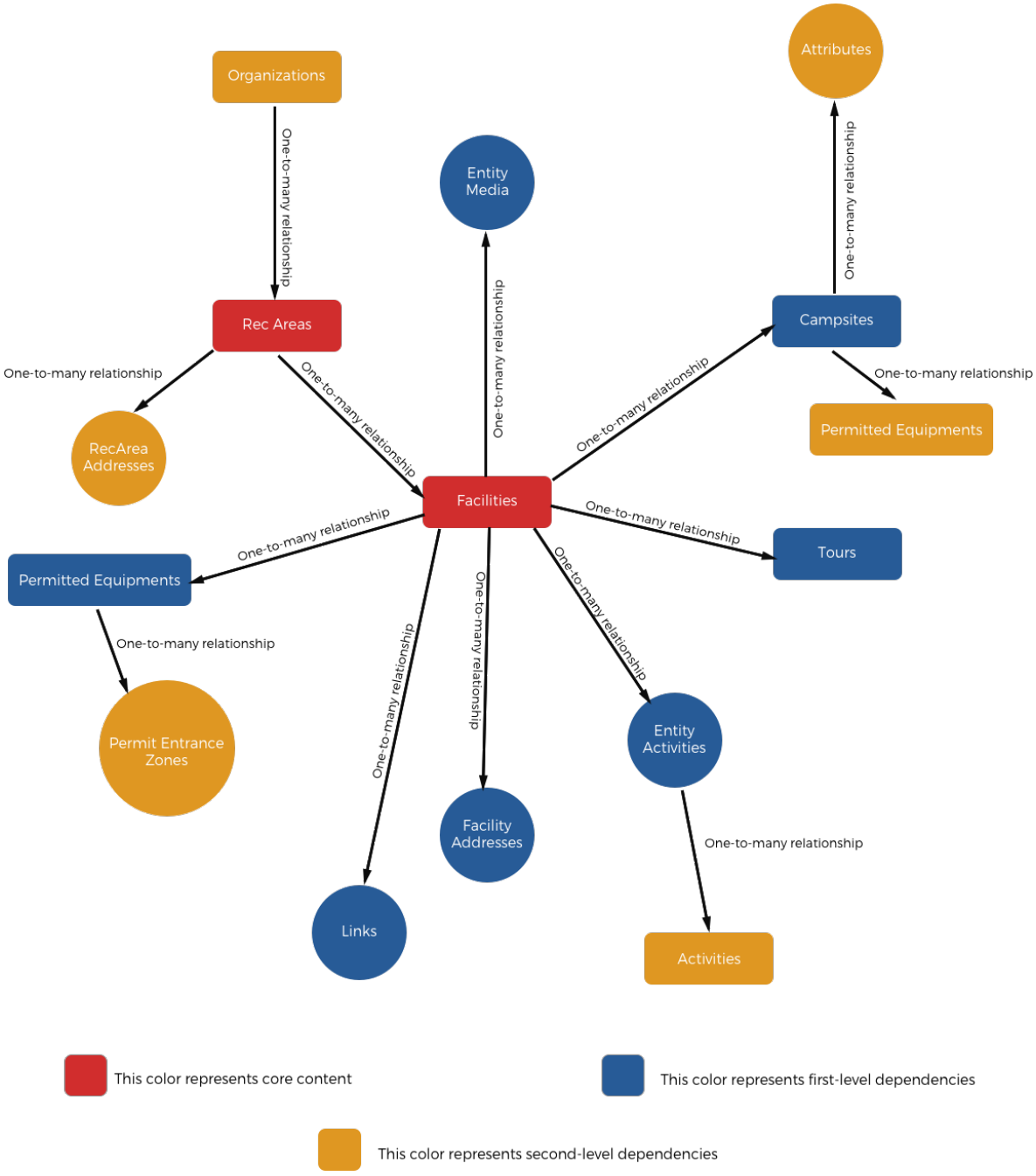
9. Media

Media content constitutes of the hyperlinks to either images or video associated with a recreation area, facility or campsite.

10. Links

Links are hyperlinks that provide information associated with a recreation area or facility. Link type examples are official websites, local partnerships, concessionaires, Flickr, Facebook, YouTube channels, Google+, and Twitter.

III. Entity-Relationship Diagram



IV. RIDB

RIDB (Recreation Information Database) is a part of the Recreation One Stop (Rec1Stop) project, initiated as a result of a government modernization study conducted in 2004. Rec1Stop provides a user-friendly, web-based resource to the

public, offering a single point of access to information about recreational opportunities nationwide. The web site represents an authoritative source of information and services for millions of visitors to federal lands, historic sites, museums, and other attractions/resources.

