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: Could you please update your GitLab repo to assign issues to people correctly, the total and individual issues seem to be off.

This was a simple fix because it was simply because we were just collectively closing all issues without first assigning them to someone. We would split off tasks appropriately but failed to document that in our git repository.

Story 2: Could you please bold the sorting options for visibility?
This was a very minor fix, all we had to do in order to bold the sorting options is put a <b> tag when writing out the labels for the sorting Options.

Story 3: Could you please bold the filtering options for visibility?
Similarly to story 2 this was a very minor fix, all we had to do in order to bold the filtering options was put a <b> tag when writing out the labels for the Filtering Options.

Story 4: It would be helpful if the filtering options were in alphabetical order so that it is easier to find specific options.
We just had to fix the order that which we were displaying the filtering Options, so we organized them to have them in alphabetical order, with the exception being “Other” which was placed as the last option similarly to other websites.

Story 5: It would be helpful if the sorting options were in alphabetical order so that it is easier to find specific options.
Once again similar to story 4, we just had to organize how we were displaying these sort options, and we fixed them to be shown by alphabetical order as well, with it being auto-selected to default which is the top, which is typically how it is in other websites.

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 instance of the used 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 and a map of the country as the two forms of media. In terms of attributes, the country cards contain the following: names, population, capital, region, and languages. These cards are connected to related news/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 the following: author, description, name, date, and 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 group's logo and an accompanying video for each support group as the two forms of media. In terms of attributes, the support cards contain the following: 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 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 was used to grab the information needed for the Support Group attributes. These include phone Name, Contact, Website, Rating, and Location. For obtaining our data for news and Asylum countries, we utilized two different APIs which grabbed 163 instances for news and 100 instances for asylum countries.

Searching:

For searching, we had to create a searchAPI file where we implemented our search logic. We simply implemented a way to pass in a query, or what a user will search, and use the filter feature and the like command from mySQL to check if our data matches with the words of the query. If there is a match, we create an array of model blocks where each block holds all the field data from the model. Then in our front end, we implemented a way to utilize axios to pass in a search query from the Search Text field to the server to pull the filtered data back to the front end. Then we use the map function to render the data as cards. This is implemented in the same way for all 3 models: news, asylum countries, and support groups. The cards rendered from the search for each model are also filterable and sortable. Moreover, the cards are paginated such that there are only 6 per page and we implemented a next and previous button to traverse through a set of cards. This makes it easier to go through the searched information and enables the user to find what they need.

Moreover, we also implemented our global search. For the global search, we reused all 3 model searches and combined them to perform the same task. In our front end for global search, we utilized tabs to divide each model's responses to make it easier for the user to search for what they're looking for. This way we render the searched data of all models in a structured manner for the user.

Filtering:

We decided to place our filtering algorithm within the "Get All Instances" for each model. For our filtering algorithm, we allowed users to select two specific attributes as well as different specific examples of those attributes. If the specific example wasn't a checkbox, it was categorized under the other section. The checkbox drop-down menus were implemented in the front end. These choices were then passed to the backend and the sorting was handled there. You can find the filtering code within the modelsAPI.py file and the check box code in news.js, support-groups.js, and asylum-countries.js.

Sorting:

Similarly to filtering, we also decided to implement our sorting algorithm within the "Get All Instances" for each model. For our sorting algorithm, we allowed users to select an attribute and method of sorting through two drop-down menus on the front end. These choices were then passed to the backend and the sorting was handled there. You can find the sorting code within

the modelsAPI.py file and the drop-down menu code in news.js, support-groups.js, and asylum-countries.js.

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.

All of our Phase 3 code was added to the corresponding page javascript files (support-groups.js, news.js, asylum-countries.js), and they get all endpoints for each model in the modelsApi.py file. For this phase, we also created a new page called search.js which was for our global search.

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: Adjusting Filtering Algorithm for Support Groups

We ran into some issues with our code for filtering our support groups. Originally, we intended to use the same formatting/code from the previous two models to accomplish this task but we shortly realized that this would cause issues because this information was scraped rather than produced directly from an API like the news and country models were. We were eventually able to solve this issue by accounting for different instances of the same information for example if there was an extra space between a comma and a character.

Challenge 2: Fixing Sorting and Filtering on Searched Instances

Because we implemented sorting and filtering of instances before searching, we ran into an issue where the search results were not able to be sorted or filtered. This was a challenge that took some brainstorming to solve. We were eventually able to resolve this issue by adding the filter and sort algorithms to where the search instances were populated. This allowed us to search and still be able to use our filtering and sorting options.