CS373 Group 22 - Foster Hope Phase 4 Technical Report

Grace Pan, Maadhav Kothuri, Nathan Cheng, Raymond Wang, Alea Nablan

Purpose

The purpose of this project is to bring awareness about Foster Care within Texas and provide various resources to those that are in the system. Children in foster care can be especially vulnerable, so supporting them by providing them and their caregivers with useful information is very important. To do this, the project will provide information about the Texas counties, Foster Care organizations and Foster Care resources within Texas.

Stages

Phase 1

In this phase, we built the initial front-end of our application. This included determining our use case, our models, and our instance attributes. We built the general architecture of our model and a basic user interface using limited instances and hard-coded data.

Phase 2

In this phase, we built the back-end of our application. This included building an API, setting up back-end hosting and creating our database. We also used REST APIs from various data sources and scraped the internet to populate our database with real data. Using this data, we dynamically created instance cards on our model pages, as well as a specific page for every instance. Lastly, we cleaned up our user interface and implemented pagination for each of our model pages.

Phase 3

In this phase we implemented searching and sorting for each model and the overall website. This included updating our APIs to implement a search query and the ability to sort the instances based on their five attributes. We also created search bars for each model page and added a search bar on the landing page. Then for each model page, we implemented a dropdown menu to choose which attribute to sort the instances by and a toggle to show the instances in descending or ascending order.

Phase 4

In this phase we created visualizations of our data using our REST API and placed them on a separate page on our website. We also added critiques of our website on that same page.

Similarly, we utilized the REST API of our developer team to create visualizations of *their* data, and provided critiques of their website on a separate page of our website. Finally, we refactored our code and created a Pecha Kucha presentation to provide information about foster children in Texas.

User Stories

Phase 1
General Information
Add description for foster hope on splash page

This is an important feature since the splash page is the first page that users will see. It will also help users understand what the website is about. As a result, we addressed this user story by adding the description to the model page and adding more styling to enhance the visual appeal.

Add models on organization page
We added our model's instances onto the organization page in the card format. This way, users can get the information needed about the organizations supporting foster care. However, we hard-coded the values for this phase and will programmatically populate the instances in the next phase.

Visuals
Add images for about and splash page
We addressed this user story by adding images of our tools to the about page and other images as a carousel to the splash page, hoping to increase the visual appeal.

Include a photo and bio for each member of your team in the "About" page
Being a customer of your website, it would be helpful to be able to put faces to names

At the time of receiving this user story, we did not have any images on our splash page or on the about page so we can see how it can be helpful to users as images can provide a more vibrant experience when looking at the website.

Use Bootstrap, Tailwind, or another framework
We addressed this user story by using React-Bootstrap, which provides pre-made React components to simplify the creation of user interfaces. We used many components from this framework, such as rows, containers, columns, carousels, and a navbar.

Phase 2
Availability
Site Down? It seems that at the moment, your website is not available.
At the time of this user story, we were shifting DNS services from Namecheap to AWS Route53, which caused our site to be down. We addressed this story by ensuring that our site migrated correctly and that it was available again using the proper AWS services.

Visuals

Make Model Instance Card Sizes Consistent

When we received this story, our card sizes on our model pages varied widely due to differences in amount of information and image size. Since this is very jarring from user experience standpoint, we made all of our cards the same size to ensure that they look much cleaner.

Add Related Instances as Cards to Each Instance Page

At first we had links as related instances on each instance page but now, we have a card template for each model that we can use to display on each instance page. This makes the related instances nicer to see for each instance. We also labeled to show what model the cards fall under.

Add a Logo/Favicon

We added our logo to the navbar and also have it come up in the tab of our website as well. Our logo shows a smaller hand and a bigger hand coming together. This represents our website as we are working to give a hand to children in foster care while also bringing awareness to others about foster care in Texas.

Get Better Quality Images and More Related Images for Slideshow

To address this user story, we added much higher resolution images to the carousel on our splash page and made sure that no text was cut off in the images. This helped make the site as a whole look cleaner and more professional.

Phase 3

Documentation

Postman link on About Page

We added a link to our Postman page for documentation of our API on the about page.

GitLab Repo link on About Page

We also added a link to our GitLab repo on the about page under the tools that we used section.

Visuals

Fix Foster Hope Navbar Icon Bug

Our icon in our navbar was going away whenever a user navigated to an instance page. This was due to an incorrect image path, so we were able to fix this problem.

Splash Page down arrow button

Our users asked for a down arrow on the splash page so that it was clear that there was more content on that page. We implemented this so that the arrow persists until the user hits the bottom of the page, at which point it disappears.

