HikeAdvisor Technical Report

Motivation

Hiking is a hobby enjoyed by many, but it is hard to find reliable information about hiking trails that a hobbyist may be looking for. With HikeAdvisor, we hope to fill this lack of information and provide a hub for people to plan their hike beforehand. Our site includes important information about a multitude of hiking trails that a hobbyist might find useful such as animals on the trail, elevation, grade, and much more.

Models

HikeAdvisor is broken into three models, each one connecting to the other in some form:

- *Hiking Trails* - The main model of the website. Each hiking trail will have a list of *Animals* that can be seen on the hiking trail. The hiking trail will also have the *State* it is located in.
- *Animals* - Each animal will have a list of *Hiking Trails* and *States* that the animal can be found in.
- *States* - Each state will have a list of *Hiking Trails* and *Animals* that are located in that state.

Source

For our data, we scraped from the following API:

- https://www.inaturalist.org/pages/api+reference: The Inaturalist API
- https://www.hikingproject.com/data: The HikingProject API
- https://timezonedb.com/api: The TimeZoneDB API
- https://products.wolframalpha.com/api/: The Wolframalpha API

User stories

From our Customer

User stories for phase 1:

- *Add a favicon* - Created a custom favicon for the website
- *Link to API documentation on About page* - Listed under the tools section of the About page

- *Weather information on each trail* - Each trail has general weather conditions as an attribute
- *Information about animals on animals page* - Three animal instances on the animal page
- *Information about cities on cities page* - Three state instances on the state page (we changed from cities to states)
- *About page stats* - Fixed a few typos and fixed the about page not showing stats.

User stories for phase 2:

- *Animal instances* - Fixed a typo and included all info about each animal on their respective pages (instead of just showing the description of each animal).
- *More trails* - Changed layout of trails to tile in a rectangle instead of in a vertical line. Also added filtering trails by states.
- *More animals* - Added more categories of animals to the animal tab (by making site dynamic, pulling from API).
- *More states* - Added all 50 states to the states tab (also pulling from API).

User stories for phase 3:

- *Pagination* - Implemented more sophisticated pagination where the user can jump to the first and last pages.
- *Searching* - Implemented search using navigation search component.
- *Sorting* - Implemented sorting for each model. Each model now can be sorted in ascending order according to particular criteria (number of rating, length … etc).
- *Filtering* - Added filtering where the user can only display models that meet a certain criteria (ex. only show results that have a peak elevation of at least 1000 feet).
- *Search Box* - Added general search bar that searches all the models

To our Developers

User stories for phase 1:

- As a user, I would like to see where these reports are happening on a map so I can visually see which areas to avoid/look out for. It would be nice to have an interactive map (or even just a snapshot of a map) showing the location of the report. This way, we could see what streets/areas to avoid more easily.
- As someone looking for housing, I would want to see how much money I would spend in the areas listed. With this, I could decide whether or not I could afford to

live in that area. It would also help if there were more resources to help secure housing in that area (such as a link or something).

- As someone who is new to living in Austin, I would want to know general safety tips about the areas around Austin so that I know what things to look out for. It would be nice if somewhere on the website there were general safety tips about staying in Austin. This would be especially helpful for foreigners who might not be used to living in such an environment.
- As a user, I would like to see a heat map showing where most reports are made so that I can easily figure out what areas are busy. This would be especially useful for traffic reports. It could also help with people who are with their children and want to avoid dangerous areas.
- As someone who is trying to be safe in Austin, I would want to have contact info such as a phone number for public emergency services. This would be useful in many situations and would many during emergencies. It would also be nice to have a description of what each service is for.