An RIDB API is also present to provide access to the information regarding federal recreation areas, facilities, campsites, tours, and permits. By the means of various HTTP requests, all the aforementioned types of data can be accessed using this REST web service. In addition to this, the complete data-set is also available for a simple download in the form of .csv files, with different types of content segregated into different files.

V. Database Vs. API Integration

Determining whether to integrate the RIDB API in order to access all the required information, or to implement a database schema to store all the information, and create an update script to handle all the changes in the data was the first core problem we had to resolve when working on this project. It was also one of the most important decisions we had to make as it would determine the way in which the underlying architecture of the complete application would be designed.

Using the RIDB API would have been a very easy undertaking. With the help of the documentation provided, as well as the HTTP request-response architecture available, free of cost, without any limitations, accessing, and querying all the information required seemed to an easy undertaking. But while testing out the API we found out that a single API call requires around 500ms to return the required results. This in itself was not a considerable latency, but due to the complexity of the information we were looking to display on the UI, it would require multiple sequential calls for this API, and hence increase the net latency.

On the other hand, creating a database would allow us the flexibility to model the information in a way which would make the information access easier, and faster. Also, by exploiting the Entity-relationships, it would be possible for us to create complex queries and fetch content from across different content types without the need to initiate multiple sequential queries. This would mean that we would be able to create a much more responsive interface, which would be much faster as compared to the API implementation.

So, by considering all these factors, we decided to go ahead with the idea of designing our own database. The next section describes the implementation of the Database using DJANGO and SQLITE.

VI. Database Design – Using Django Models and SQLite

This section will provide the details regarding how the Database Schema was created using the DJANGO models' framework.

Although the variation in the type of content we encountered was large, the amount of information for each of these types were not too large. Hence, while selecting the database to use, scalability and ability to handle very large data was not a concern. SQLite (specifically, SQLite3) is the database package which has been included by default in the new Python 3.5 version. Hence, we decided to go ahead with using the SQLite database as it provided us with an easier way to integrate the database with the application, which was being developed using Python.

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so that it's users can focus on writing the application without needing to reinvent the wheel. In addition to providing a clean, and modularized method of creating an application, it also allows the flexibility to utilize multiple types of database as the content management layer in the application. Hence, Django was the perfect choice for us to utilize in order to create a well-designed application.

1. Configuring the SQLite database

Django provides an easy method to configure the type of database the user would like to integrate with the framework. Following is an example of how SQLite3 can be integrated with Django:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

This configuration is included in the settings.py file present as a part of the Django framework. The 'ENGINE' is assigned one of the predetermined values, a list of which can be found in the Django documentation. The 'NAME' parameter can be configured to suit the user needs. In this example, the database name is 'db.sqlite3'.

2. Creating the Schema

Django uses a 'models' framework, which is analogous to 'tables' in the case of traditional database systems. Following is an example of how a 'model' can be created using Django:

```
class RecAreas(models.Model, BaseES):

      keywords = models.CharField(max_length=100)

      lastupdateddate = models.CharField(max_length=100)

      orgrecareaid = models.CharField(max_length=100)

      recareadescription =
      models.CharField(max_length=100000)

      recareadirections =
      models.CharField(max_length=100000)

      recareaemail = models.CharField(max_length=500)

      recareafeedescription =
      models.CharField(max_length=100000)

      recareaid =
      models.CharField(max_length=10,primary_key=True)

      recarealatitude = models.CharField(max_length=100)

      recarealongitude = models.CharField(max_length=100)

      recareamapurl = models.CharField(max_length=100)

      recareaname = models.CharField(max_length=1000)

      recareaphone = models.CharField(max_length=1000)

      recareareservationurl =
      models.CharField(max_length=1000)

      staylimit = models.CharField(max_length=20)
```

Here, the class definition of 'RecAreas' acts as a 'create' statement for the 'model'. The fields mentioned in the class are created as distinct columns, with the associated properties. For example:

```
recareaid = models.CharField(max_length=10,primary_key=True)
```

This would result in creation of a column named "recareaid", which would accept string as input, with a maximum length limit of 10 characters. Also, since it has been assigned the attribute 'primary_key=True', this will also function as the primary key for this 'model'.

3. Migrating the data

As mentioned in the RIDB discussion, the complete content was available for download in .csv file formats. Hence, after creating a schema, the next step was to create a script which would migrate all the content from the csv files to the SQLite database.

