Technical Report

Ameya Joshi, Skylore Evans, Jordan Bogaards
Dylan Kan, Roman Kuhn, Pedro Silva

1 Motivation

The goal of this project is naturally to try and further species protection through information, but also to encourage viewing these species in their natural habitats. While many people are aware of the threats faced by endangered species, we wanted to also encourage sightseeing in national parks to get otherwise inactive out and actually experiencing nature. We connect parks, plants, and animals by their states, hoping to recommend a park where you can find these rare species, while also helping us gauge which states are most involved in contributing to the greater good.

2 User Stories

2.1 Developer Stories

Here are the user stories given to us by our customer, Group 10, for us to implement for each phase of the project.

2.1.1 Phase 3

1. Tables On Animal Instance Pages Are Not Uniform
    - Description: Some of the tables are getting scrunched up on the animal instance pages. They look nicely spaced on most of the pages but on some of the pages, the table looks slightly off. https://www.parkprotection.me/Animals/51 has an example of this on Chrome.
    - Time Estimated: 5 minutes
    - Time Taken: 5 minutes
    - Implementation: The tables were resized because the values of the strings (namely the conservation plan and "Distinct Population Segment") are too long to fit, so they grow vertically. To make it more eye-pleasing, we have made "Distinct Population Segment" "DPS" because that's the biggest reason it looks weird.

2. Cards And Images On Instance Pages Are Not Uniform

- Description: Some of the images on the instance pages end up getting cut off from what the image shows on that pages card. It would be nice to have the full image shown when going from the card to the instance page. This was noticed in some of the animal instance pages.
- Time Estimated: 30 minutes
- Time Taken: 30 minutes
- Implementation: Recommended using a desktop browser as it was discovered after investigation that while the website was responsive for mobile devices and skinny screens, although the aspect ratio of the images would be slightly changed.

3. Error Page Unhandled Rejection
    - Description: When visiting links that shouldn't work, Unhandled Rejection (TypeError): Cannot read property 'map' of undefined was displayed. This occurred at https://parkprotection.me/Animals/asdasd. This should cause the user to be routed to an error page that you have designed.
    - Time Estimated: 5 minutes
    - Time Taken: 20 minutes
    - Implementation: Whenever an assumed instance is incorrect, we now redirect to the error 404 page.

4. Filtering Not Working
    - Description: Currently, the filters do not work when selected. Additionally, the filters for the animals only include birds and reptiles. We would like to see the active filters and updated animal groups..
    - Time Estimated: 10 hours
    - Time Taken: 12 hours
    - Implementation: Implemented filtering with the help of React Select. There is now the possibility for multiple filters. All models now allow sorting by a single attribute, followed by any number of filters. Filters for separate attributes are AND'd while filters within the same attribute are OR'd [(CA or TX) and (Endangered or Threatened)].

5. Issues on mobile
    - Description: When using the site on mobile some of the UI is a bit off. The filter buttons are shrunk and the pictures get cut off.
    - Time Estimated: 15 minutes
    - Time Taken: 10 minutes
    - Implementation: The website was tested at different browser sizes. While they do get closer together, they stay proportional for a good bit and don't clip, so it should classify for the project's guidelines.