User stories for phase 2:

- I noticed the live website isn't up yet. I think it would be a good idea to have it up soon! It would make testing easier on me since I wouldn't have to pull the files from your repo, and it would also help me distinguish in-development versions of your website from live versions.
- I noticed a small typo in the about page. Under the Affordable Housing data source, it says "This data include" when it should be "This data includes". Not too big of an issue, but I thought it was something that could be easily overlooked so I felt the need to point it out.
- I think the site would look better and have a bit more color if you guys had some type of symbol for your website and maybe place it on the top left beside your website name. I feel like a simple, modern, material design style symbol fit with your website. Some ideas for a symbol could be a safety related symbol or maybe something related to Austin.
- I feel like you guys should consider removing the "Keep Austin Safe" header in each of the model tabs. I just think it doesn't really serve any purpose and just takes up unnecessary space. Instead of that same header for each tab, you could perhaps include a short description of each model.
- I feel like the click on the website title (on the top left) should take you to the home page. I understand that there is a home button right next to it, but instinctively I would expect that pressing the website title would also bring me to the home page (as many modern websites behave this way). It seems like this

request is kind of nit-picky but I think adding this simple feature would improve the usability/flow of your website.

User stories for phase 3:

- You guys did an excellent job on my previous user story to implement general safety tips! The info included is great, and it's exactly what I was looking for, however I think you could do a better job on the formatting. I feel like it would look much better and be much easier to read if you included bullet points or some type of numeration for each tip.
- The new logo is excellent and it looks great, however I noticed that it caused the top bar to increase in height. It looks a bit off because the text is still the same size, so the bar just looks too big for the text. Maybe you guys should increase the size of the text or try to get the bar to decrease in size so everything matches in size.
- I noticed in every category the titles of the first group headers are right justified, while the rest of the headers are left justified. This makes things look a bit off and makes the first header look like it shouldn't be there. It would look better if you guys keep the format of the titles consistent with each other.
- I noticed you guys don't have any records available yet. I think it would be nice if you got them up soon! It would allow me to test more things about your site such as if the grouping works.
- I noticed that the about page lists no unit tests at all for anybody. This seems like it could be a bug, as unit tests were required for the last phase. I also noticed that all counts are 0 for Sarvesh, which also seems like it could be a bug.

RESTful API

The design/documentation for the RESTful HikeAdvisor API is on Postman..

The RESTful API has a GET endpoint for each of the three models:

api/animal - Returns the list of animal instances

api/trail - Returns the list of hiking trail instances

api/state - Returns the list of State instances

Each endpoint also has one or two identifying parameters, which provides more information about a specified instance:

api/animal/

- *id*

api/trail/

- *id*

api/state/

- *State_name*

api/search?q=

- *search_query*

Tools

[Backend]

- *AWS* - Hosts and deploys to the website
- *Postman* - Designed the RESTful API here
- *Namecheap* - Obtained the domain from here
- *Gitlab* - Used for collaboration and version control
- *Slack* - Used to communicate and collaborate between team members
- *Docker* - Used OS-level virtualization to deliver software in packages called container
- *Flask/Flask Restless* - A micro web framework for Python. It acts as a framework for various extensions that can add web functionality
- *SQLAlchemy* - A Python SQL library and Object Relational Mapper. It is a popular option for implementing SQL functionality within Python. It has a data mapper pattern, meaning that it maps classes into the database to create an object model and database schema
- *Flask SQLAlchemy* - An extension used to implement SQLAlchemy
- *PostgreSQL* - A powerful database tool that is object-relational
- *Elastic Beanstalk* - Using Elastic Beanstalk, we were able to setup Dockerfile which is running on EB instance
- *Python UnitTests* - Used to test our back end files which are running in Python

[Frontend]

- *React* - Use to render page components

- *React-Router* - Used to handle navigation between pages
- *Material-UI* - Used to theme and style our components
- *Material-UI* Table - used to visualize query results
- *Mocha* - A Javascript test framework for Node.js programs, featuring browser support, asynchronous testing, test coverage reports, and use of any assertion library
- *Selenium* - A portable framework for testing web applications
- *Docker* - Used OS-level virtualization to deliver software in packages called container
- *Elastic Beanstalk* - Using Elastic Beanstalk, we were able to setup Dockerfile which is running on EB instance

Hosting

Phase 1

For this specific task, we obtained a free domain name https://hikeadvisor.me from https://www.namecheap.com/. We then host our static website on AWS (Amazon Web Service) using S3, Cloudfront, and Route53. We utilized S3 for storage and Cloudfront to deliver our website efficiently. We configured Cloudfront with custom SSL to support HTTPS and also redirect all HTTP traffic to HTTPS. We then used Route53 to configure our domain name obtained from https://www.namecheap.com/. Only for this phase, we use S3, Cloudfront, and Route53 to host a static website; however, we will have to switch to Elastic Beanstalk soon since for the next phase we have to host a dynamic website. This concludes the hosting task for this phase.