In order to do this, we had to create a mapping between the columns present in different CSV files, and the columns defined in the different models defined within the Django framework. The script then picked up the content from the CSV files, one row at a time, and depending upon the file from which the content was being picked from, as well as the mapping of the corresponding 'model' being provided, inserted the content into the SQLite database.

Using such an expensive (in terms of processing power) method of storing information would have been incorrect if not for a couple of reasons. Firstly, even though the number of records that were to be transferred was large, the content itself was not susceptible to lots of changes. This meant that once the data was migrated, there would be minimal overhead in terms of updating the information. Secondly, this method of migrating the data was easier to implement, and robust. This meant that even if there were changes in the content from the RIDB API, the current implementation would not cause errors.

VII. Challenges

Even though our team constituted of members with great experience in working with data-sets, and database applications in general, none of us had ever worked with integrating the SQLite database with Django framework. Hence, we faced numerous challenges in the process of implementing the database, and making sure that it functions as expected in acting as the backbone of the complete application architecture.

- *Establishing ER relationships*

  Even though the SQLite database, and the Django framework are well suited for creating Relational databases, the manner of implementation is very different when compared to traditional database applications. A lack of exhaustive

documentation regarding how to handle Entity relationships meant that we had to struggle a lot to get the implementation correct.

- *Defining the Entity-Relationships*
  Even though the content being provided by the RIDB API was well-documented, and segregated, minimal information was provided regarding the relationships between those content types. Mapping information for Primary Key and Foreign Key relationships between different models was completely absent. Hence, it was very difficult for us to determine the relationships, and then define them in terms of One-to-many, Many-to-One, and Many-to-Many relationships.

- *Erroneous Content*
  The content accessible through the RIDB API was inconsistent, and did not contain multiple fields of information that were necessary for us to be able to fully exploit the distinct affordances this information would enable. In order to solve this problem, we had to work on integrating multiple Third-party services to retrieve relevant information. A few examples of the third-party APIs used are 'Custom Google Search API' for Google Image Search, 'Google Maps API' for location and coordinate information, and 'Free-Geo-IP' for geo-location.

Search

Proxima DasMohapatra

I. Introduction

Search is an integral part of this project since for an amateur camper/ outdoor activities enthusiast, finding the right candidates for activities is a very important step towards the decision of going outdoors. Our goal was to present to the user a simplified platform to start the search process, so that he/she doesn't get too overwhelmed by the information present regarding outdoor recreation.

From our user interviews, we realized that it is very challenging to think of one web application as a 'one-stop-shop' for our user base. In most of our interviews, people navigated through multiple websites that they discovered through Google search, collecting information and making a decision at the end of this process. However, we did notice some frustrations and shortcomings in this approach, which we plan on addressing in our product.

We ended up using a combination of technologies to enable search for our application. The website is able to suggest activities and locations while the user is typing, so as to make the search process similar to what users are currently used to in other search platforms such as Google, Yahoo, etc. The second part of our search is to search the RIDB database to return results for the combination of activity and location as entered by the database. The third feature in our search is a "geographic proximity" search, which would enable users to explore activities near the location of interest in a scenario where their search query does not return any results. As a new outdoor enthusiast, our users might not necessarily have a knowledge of the type of activities available around a certain region. In case of searches where no activities of the user's interest are found in the geographic location indicated by the user, we aim to suggest any activities nearby the location searched by the user.

II. Method

We decided to use **Django** (see Technology Glossary for more information) as the web development framework for our application since it is Python-based, which is a common programming language that our entire team is comfortable with. Django uses SQLite as a default database for the application, which is ideal for small sized databases. Hence, we decided to use the default database, since we did not have a very large amount of data from **RIDB** (see Technology Glossary for more information). Django provides with some basic search functionality that can be used

to search the database. However, we needed some additional search capabilities which led to us to explore search tools beyond the Django search.

To make the search on our website more intuitive, we decided to incorporate the auto-suggest feature which is now present in almost all search applications. It helps the user in their decision-making progress, while also informing the user of the range of the data that the user is trying to search for. We decided to incorporate **Elasticsearch** (see Technology Glossary for more information) into our web application architecture to enable this feature. Since Elasticsearch requires indexing of data that needs to be searched by Elasticsearch, we indexed only select attributes which would be used as a type-along suggestion for the user. Initially, our approach was to use Elasticsearch to support the entire search functionality on our website. However, the RIDB database schema was designed such that joining of tables was required to get the desired results. Elasticsearch, however, is not equipped to handle such search queries, since it specializes in distributed search and thus is fundamentally different from a traditional SQL based database. This led us to incorporate Django search with Elasticsearch to get the desired results within a response time that can be acceptable for a complex search. The Django database model helped us in exploiting the Entity-Relationships that existed in the RIDB database. This feature enabled us to leverage primary key and foreign key relationships in order to provide the user with better results without compromising the response time of the web application.