2.1.2 Phase 2

1. Add a disclaimer that parks may be closed due to the Covid-19 outbreak
    - Description: People should be sheltered and staying home right now. Parks may not be accepting visitors and people should not be out and about anyways. Consider adding a disclaimer to inform the user of this information.
    - Time Estimated: 10 minutes
    - Time Taken: 20 minutes
    - Implementation: An alert component was placed near the top of the page for each instance page alerting users that the parks may be closed. The alert also provided a link to the CDC's website for more information.
2. Add a 404 page
    - Description: Please add a 404 error page so that the customer knows the page is not available. Currently going to page that doesn't exist just loads an empty page with the top bar. This should be an error page due to the page not actually existing.
    - Time Estimated: 10 minutes
    - Time Taken: 5 minutes
    - Implementation: The website redirects users to a 404 page in the event that they entered in a URL that doesn't exist. This is done by routing through App.js
3. White space on park instance pages is inconsistent with the other model's instance pages
    - Description: The is much more white space on the park instance pages than there is on the plant and animal pages. The white space is in between the contact info and the related plants and animals. Please fix this so that the instance pages can have a consistent style.
    - Time Estimated: 1 hour
    - Time Taken: 1 hour 30 minutes
    - Implementation: A map is now on the park instance page that replaces the whitespace and removes inconsistencies between the park instance page and the plants and animals instance page. This was done using Google Maps api.
4. image misalignment on plant and animal instance pages
    - Description: The images on the instance pages for the plants are off-center. Please fix this so that the style is consistent. Also on the iguana page, a map sometimes loads and if it does, it crashes.
    - Time Estimated: 1 hour
    - Time Taken: 1 hour
    - Implementation: There is now a working map next to the image of the plant/animal and they are centered on the screen.
5. Consider adding a map to each parks instance page

- Description: Consider implementing a static or dynamic map to each park's instance page. This will allow the user to better understand where the park is geographically. Additionally, it will potentially allow users to see what else is nearby for them to create a better trip.
- Time Estimated: 1 hour
- Time Taken: 1 hour 30 minutes
- Implementation: There is now a map on every park instance page, centered on the park's location with a marker in the middle of the park. This lets the user know exactly where in the US the park is. This map was implemented using Google's Google Maps API.

2.1.3 Phase 1

1. They would like us to dynamically load info on commits from Gitlab in the About page. The About page they saw was one that was just static. We had a rough draft website that ended up being the one they reviewed since we couldn't get the full one out soon enough, so this one ended up being already completed by submission.
    a. Time taken - 3 hours
2. They want us to make the "Get Started" button on the Home page clickable. Their version had just a button that did nothing, it was there for the look of the Home page. The current one is linked to the Park page.
    a. Time taken - 15 min
3. They want us to display all the information relevant to search criteria on each detail card on the list pages. We had forgotten about that requirement and were just going to have Cards on the model pages with an image and name. We have since added the 5 attributes that we allow sorting and filtering by as a vertical column of text for all models.
    a. Time taken - 1 hour
4. They want us to add descriptions for information given on 'Animals' pages. Our data source is the US Fish & Wildlife Preserve, and they have many fields like "BCC" and "Distinct Population Segment" as part of their data. We didn't think too much about it, but it makes sense that the average user needs to learn what these terms are. We think that the best way to implement a solution is to make the terms hyperlinks in the tables, and link to a definition of them.
    a. Time estimated - 15 min
5. They want us to make the map on the Parks page readable. Apparently, they had to zoom in too much to see it. We had our own issues with getting the map to be the right size, but we think we've got the hang of it, so it should be a simple matter of trial and error to see what size looks good.
    a. Time estimated - 15 min

2.2 Customer Stories

Here are the user stories given to our developer, Group 2 - 90 min in One, for them to implement for each phase of the project.