Phase 2

For the second phase of the project, we continue using the free domain name https://hikeadvisor.me obtained from https://www.namecheap.com/; however, we do not use S3 and Cloudfront from phase 1 for this phase since we now no longer host a static website but a dynamic one as we already mentioned about this from phase 1. Starting from this phase, we now host our website using Route 53 and EB (Elastic Beanstalk) with high availability configuration since free tier configuration does not support ELB (Elastic Load Balancer). We set up two EB, one for the back end and the other for the front end. This concludes the hosting task for the second phase.

Task

Database

We created a PostgreSQL database on AWS RDS. Using the data source specified in the **Model** section, we collected/scraped the data and put it in the database. Unfortunately, we, the back end team, forgot to add connections between models so we had to implement a search for our front end to use to connect models. The detailed description about it can be found in the file app.py. By implementing the search, we do not have to redesign the database.

Pagination

We used the **material-ui-flat-pagination** component in order to allow the user to access different pages. Each model has an offset variable that keeps track of the page offset, this is parsed from the url using the **useParams** function from **react-router-dom**. When a user clicks on the next page button it takes them to the url /trails/{the next page number} which is rerendered based off of the url. To fetch the data for each page we call our api with the page offset.

Testing

[Backend]

We wrote unit tests for the backend, which can be found in backend/unit_tests.py. These tests use the Python built in unittest package and cover various aspects of the code to seed the database. Additionally, we made unit tests for our API using Postman. When a request is sent, the tests are run on the response to make sure that the API is working correctly.

[Frontend]

For the frontend, we used Mocha to write unit tests for various parts of the frontend. Also, we created acceptance tests of the GUI using Selenium. By using these tools, we can determine if our website works correctly.

Filtering

[Frontend]

For filtering the data on the frontend we used a FormControl from the material-ui library that allowed us to create a dropdown of selections the user can use to filter through our models. For both trails and animals we can filter by state and for states we filter by timezone. To do this we used the help of our search function that allowed us to search

through the key that we wanted either state or time zone that related to our data and if it was a match, we selected it. You can use the feature on all 3 of our model pages.

Searching

[Backend]

In the backend, we implemented a Flask route that handles the query and searches all the models based on that inputted query. In the route, the code loops through all the rows in the database and checks if the inputted query appears in any of the attributes in each row. Next, it returns the instances of each model that match the inputted query as the response.

Example API call: https://api.hikeadvisor.me/api/search?q=Texas

[Frontend]

We implemented the search by creating a Promise that allows us to use search without having entered a search or not because the Promise object is simply a wrapper around a value that may or may not be known when the object is instantiated and provides a method for handling the value after it is known. By doing this it doesn't matter if a user hasn't started typing because search won't run but once a user does start typing into our search the feature will keep up word for word or even deletion which we compare to our title names to see if the search is contained by the title. The search only returns the titles that contain the query and that is what we return. Then for the highlighting trick we use react-highlighter to highlight the contained query within our title.

Sorting

[Frontend]

We implemented sorting in a similar way to how we displayed our filter dropdown. Likewise we used a FormControl to help us create the dropdown with the choices based on our models data. Like our search feature which uses a promise we do something similar here but instead of a letter by letter typing, the promise uses either a click or it isn't instantiated. Once the user clicks on a menu item from our FormControl, we run that Promise object which sorts in descending order the data that is passed in. All of our models contain data in the form of numbers so it is useful for sorting a lot of the data we pass in. Once it runs all of our data is sorted according to which data point we chose.