InclusiAbility: Phase 3

Group 05
Arul Tayal, Alexander Lee, Brandon Ling, Nick Favoriti, Ishaan Salhotra

Purposes/Motivation:

Our project strives to provide a detailed listing of disabilities and the diversity within this community. We also plan to include information about the various resources available for people in need, such as accessible places/events and technologies/services that can assist those with disabilities.

Website Architecture

For the Project, our architecture consists of the Frontend, which was built using React, and Backend, which was built using Flask.

- Frontend (React)
    - Bootstrap:
        - Easy and consistent styling throughout
    - public folder:
        - contains images
        - people folder:
            - images of the team for about page
        - tools folder:
            - images of the tools we used for about page
    - Split src into:
        - components
            - contains the different custom components we used
            - i.e. navbar
        - data
            - contains json files with the website data
        - images
            - contains the images that are directly imported
        - pages
            - contains each page in the site
        - styles
            - contains the css styling for each page
- Backend (Flask)
    - App
        - Contains the endpoints for model pages to get all the instances
    - Database

        - Contains how the database is set up, including all variable names
    - Scraping
        - Contains all the different websites we scraped
        - Contains the completed and filled in databases
    - Tests
        - Contains our unit tests for checking all endpoints, ensuring we are getting the correct data back from the API calls
    - Routers
        - Contains connecting our different models together to work as a whole for the website
        - Where the search / filtering / sorting is taken care of within the specific model files

User Stories (Phase 1)

1. Number of Instances
    a. Customer - “I would like to see more instances as right now it seems you are planning to have 20, 30, 30. As a reminder, we need to have 50, 100, 150 instances.”
    b. We handled this Customer request by expanding to the state of Texas, rather than limiting ourselves to Austin. This allowed us to get more information for the things to do as well as resources. In regards to disabilities, we are going to scrape more specific disabilities, rather than looking at a higher level.
2. Media
    a. Customer - “Looking at the media you want to provide I see you are providing pictures of disabilities. I would like to see a graphical representation of how many people have that disability instead.”
    b. We handled this Customer request by implementing some basic graphical representations for now, but we will work on scraping better images in the next phase as we scrape APIs.
3. Resources
    a. Customer - “Right now it looks like you guys are leaning toward treatments as resources. Maybe also have help centers, hospitals, and organizations that provide support to disabled people as well.”
    b. We handled this Customer request by expanding our reach for treatments, which also helped us get more instances for resources. We will scrape these other data points as we begin web scraping.

4. What the website will display
    a. Customer - “You say you will provide an "overview of what disabilities are" in the project description but I would rather like to see detailed descriptions on different types of disabilities, like how it hinders the people affected and their struggles.”
    b. We handled this Customer request by making our descriptions more specific for each instance, handling the specific request given above. For our future instances, we will be sure to scrape specific information like those given above by our customer.
5. Sortable Topics
    a. Customer - “Have more ways to sort and filter your attributes. So like for resources, maybe a distance attribute so people can see the resources closest to them.”
    b. We handled this Customer request. We have manually sorted the instances, but as sorting features are not required to be implemented yet, we do not have the drop down yet. However, manually they are sorted. In future phases, we will have the sort and filter options, in which we will include other attributes like the distance example given.

User Stories (Phase 2)

1. Add a video to the splash page
    a. We handled this customer request by adding a link on the splash page directing to a general video about disabilities.
2. Get rid of the gray bar at the bottom of the cards
    a. We handled this customer request by getting rid of the gray bar at the bottom of the cards. This was a simple CSS change that took the gray bar away.
3. Have a planner
    a. Customer: “Lets users pick locations and it will output the distances from the events”
    b. This is a great idea that we had planned to implement in Phase 3 of the project. Right now, we do not have sorting and searching implementations, so being able to complete this user story would be beyond the scope of this phase. We have contacted our developer group and have moved this implementation into phase 3 once we work on sorting and are able to ask our mentor for more help and guidance into this.
4. Need 5th attribute to sort by
    a. Customer: “Needed for disabilities and resources models”

    b. We handled this customer request by adding a 5th attribute to sort by for both the disabilities and resources models.
5. Types of events/locations
    a. Customer: “Add volunteer events that are held on a schedule (like weekly or monthly) for disabled people to go to.”
    b. We handled this customer request by incorporating a wide variety of events, which also include regular volunteer events that occur.