2.2.1 Phase 3

1. Sorting, Filtering, and Searching
    - Description - The website has no UI for sorting, searching, and filtering. These are all required by the project guidelines, so if it's only the front-end missing and you focused on the back-end up until now, I recommend you quickly get all that code in. Also, you need 5 filterable/sortable attributes per model on the model pages. You guys have an image + 4 attributes on the model pages. You can't sort or filter by image, so you should consider which other attributes to include for that.
    - Time Estimated - A couple of hours
    - Time Taken - 16 hours, judging by their tracking
    - Implementation - Their response: “Searching (for all models) took about 1 day to finish and apply to all models. Ho-Jae took another hour to improve it so that it sorts search results by most relevance”
2. Related Links
    - Description - I checked all your instance pages and found no related links between models. The project requirements state that each instance should be linked to related instances in the other models. As of right now, I would have to manually figure out the links between the instances and search on my own.
    - Time Estimated - 1 hour
    - Time Taken - 1 hour
    - Implementation - Their response: “For just the teams, once we pass in team_id to players, we can get the links working in about an hour. We need to change our current design (of passing data from model pages to instance pages) to consuming our api and retrieving data for rendering our instance pages.” ““We ran into the issue of passing data but resolved by adding more, appropriate data to our json. Switched to just passing an id around rather than a whole object and then making an API call to fetch the object corresponding to that id in the necessary places.” “Need to add more links though once we have the right relationship data set up. But for now, we have Player instances with their related linked to teams.”
3. Alt for Missing Images
    - Description - I noticed on your League models page that some flags and league logos were missing. I understand that some data is bad and scraping isn't perfect, but I think an alt text saying "Flag unknown" or "Country unknown" would

be better. Even better if you photoshop a default "No Flag" image to replace all the missing images, but the alt text is good enough.
    - Time Estimated - A couple of minutes
    - Time Taken - Not reported, but assuming a couple of minutes
    - Implementation - Their response: “We decided to reduce the size of our leagues data and in turn, that will get rid of all the missing data. This should take just a couple minutes as they are just SQL commands.”
4. Remove Field Titles on Instance Pages
    - Description - Your instance pages have all the attribute name followed by its value. While that's fine for "Standings? No", you should consider removing it for more common-sense fields, like "Flag: ". Just having the flag image gets the message across and looks cleaner.
    - Time Estimated - 3 hours
    - Time Taken - 2 hours
    - Implementation - Their response: “We are planning to completely redesign the layout of our instance pages, and that will in turn take care of these field titles. We estimate that to take about 3 hours” “Completed in 2 hours. We need to collect more attributes and will update the instance pages later when we do.”
5. About Page Fixes
    - Description - The About page looks much better now, great job. Only two quick tweaks. First, I can't find a link to your Gitlab Repo. That's required by the project guidelines. Second, I think that your bios need to have more than just "full-stack developer" since Professor Downing told us that we have to be more specific on what we worked on.
    - Time Estimated - A couple of minutes
    - Time Taken - 30 minutes
    - Implementation - Their response: “The link to our Gitlab repo should just be the Gitlab icon. Updating our roles will be a quick fix. It should only take a couple of minutes.” “It just came to my attention that since we have not deployed our website, our customers did not see the most updated version of our website, and therefore did not have the links. Those links took about 30 minutes to add.”

2.2.2 Phase 2

1. Move Contact Info to About
    - Description - The Contact page feels pretty bare if all it's supposed to do is provide an email address. I would recommend removing it entirely and having the link in the About page. It's perfectly fine to have any contact info in the About page of a website, and that way there isn't a page with 90% whitespace.
    - Time Estimated - 30 minutes
    - Time Taken - 10 minutes

2. Add a mission statement to the About page
    - Description - I saw that you guys added a lot of the things missing last time to the About page. However, I still don't see a mission statement. It would be nice to see a solid description of the website rather than just seeing that it has something to do with soccer.
    - Time Estimated - 10 minutes
    - Time Taken - 10 minutes

3. Two pieces of media on instances
    - Description - You have two images for the Leagues instances, but only one image for Teams and Players instances. One of the requirements for the instances was that there be two examples of media for each instance. I would recommend adding at least a flag representing the team and players' country.
    - Time Estimated - 3 hours
    - Time Taken - Not reported directly, assuming 3 hours based on their tracking

4. Not all instances on the models page are clickable/route
    - Description - I tried to check out different instances for a model, but not all of them directed me to an instance page. You should check your routing. The top few in the table route fine, but none after that seem to work.
    - Time Estimated - 2 hours
    - Time Taken - Not reported directly, assuming 5 hours based on their tracking