Move Number of Instances Label to Top of Model Pages
Before getting this user story, the number of instances for each model was displayed at the bottom where the pagination was. Now it was moved to the top of each page below the search bar. We think that this makes it easier for people to see how many total instances.

Phase 4
Visuals
On Instance Page, give the area with title and location a background color to increase visual interest.
Our background for this area was a plain white, so we changed it to the light blue color that is in the background of the cards on the About Us page.

Bold attributes on the Model Cards
Our attributes were getting lost on our model cards, so we bolded them to make them more visible.

Center Text on About Page to Improve Style Consistency
We made sure to center the paragraphs at the top of the page to conform to the style that our titles were in.

Increase Text on Splash Page to Improve Readability
We increased the size of our title, as well as the size of the information paragraphs, to complete this user story.

Make the Gitlab and Postman Buttons Stand Out More
The links to our Gitlab and Postman documentation were clickable, but users couldn't tell until they hovered over it. As a result, we changed the color of the text and bolded it so that it was clear that it was clickable.

Tools

AWS Amplify: Used for hosting our website
GitLab: Allowed us to store files needed for the full-stack website in a repository. We also used the issues tracker to plan our project and divide up work.
Postman: Used to plan and document our API
React.js and Next.js: Used for our front end of the website and create an interactive UI
React-Bootstrap: Has design templates for our website's UI
Flask: Used to create API endpoints for our custom REST API
AWS EC2 and Route53: Used to host our backend server and hold DNS records for the API
Google APIs: Used to help scrape data for our database and get various media for our instances
Supabase: Used to store the scraped data we found for each model

Selenium: Used for GUI tests and web-scraping
BeautifulSoup: Used for web-scraping
D3: Used for data visualizations

Data Sources

Google Custom Search API:

- Resources and associated attributes
- Images for models

Youtube API:

- Videos for county instance pages

ChatGPT API:

- Descriptions for counties and organizations

Google Maps API:

- Organizations and associated attributes
- Embedded links to Google Maps for organizations

Beautiful Soup Library:

- Used to scrape for resource events on eventbrite

Frontend Architecture

Our frontend is a Next.js project and is hosted by AWS Amplify. Our current project has the landing page, an about page and 3 model pages where each model page has instance pages. To install the project: Download node.js and then clone our repository. Then in the terminal:

1. Move into the front-end directory
2. Run **npm install** to install the needed packages
3. Then to run the application on a local server, run **npm run dev**

Backend Architecture

For the backend, we used Flask in order to define our API routes and connect to our app folder. We then used SQLAlchemy to query into our database which will be sent to Flask and then sent to our frontend to be used. Using the endpoints described in our API documentation, we can get the data from our database.

File System

Each of the main pages (about and model pages) have their own folder within src/app:

cs373-group-22/front-end/src/app/about
cs373-group-22/front-end//src/app/counties
cs373-group-22/front-end/src/app/organizations
cs373-group-22/front-end/src/app/resources

Currently in each model page, there is a page to show all the cards under the model. Then for each instance in each model, there is a folder called [name] that has the template for the instance page.

cs373-group-22/front-end/src/app/[model]/page.jsx
cs373-group-22/front-end/src/app/[model]/[name]/page.jsx

Each model also has a component folder to hold the model's card template.

cs373-group-22/front-end/src/app/[model]/component/[model-card].js

Our components for the splash page, such as the Navigation Bar, will be in the components folder.

cs373-group-22/front-end/src/app/components

Components for other pages will be in their respective folders. For example, any components used in the about page will be in *cs373-group-22/front-end/src/app/about/components.*

Our frontend acceptance and unit tests, along with our API tests are in the tests folder. There will be the Jest tests, Selenium tests and our API tests.

cs373-group-22/front-end/tests/api_tests
cs373-group-22/front-end/tests/jest_tests
cs373-group-22/front-end/tests/selenium_tests

Photos are under the public/images folder.

cs373-group-22/front-end/public/images

We also have a *page.module.css* to style our pages uniformly and a *globals.css.* It is important to note that we often utilize the .main and .description styles located within *page.module.css*.

Models

The models we will use for this project are going to be counties, organizations and resources. The attributes for counties will be the name of the county, population, the number of agencies, number of foster children and the number of foster homes.
For organizations, the attributes are their name, location, the type of organization, a review of the organization and the hours of operation.
For resources, the attributes are the name of the resource, location, type of resource, phone number and the website.