1. Indexing data for Elasticsearch

The Elasticsearch engine requires a separate copy of the data to be "indexed" which can then be made searchable by the engine. This meant modifying the existing Django database models which would index specific attributes from each table. This was done so that Elasticsearch would have access to only the required attributes, making the size of its indexed data small and the performance of the engine faster.

Django models for tables in a database are defined as Python classes. In order to index specific attributes for Elasticsearch, a Meta class was defined which would contain information pertaining to the indexing of specific attributes. Below is an example of such a class:

```
class Meta:

          """Used for Elasticsearch. Meta-fields are used
to customize how a document's metadata associated is
treated."""

     es_index_name = 'django'

     es_type_name = 'organization'

     es_mapping = {

       'properties': {

          'lastupdateddate': {'type':'date'},

          'orgabbrname':{'type':'string','index':'not_ana
          lyzed'},

          'orgid':
          {'type':'integer','index':'not_analyzed'},

          'orgimageurl': {'type':'string'},

          'orgjurisdictiontype':
          {'type':'string','index':'not_analyzed'},

          'orgname': {

               'type':'completion',

               'analyzer': 'simple',

               'preserve_separators': True,

               'preserve_position_increments': True,

               'max_input_length': 500

          }, # Suggest while typing

          'orgparentid':
          {'type':'integer','index':'not_analyzed'},

          'orgtype':
          {'type':'string','index':'not_analyzed'},

          'orgurladdress': {'type':'string'},

          'orgurltext': {'type':'string'},

       }

}
```

Adding such a class within models led us to write an index-creation script, which would go through each rows of the tables specified and would index only the attributes defined in the Meta class, as showed above.

The result of two search tools integration (Elasticsearch and Django) led us to a two-step search process. When the user is typing a location, or an activity in the input fields, Elasticsearch uses its auto-suggest functionality to suggest values from its indexed data to the user as they type. After the user inputs the fields that he/she is interested in and clicks on the Search button, Django searches for the location and/or activity in its SQLite database to come up with relevant results. The web application has different approaches towards coming up with results depending on the user input. Below are details on the various approaches.

2. *Search with activity and no location*

In a scenario where the user searches for an activity of interest, but does not specify a preferred location, the application accesses the user's current location through the web browser to narrow down results. This ensures that the user is not presented with a large result set due to a missing search parameter.

3. *Search with location and no activity*

In this scenario, the search results are limited by the activities available in the specified location of interest. This list has the potential of being long due to a lack of capability on inferencing an activity of interest from the user.

4. *Search results when there are no rec areas found for the given location and activity*

There are multiple combination of activity and location search parameters for which there are no rec areas fulfilling the conditions in the database. In a scenario where no rec areas are found for a specified activity of interest, higher priority is given to the activity and the application suggests rec areas nearby the area of interest that have the specified activity in increasing order of distance from the location of interest.