5. Pretty attribute names
    - Description - Some of your attribute names are displayed just like variable names, like "venue_address". While I understand that it's easy for testing their values, it looks kind of awkward. Just put in pretty titles like "Venue Address" or "Address" instead.
    - Time Estimated - 1 hour
    - Time Taken - Not reported directly, assuming 1 hour based on their tracking

2.2.3 Phase 1

1. We want to see players that are on the same team in one place. Their instances page didn’t have any filters in mind, just a list of players. Grouping them by teams is useful when interested in the team overall. We’re thinking they should point to each other’s instances.
    a. Time estimated - 1 hour
2. We want to be able to browse players. There was no indication that searching was possible for players. We’d prefer not to have to scroll through a randomly listed table to find individuals. A search bar or sorting alphabetically is useful enough.

    a. Time estimated - 15 min (they only need to put the search/filter bar in, not process any requests yet)
3. We want to be able to see a team's history. That's useful to answer the questions of if a team is getting better or not. A table of win/loss is enough for this.
    a. Time estimated - 30 min
4. We want to know which teams are from which country. That's useful if we want to find information on a player from our country. This should be a displayed attribute, possibly filterable, on the models' page for teams.
    a. Time estimated - 15 min
5. We want a way to compare teams. This goes along with seeing a team's history. With that, they can also rank teams. Some way to filter by win/loss on team model would be nice for this.
    a. Time estimated - 15 min (again, just the filter bar)
6. We want more media on the Home page. There are 3 coloured buttons, which is nice, but a logo or something more would be more flashy and attractive. They already have a soccer ball as their icon, so another one would be good enough to let people know that this website is about soccer.
    a. Time estimated: 15 min
7. We want to see a mission statement, headshots, and links to tools on About page. They went ahead and did the dynamic pulling from Gitlab first, so they did the hard part first. They need to have all the other, static info and design it as well.
    a. Time estimated: 1.5 hours
8. We want to see all 5 filterable/sortable attributes on models pages. They currently have only 3 attributes for each instance on the models' pages. Since 5 of them minimum need to be filterable/sortable, letting us see them would be appreciated.
    a. Time estimated: 15 min
9. We want to see at least 10 attributes on the instance pages. Again, they only have 3 attributes per page. The minimum requirement was 10.
    a. Time estimated: 4 hours
10. We want to see the Contacts model filled. It was empty when we got it, so we're assuming they have to get the data and design the page still.
    a. Time estimated: 1 hour

3 RESTful API

We have documentation for the API on Postman. We have 2 endpoints for each model of parks, plants, and animals for a total of six. The first endpoint for each is to get a list, designed to populate the models' pages. We have optional queries for pagination handled by Flask-Restless to restrict the page and page size. The list elements will be model instances in JSON format. The second endpoint for each model is designed for getting individual model instances and

loading instance pages. The unique keys used for identification are from the original APIs, either the park code or ECOS ID.

4 Models

4.1 Relation

All of our models are related to each other by their respective states. We store where every park, animal, and plant can occur in the 48 continental US states. While it may seem a bit too arbitrary at first, there's actually a reason we chose this relation. The main ideas behind the website are to identify state conservation efforts and to locate areas to view the endangered species in their natural habitat. We can clearly see state efforts through this link, and the homogeneity of the natural environment within a state makes a safe environment like a natural park have the highest likelihood of containing these rare species. So, our relation basically shows where a species can be encountered and what other species could neighbor it.

4.2 Parks

The parks are where the plants and animals might be located. They are connected to The data for them is from the National Park Services' API.

- Sortable Attributes
    - Names
    - Emails
    - Phone Numbers
    - Designations
- Filterable Attributes
    - States
- Descriptive Attributes
    - Images
    - GPS Coordinates (Displayed on a map)
    - Park Code
    - Description
    - Directions Info
    - Weather Info
    - Addresses
    - Website

4.3 Plants

One of the categories we divided endangered species into. The data for them is from both the US Fish & Wildlife Services Environmental Conservation Online System and a USDA sponsored API. For the images we used Bing's image search API and for the descriptions we used Wikipedia's API.