User Stories (Phase 3)

1. Get rid of grey bar on instance cards
    a. We handled this by removing the gray bar that we previously had on each instance card for each of the 3 models. We thought this was a great suggestion as the simplicity of the white color throughout looks more aesthetic, and the blue button sticks out more now as well.
2. Add a sort by category for disabilities model
    a. We handle this by adding a drop down to select one of the different categories for disabilities. This includes categories such as Autism, Brain Impairment, Hearing Impairment, etc. This was yet another great customer request as it enables easy searching through the 100+ instances of different disabilities.
3. Add a sort by cost for event model
    a. We discussed this with our customer group as we do not have a cost feature for the events, but rather a cost feature for the resources. Instead of sorting by cost for the event model, we sorted by ratings as that is one of the features we have available in this model. We communicated this with the customer group.
4. Add a sort by price for resources model
    a. We handled this customer request by adding a button to sort by price for the resource model. A customer can sort in either ascending or descending order.
5. Add a video to splash page
    a. We handled this customer request by adding a redirect link to YouTube with a general video on the outlook of Disabilities. This general video serves as a great starting point for what the rest of the website will contain.

RESTful API Documentation

Our RESTful API documentation can be found here. The API will have endpoints to get ALL instances of disabilities, things to do, and resources stored in the database. Additionally, our API will have endpoints to get specific instances of disabilities, things to do, and resources based on various parameters.

For GET requests, we have not implemented authentication as there is no sensitive information stored in our database. However, for any PUT, POST, DEL requests, we will require higher privileges and keep them internal to the team of developers.

Here are the endpoints we created:
GET disability/all: Returns a list of all disabilities stored in the database.
GET locations/all: Returns a list of all things to do stored in the database.
GET resources/all: Returns a list of all resources stored in the database.
GET disability/<id>: Returns a specific instance of a disability, identified by its id number.
GET locations/<id>: Returns a specific instance of a thing to do, identified by its id number.
GET resources/<id>: Returns a specific instance of a resource, identified by its id number.

Data Sources

RestfulAPI (YouTube): https://developers.google.com/youtube/v3
Google Search: https://developers.google.com/custom-search/v1/overview
Gemini: https://ai.google.dev/

Info on Disabilities: https://askjan.org/a-to-z.cfm
Info on Things to Do / Locations:
https://directory.dmagazine.com/search/?sections=Attractions&features=Pet+Friendly
Info on Resources: https://udservices.org/adaptive-devices-people-disabilities/

Models

Our project is utilizing three models: disabilities, things to do, and resources. Each model has at least 5 attributes associated with it. The disabilities model will contain the name of the disability, the category of the disability, causes, population statistics, and a list of famous people with such disability. The things to do model will contain the name of the place, its address, its zip code, disabilities it is accessible for, rating from google, and the average cost. Finally, the resources model will contain the name of the resource, disabilities it applies to, how to obtain it, how to use it, and its average price range. More information on each of the models (sorting and filtering options, connections to other models, and rich media features) can be found here.

In order to get the data for these attributes, we will scrape information from a series of API's. These scripts will upload data to our database after processing them. Then, we can access the information from our database and use it to regularly update our instances on our webpage.

Tools

We are utilizing the following tools in our project so far:

AWS Amplify
- Allowed us to easily build, ship, and host our full-stack website

GitLab
- Allowed us to store files related to full-stack website in a repository, and allowed us to plan out out project using the issue tracker and milestones

React
- Allowed us to build an interactive UI for our website dynamically

Bootstrap
- Provided us with a framework and design templates for our website's UI

Zoom
- Allowed us to meet remotely to work on different aspects of the project

VS Code
- Used as our IDE for support with coding, debugging, and Git

Postman
- Allowed us to plan and document our API

Flask
- Micro web framework for Python that allows building web applications

SQLAlchemy
- Object-Relational Mapping library for Python that facilitates interaction with SQL databases

NameCheap
- NameCheap is a domain registrar and web hosting company that allowed us to get our domain name.

