Ukraine Crisis: Phase II

IDB Group 6
Chuma Anigbou, Derek Chen, Samuel Osibamowo, Akhilesh Bitla, Alex Jimenez

Our Mission:

Ukraine Crisis is a platform dedicated to raising awareness among those affected by the ongoing situation in Ukraine. Our goal is to provide users with the opportunity to explore stories of numerous refugees, discover resources and support groups that raise awareness or funds for the cause, and keep them updated on the current events unfolding in Ukraine

User Stories:

Story 1 : Format the phone numbers the same way for all the support groups

During this phase, we successfully implemented the user story by scraping a website that had all of the phone numbers formatted the same way and if they weren't we adjusted for that.

Story 2 : Make the links (for example the navbar links) change the mouse cursor into a different cursor type to indicate that it is clickable (right now it is showing the text editing cursor instead of the pointer finger cursor when you hover over anything that is clickable).

We've successfully resolved the issue! Now, when users hover over any clickable element, they see the pointer finger cursor, which replaces the default text editing cursor. This change clearly indicates to users that an element is interactive and can be clicked on, enhancing the user interface by making it more intuitive.

Story 3 : When clicking on certain instances such as "Russia launches barrage of 45 drones over Ukraine as Kyiv changes more military leaders" and "Two years of war: Ukrainian refugees face lasting exile" under the News and Media models, the cards that link to other instances have the read more button cut off. Make the formatting consistent for these cards.

This issue was due to how we created the card instances that displayed on the connections section of the page. We were able to fix this once the information was not hard coded and instead was dynamically fetched from the backend.

Story 4 : For the cards with pictures, crop them properly or format the pictures differently so there isn't white space in the top corners of each card.

While this was by design initially, we made the change to reduce the amount of white space that was on each of the cards. We did this by adjusting the card's image generation.

Story 5 : Can you please increase the number of instances per page?

This user story was one we thought was necessary to implement to decrease the total number of pages we had. Initially we had only two instances per page which caused it to be quite tiresome.

Api Documentation:

Our RESTful API documentation can be found at this link. This API includes endpoints to retrieve all instances of news and media, asylum countries, and support groups. Furthermore, it offers endpoints for obtaining specific instances of each model. Additionally, the API will provide endpoints for retrieving connections between instances and a dedicated endpoint for searching all models based on specific parameters.

Models:

Our project utilizes three models: Asylum Countries,news and media and support groups . Each model has five attributes associated with each individual instance of the sed model.

Model 1: Asylum Countries

Instances -

Our first model contains many instances of countries that accept Ukrainian Refugees . These specific instance cards all contain a picture of the country's flag as well as a map of the country as the two forms of media. In terms of attributes, the country cards contain: names, population,

capital,region, and languages. These cards are connected to related news and media, and support groups.

Model 2: News and Media

Our second model contains many instances of related news and media from the ongoing crisis. These specific instance cards all contain a picture or video relating to the news , as well as a map of the related city or country as the two forms of media. In terms of attributes, the news cards contain: author, description,name, date and a title/caption. These cards are connected to related resource groups and asylums.

Model 3: Support Groups

Our third model contains many instances of related support groups to the crisis. These specific instance cards all contain a picture of the support groups logo and an accompanying video for each support group as the two forms of media. In terms of attributes, the support cards contain: location based in, Website, name, rating and contact information. These cards are connected to related locations/countries and recent news or media.

Hosting

We are currently hosting our site at ukrainecrisis.me using AWS Simplify. We are hosting our backend at cs373-backend.ukrainecrisis.me using AWS Old Balancer, route 53 and the EC2 instance.

Scraping:

For the web scraping, we employed a mix of using website APIs as well as manual scraping using Beautiful Soup, enhancing our data extraction capabilities significantly. By integrating Selenium for automated browser interactions, we were able to navigate through and interact with web pages dynamically, which is crucial for pages that load content based on user actions or rely on JavaScript and AJAX. Beautiful Soup's powerful HTML parsing capabilities complemented this setup by facilitating the extraction and processing of information from static parts of the web page once they were made accessible by Selenium's automation. The use of these manual scraping tools were used to grab the information needed for the Support Groups attributes. These include phone Name, Contact, Website, Rating, and Location. For obtaining

our data for news and Asylum countries, we utilized two different API which grabbed 163 instances for news and 100 instances for asylum countries.