API Documentation

Our documentation can be found on Postman here. This will have our endpoints in order to GET information on the number of foster homes, number of foster kids, the organization reviews, and

events. This information is returned with each instance requested through the API. Using the API, we can request all instances of a certain model or a specific instance of a certain model.

Hosting

We are using AWS Amplify to host our front-end website at www.foster-hope.com. Our site is deployed from the “main” branch of our repository so that every time that we push to that branch, Amplify builds and deploys the updated version.

Additionally, we are using AWS EC2 and RDS to host our backend server, which is running the endpoints for API. Although our domain, foster-hope.com, was bought from Namecheap, we are using AWS Route 53 as the DNS service for our backend (api.foster-hope.com). This allowed us to use an EC2 instance and an elastic load balancer to support HTTPS connections to our API.

Challenges

Phase 1

AWS Amplify

We had some problems while using AWS Amplify as it gave us some errors but we were able to fix the main issue by fixing our imports. When we want to import a component, if we need to import more than one, we had to put curly braces around it. We also had issues with our RootLayout folder having incorrect fields, which hid many errors and took a lot of time to diagnose and fix, since the error was only being shown by AWS Amplify. It is also important to note that we had to use a custom AWS image to host AWS Amplify with Next.js (Amazon Linux 2023).

Next.Js

Rather than using React.js, we wanted to try out Next.js. Since most of us are not familiar with next.js, it was a challenge to figure out how it worked and how Next.js would interact with the other tools we were using. Using the documentation helped us considerably, since they had a lot of examples with code that we could directly reference and learn from.

Project Plan

We were having trouble in the beginning when trying to figure out what group of people we wanted our site to be about. We came up with several ideas at first and looked to see if we could find enough information for each topic. In the end, we decided on providing information about foster care and to help foster kids.

Phase 2

Backend Hosting

Backend hosting was a large challenge for this phase. Even though a tutorial was published, it was difficult understanding how transferring DNS services from Namecheap to Route53 could be done properly. Since our site would go down whenever this was done incorrectly, it also increased the pressure to finish this part quickly, since no work could be done otherwise. The main issue was that when we got the backend working in Route 53, the main website would go down. By learning about how these services worked and interacted with each other, we realized that we had to add the previous CNAME record for our main site that had been in Namecheap, which solved our problem.

Dynamic Routing

Dynamic routing was also difficult in certain areas due to the tools we were using. Since we were using Next.js, there were certain actions that could only be performed in server-side components, and certain actions that could only be performed in client-side components. This made it difficult when using state and asynchronous functions in our components, so we had to be very careful with how we designed our front-end structure. Whenever we had this kind of problem, we found that we usually could create a new, nested component that encapsulated the functionality that we needed to solve the problem.

Data Collection

We also struggled with certain areas of data collection, specifically the requirement that each instance have a lot of text on its instance page. To meet this, we tried to find descriptions for each instance, but web scraping for this information proved to be very difficult and time-consuming. In order to mitigate this, we utilized the ChatGPT API to collect the information for us and provide it in a usable way. This greatly simplified this task and saved us a lot of time and effort in web scraping.

Phase 3

Implementing Searching

It was difficult at first to get the API calls to work correctly for searching, largely because we had to figure out whether to perform searching using the SQL alchemy functions or the SQL query itself. Also, since we are using Next.js, there are certain ways that we have to use state and async functions that made implementing searching on the frontend more complicated. However, by thinking carefully about all of these moving parts, we were able to implement searching successfully.

Fixing the Navbar Logo

Although this was a small part of our work for this phase, this problem was definitely frustrating. One of our user stories wanted us to fix a bug where the logo in the navbar would disappear in an instance page. We looked carefully through the code, trying multiple different ways to structure the code to see what was going wrong. However, it turned out that a missing “/” from our pathname was the issue all along. Little problems like these can frustrating since they can take up more time than necessary, so designing things carefully and paying attention the details is very important.

Creating a New Page for Global Search with Tabination

For our global search, we have a search bar on our splash page that, when used, takes us to a new page with the search results for all models. To do so, we had to make sure that we properly routed the user and sent all of the necessary data as well. Furthermore, we had to implement tabination to show search results for each of our models. We needed to read up on the React documentation, since we were using the React-Bootstrap tabination component, so we needed to design everything properly to achieve a functional and clean user interface.

Phase 4

Visualizing Data Using D3

Although this phase was quite straightforward, understanding how to use D3 definitely took some getting used to. The way that the D3 website is presented, it’s very difficult to understand how the tool works and how to create the charts that we needed. However, we found a website with better explanations and documentation for creating the correct graphics, which helped us substantially. We used this site as a reference to create our data visualizations and the visualizations for our developer team.