AWS EC2
- A web service that provides resizable compute capacity in the cloud.

Selenium

- Used for automated testing of web applications, which allowed us to simulate user interactions and perform functional testing across various browsers.

Hosting

We are hosting the website at https://www.inclusiability.me/ via AWS Amplify. AWS does the majority of the cloud hosting for this project. This includes things such as IAM Identity Center, Route 53, and Amplify. The IAM Identity Center is used to maintain user accounts to manage the AWS Account. Our domain was registered through Route 53. Amplify connects our GitLab source repo to run the web application.

Pagination

We implemented pagination in the backend folder by putting a limit of 20 instances per page. We essentially split up the instances into groups of 20, and display each group one page at a time. We also incorporated this feature when a customer searches a specific word. The instances are paginated then as well, ensuring consistency.

Search / Sorting / Filtering

In Phase 3, we implemented the ability to search, sort, and filter based on different features for each of the three models - disabilities, events, and resources. Dropdown menu components were created for selecting filter and sort options on each of the model pages. The code associated with these dropdown menus and the logic associated with handling the selection is all contained in the back-end folder, with important files being common_funcs.py and all the files within routers, such as disabilities.py, locations.py, resources.py. These files contain the specific implementations of how these functionalities work on the back end for each of the models. Then, on the front-end side, we added the dropdowns and buttons so a user is able to perform these searches. The code for these is located within the front-end folder, with the most important classes being those inside the pages and components folders. The class front-end/src/data/ModellData.js shows all the options for all the different drop down menus.

Benefits:

- Improved User Experience: Users can find the information they need more quickly and easily.
- Increased Efficiency: Filtering allows users to focus on relevant data, saving time.
- Flexibility: The system can be adapted to accommodate future models with specific filtering needs.

Challenges (Phase 1)

- The first challenge we faced was general communication. Figuring out a time and place to meet and work for all 5 of the team members proved to be harder than expected. This made it harder to find time to meet up with my group and get things started efficiently. However, we figured out times nonetheless and met up virtually when needed so we could all work together.
- Another challenge we faced was having to learn React for the front end. Most of us had little to no experience with React, so we had to spend extra time really learning how it works.
- A third challenge we faced was with using the Postman interface. Designing the interface meant we needed to very carefully plan every possible parameter and return value we would be getting in the future. It was also hard to figure out how to publish the documentation to be able to view it in a documenter.getpostman. However, with researching through help articles on Postman, we were able to figure it out.
- A final challenge we faced was with the entire set up process, but more specifically with Amplify and getting the Namecheap linked to it properly. However, two team members worked closely through it, and after researching how to connect, were able to figure it out.

Challenges (Phase 2)

- The first challenge we faced was the whole setting up of our backend with Flask. None of us had used Flask before, so we had to self-learn a lot and debug continuously to figure out how to set all the connections to the database up.
- Another challenge we faced was general web scraping. Each website had a very different structure for how to scrape, so writing the code for each website was very personalized and took a lot of time. Even after we would scrape the websites, a lot of information would still be missing, causing us problems.
- Finally, connecting our backend to the frontend, in terms of hosting, was a real struggle. We strictly followed the tutorial; however, we still ran into many bugs throughout the way, which caused us to struggle through the process. After we got it connected though, it was a lot smoother.

Challenges (Phase 3)

- The first challenge we faced was having to refactor and reuse our existing code across the website. Implementing searching was challenging at first as we needed to figure out how exactly we wanted everything to work together
- Another challenge we faced was updating our user-interface to incorporate the new buttons and dropdowns. We wanted to keep a professional, simple colors look on our website, so going back and forth between different colors, fonts, and styles took some time and deliberations to figure out.
- Lastly, another challenge faced was in general time management. With final rounds of midterms coming at the same time as this phase was due, as well as the unique nature of this phase, we found ourselves struggling to find time to work on the project. As a result, we had to take some slip days to complete it, but we wanted to ensure that what we submitted was of good quality.