- Sortable Attributes
    - Common Names
    - Scientific Names
    - Family Common Names
- Filterable Attributes
    - ESA Listing Statuses
    - States (Displayed with highlighting on a map)
- Descriptive Attributes
    - Images
    - Wikipedia Description
    - Family Scientific
    - Family Common
    - Category
    - Duration
    - Growth Habit
    - Toxicity

4.4 Animals

The other category we divided endangered species into. Their data is from the US Fish & Wildlife Services Environmental Conservation Online System. For the images we used Bing's image search API and for the descriptions we used Wikipedia's API.

- Sortable Attributes
    - Common Names
    - Scientific Names
- Filterable Attributes
    - Groups
    - ESA Listing Statuses
    - States (Displayed with highlighting on a map)
- Descriptive Attributes
    - Images

    - Wikipedia Description
    - ESA Listing Date
    - Are they a distinct population segment?
    - Are they aquatic?
    - Are they Birds of Conservation Concern (BCC)?
    - Conservation Plan Title, if one exists

5 Tools

Along with the above APIs and the necessary tools React, Bootstrap, React-DOM, and the testing frameworks, we used 3 optional tools.

- React Select - We wanted to allow selection of multiple filters (e.x. Filter both reptiles and birds), but React Bootstrap's dropdowns don't support multi-select by default, so we decided to use this instead. It's a dropdown with high levels of customizability, but we plan on only using the basic provided ones with no customization.

- Google Maps React - As the name suggests, it allows embedding a Google map with React. We use it for park instances to show GPS coordinates of parks and for animals and plant instances to show the range of states they are present in.

- Styled Components - This tool is to allow easier styling for components. We saw this tool in a React Bootstrap tutorial, so we just followed along and stuck with it. We're using it for simple styling, like centering text in a column.

6 Hosting

6.1 S3

Currently deprecated, we used to use it for hosting the static front end of our website. While it would've been easier to keep on using S3 to host the frontend, we needed environment variables to secure our Google Maps API key and S3 doesn't support environments. We might go back to it later because while our API key is secure from Gitlab, it is still visible if anyone inspects the website itself.

6.2 Cloudfront

Currently deprecated, it handled delivery of the static front end when it was on S3 by working as a proxy. It allowed us to secure the content delivery with HTTPS. We did not use it for HTTPS with EB because we read that it was better to use listeners for the load balancer.

6.3 Elastic Beanstalk

Elastic Beanstalk hosts both our frontend and API instances with Docker. We used our SSL certificate to secure both instances and routed to it using Route53. It was very simple to add environmental variables to it to secure our database access for the API, and we're looking into a way to secure the Google Maps API key beyond our current method.

6.4 NameCheap

NameCheap is where we got our domain name parkprotection.me which is in a hosted zone in Route53 to direct to the EB instances.

6.5 Route53

Route53 has the nameservers of our domain and handles resolution of our domain names to the EB instances.

6.5 RDS

RDS is used to host our PostgreSQL database and populate our API.

6.6 ACM

ACM provides us with a free SSL certificate to allow secure delivery of our content.

7 Pagination

Pagination is handled on the backend almost entirely by Flask-Restless itself. Flask-Restless needs us to define our database in tandem with SQLAlchemy and creates all the endpoints for us. It also allows optional parameters for the models endpoint describing a number of elements to show per page and the current page, which is what we fetched from in the front end. The pagination on the front end itself consists of getting a list of instances from the API and dynamically making a card for each instance to populate with the data.