III. Challenges

- *Integration of Elasticsearch engine and Django*

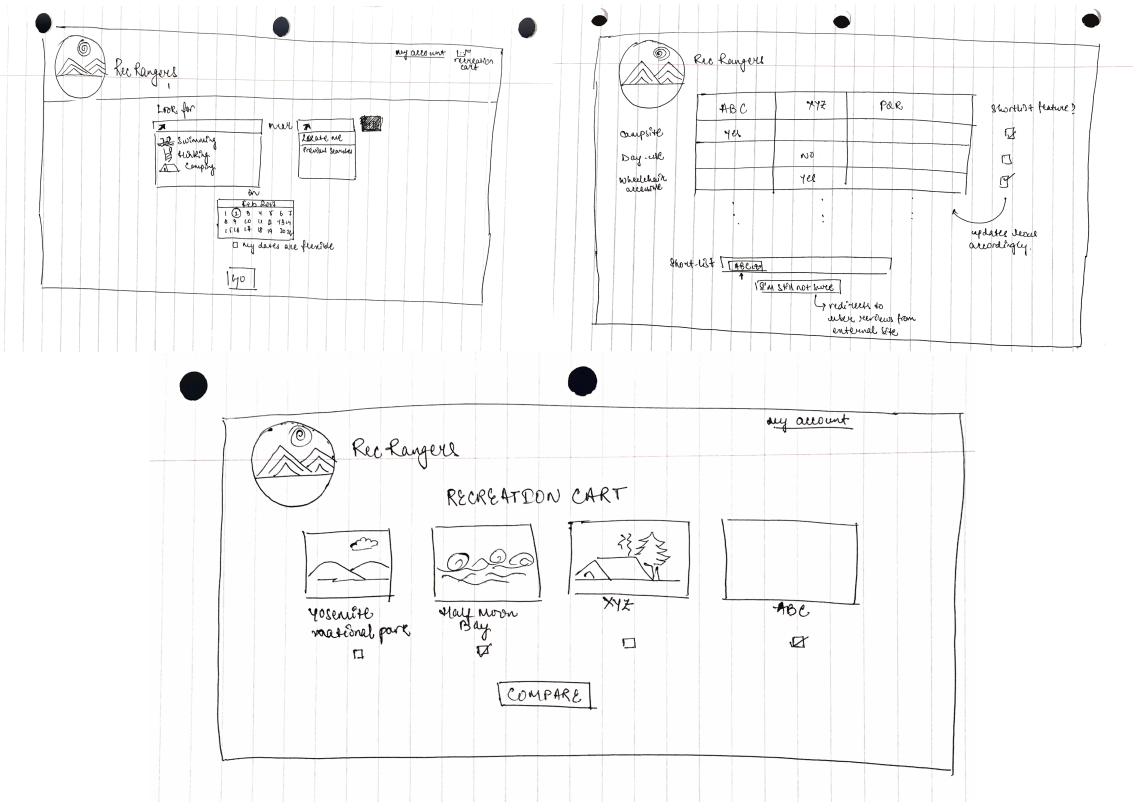
  Making the **RIDB** website more accessible to the users in a more intuitive manner required for us to search the database and add additional information available over the internet in new and innovative ways. Implementing **Elasticsearch** engine with the **Django** framework proved challenging in the beginning, due to limited documentation regarding the integration of the two technologies.

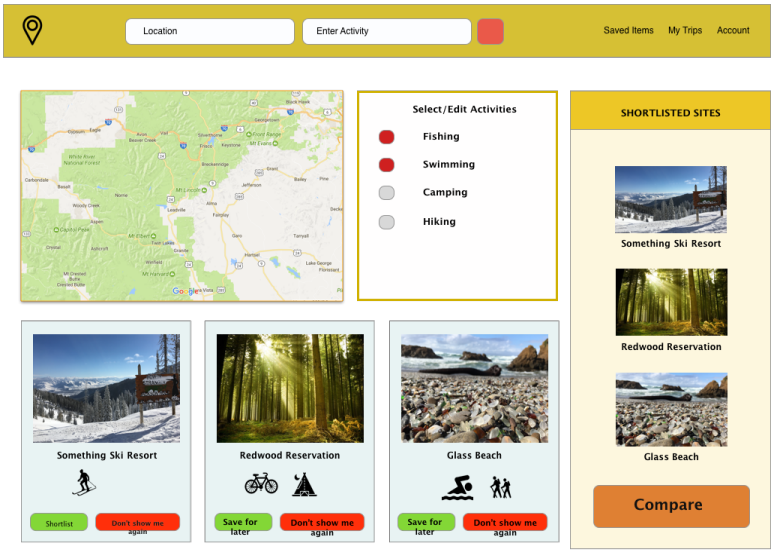
- *Suggesting relevant results in case user's search yields no results from the database*