Database Set Up:

For our database, we utilized a MySQL Community Server to initialize the database, MySQL Workbench to view and debug the data and the tables from the database, and AWS RDS to deploy the database on the cloud. Initially, we created the MySQL database on one of our systems locally to ensure that we were able to interact with it efficiently. We wrote Flask code to create our tables which synced with the database. We also utilized a Flask package called Flask-migrate that enables efficient interaction with the database to migrate different changes from the data itself to creating new columns in a table or a new table itself. Finally, we created a new database instance on RDS and dumped our local database into an SQL file which we imported into RDS. This way, we have a running RDS server that presently stores our data in the cloud.

Architecture:

Our code is well-organized, with different folders corresponding to the tabs or models they belong to. Each model page has its dedicated folder within the 'pages' directory. Within each mode’s javascript file, pagination was implemented and contains a “Next” and “Previous” button Inside these dedicated folders, there is a single [id].js file that houses the actual cards for each instance, accessible by clicking the 'read more' button within their respective models.

The id.js file within the 'asylum-countries”' and 'news-and-media' directories also contains our calls to the Google Maps API. The code relevant to our splash page is located in the index.js file.

For the 'about us' page, the code resides in the about-us.js file. Here, you can find all the displayed information, along with calls to the GitLab API. This API calls automate the updating of statistics displayed for each team member.

The code used for scraping our data can all be found within the scripts folder which is located in the backend directory. In this folder, each model has its own respective Python file for scraping. Also in this backend directory is our REST API as well as our DockerFile and requirements.txt file which contains all the packages we've used up to this point.

Tools:

During this phase, we are leveraging several tools to streamline our project development process:

- **AWS Amplify**: Facilitating easy development, deployment, and hosting of our full-stack website.
- **GitLab**: Serving as our repository for storing project files, and facilitating project planning through its issue tracker and milestones.
- **Next.js**: Empowering us to create an interactive user interface efficiently.
- **Tailwind**: Offering design templates and a framework for crafting our website's UI.
- **Node**: Enabled us to build a scalable website capable of handling multiple concurrent requests.
- **Zoom**: Facilitating remote meetings for discussing various project aspects.
- **Ed Discussion**: Providing a platform for project-related communication via messaging.
- **VS Code**: Serving as our integrated development environment (IDE) for debugging and version control with Git.
- **Postman**: Enabling us to effectively plan, document, and test our API endpoints.
- **Name Cheap :** Allowing us to own temporary rights to our domain name
- **MySQL Server:** Enabled us to create our database and run it locally on our system
- **MySQL Workbench:** Enabled us to interact with the database efficiently. We used it primarily to ensure our data was migrated correctly and ensured our columns were properly defined. It was primarily for debugging purposes.
- **AWS RDS:** Enabled us to import our local database into an RDS database instance which is stored in the cloud.
- **Flask Migrate:** Enabled us to make database migrations through Flask code, which means we were able to efficiently update different aspects of our database when necessary.

Challenges:

Challenge 1: Hosting our API

We had trouble hosting the domain for the backend. We had to make many changes like creating route 53 zones and making a custom DNS within nameCheap where we could put those name spaces. We had trouble creating the certificates because at first we weren't aware of the necessity for route 53.

Challenge 2: Changing Models

One of the significant challenges faced during this phase was the shift from our initial model of Refugee Testimonials to Asylum Countries. This transition was prompted by a shortage of available resources. Initially, we overestimated the feasibility of acquiring 50 refugee testimonials from only a few websites. After conducting further research, we recognized the necessity to pivot in order to successfully complete this project.

Challenge 3: CORS errors

Another significant challenge we faced was not even from our own doing, but rather, something we lacked. The Cross-Origin Resource Sharing (CORS) standard allows or disallows certain requests to be made across servers, and serves as a good security measure for APIs. However, that security worked against our favor because we wanted to be able to make calls to our rest API to query database records, and they were being blocked by this policy. We fixed this by learning the flask_cors library that allowed us to configure not the requests we were sending, but the server itself, to allow these requests.