8 Database

We used a PostgreSQL database available through our API using Flask-Restless. The tables briefly described are:

- Parks - Contains instances of all the national parks with the attributes described above.
- Park States - Represents a one-to-many relationship between a park and the states it is located in, with a row containing a state and a foreign key referring to the related park.
- Plants - Contains instances of all the plants with the attributes described above.
- Plant States - Represents a one-to-many relationship between a plant and the states it is located in, with a row containing a state and a foreign key referring to the related plant.
- Animals - Contains instances of all the animals with the attributes described above.
- Animal States - Represents a one-to-many relationship between an animal and the states it is located in, with a row containing a state and a foreign key referring to the related animal.

9 Testing

9.1 Mocha Tests

Mocha is a Javascript test framework that makes testing Javascript code easy. We used mocha to test several of the pages. Specifically, the mocha tests checked to make sure the website was not returning any errors and that several pages from the website were loading properly.

9.2 Selenium Tests

Selenium Webdriver is a collection of tools used to automate testing of a web application. Selenium is used in our web application to automate acceptance tests by creating a browser and programmatically clicking buttons and exploring things on a webpage. We use this to check if expected behaviors are confirmed when items are clicked, and that information is displaying correctly on the webpages.

9.3 Postman Unit Tests

Postman is a tool to help design APIs. We used it in Phase 1 to come up with a skeleton API that didn't work but allowed us to design our front end without knowing concretely how our API would function. It also features the ability to test responses from a remote url, so we used that for testing the results of our API. We test the default endpoints for our API along with the

optional queries for pagination, sorting, filtering, and searching. The API testing was defined as a collection and was exported as a JSON which we can test through the Newman CLI.

9.4 Python Unit Tests

We have two types of tests for our Python tests, one testing if our backend was accessible through code and one testing our scraping. The tests for our backend are similar to Postman - they test the API endpoints along with the optional queries for pagination, sorting, filtering, and searching. For scraping, we felt it prudent to test our results because the third-party APIs we scraped from and had a lot of plants and animals that we couldn't use, and we had to use multiple APIs and processing to get the desired data, and even then there were unexpected values for images and other fields, so we added tests where we separated out the requests to each of the APIs and tested to see that the values that going in and coming out of them were valid.

10 Searching

Searching capabilities were implemented for each model page individually, as well as a global search that will display search results for each model. A user has the capability to search for a specific plant, animal, or park, on each of the respective model pages. This was implemented by sending the search keywords from the front-end to the back-end where in a Flask-Restless post-processor, every instance of the model page is searched through. If a particular entry does not contain the relevant keywords, it is removed. However, if the entry does contain the relevant keywords, those strings are highlighted and finally sent to the front end. The front-end has a separate page that displays the matching search results as well as what specifically the keywords matched to, for the user's benefit. The searching implemented is google-like, meaning that if the user searches for "foo bar" then the search will find entries that have either "foo" or "bar" contained within them. If a user finds a card on the search results page they wish to visit, they can simply click on the card to be routed to that specific entries instance page.

11 Sorting

We implemented sorting with a method that runs as an on change function for a react select component. The method we created for each model's page, SortSelectHandler, keeps track of the field and direction with which to sort. Any time there is a change in the selected option in the Select component, this method runs. SortSelectHandler forces the list of instances to be refilled in the correct order by making a fetch call with the correct "orderby" attributes. after the list is refilled, SortSelectHandler forces a re-render which updates the page with the instances in the desired order.

12 Filtering

We implemented filtering for each of our model pages. The parks can be filtered by state, the plants can be filtered by state and listing status, and the animals can be filtered by state, listing status, and group. Each of these filters are presented as a dropdown on each of the models pages, and multiple filters can be applied at once. In the backend, filter selections trigger a call to a function that takes in the list of parameters and which kind of filter is being used, and stores that list globally. Then, it calls a function to send an API request to fetch the results that match the new filter list, and forces a page reload. The API request function simply parses all of the relevant global lists and appends the proper parameters to the URL that we send a request to, and the reload causes the new data to be displayed on the page. The number of pages on the models page is also changed dynamically to reflect the number of results after the filter is applied, so we don’t see any blank pages.