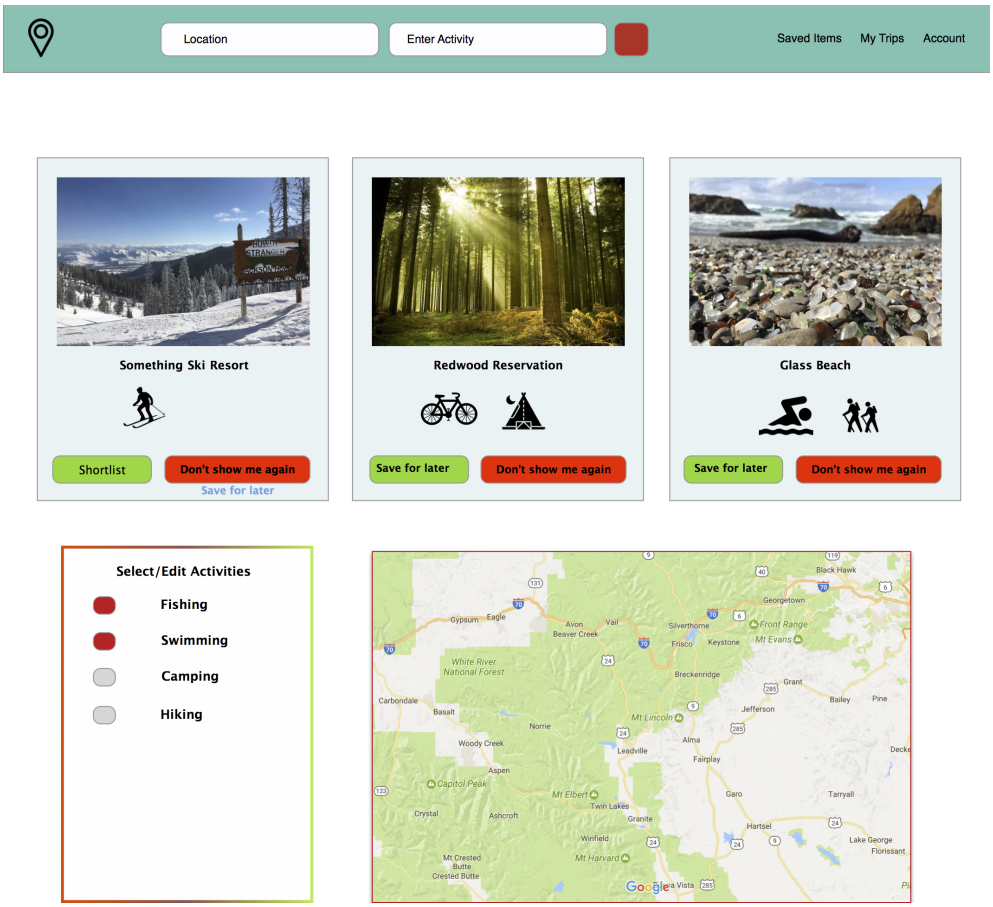
  The second scenario which required a change in the search approach is when the user would be presented with no results in a scenario where his/her search parameters would not have any corresponding data in the database. In order to present the user with some feasible results in order to guide the decision making progress, the web application was integrated with multiple open-source, third-party APIs such as Google Custom Search, Google Maps Search and **Free-Geo-IP** (see Technology Glossary for more information).. These tools have enabled much smarter search results, which would help decrease some of the user's frustrations during the decision-making progress.

Product Design Appendix

1. Wireframes






Location
Enter Activity
Saved Items
My Trips
Account
Something Ski Resort
Shortlist
Don't show me again
Save for later
Redwood Reservation
Save for later
Don't show me again
Glass Beach
Save for later
Don't show me again
Select/Edit Activities
Fishing
Swimming
Camping
Hiking
Black Hawk
Georgetown
Gypsum
Eagle
Avon
Beaver Creek
Vail
Silverthorne
Frisco
Keystone
Front Range
Mt Evans
White River National Forest
Breckenridge
Grant
Bailey
Pine
Carbondale
Basalt
Norrie
Mt Lincoln
Jefferson
Woody Creek
Leadville
Alma
Fairplay
Aspen
Capitol Peak
Mt Elbert
Twin Lakes
Garo
Tarryall
Crystal
Ashcroft
Granite
Hartsel
Winfield
Lake George
Florissant
Mt Harvard
Mt Crested Butte
Crested Butte
Google
a Vista

Technology Glossary

Django

Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "plug-ability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models. Django has a default SQLite database which has been used in our application to store the data exported from the RIDB database. Django also supports a SQL-like search feature on its database, which is used in conjunction with Elasticsearch in this project

Elasticsearch

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source software. We have used the Elasticsearch Python client in our project, since it integrates well with the Django framework, which is also built using Python.

RIDB

Recreation Information Database provides data resources to citizens, offering a single point of access to information about recreational opportunities nationwide. The RIDB represents an authoritative source of information and services for millions of visitors to federal lands, historic sites, museums, and other attractions/resources. This initiative integrates multiple Federal channels and sources about recreation opportunities into a one-stop, searchable database of recreational areas nationwide.

Free-Geo-IP

Freegeoip.net provides a public HTTP API for software developers to search the geolocation of IP addresses. It uses a database of IP addresses that are associated to cities along with other relevant information like time zone, latitude and longitude.