

Table of Contents

[Introduction](#_heading=h.gjdgxs) **2**

[Project’s Mission](#_heading=h.30j0zll) 2

[Analysis of User Needs](#_heading=h.1fob9te) 2

[User Flow](#_heading=h.3znysh7) 2

[**Planning**](#_heading=h.9hvn9f9uv5qq) **3**

[Software Requirements](#_heading=h.tyjcwt) 3

[Roadmap](#_heading=h.3dy6vkm) 9

[Design](#_heading=h.1t3h5sf) **10**

[Software Stack](#_heading=h.4d34og8) 10

[Software Ecosystem Map](#_heading=h.2s8eyo1) 12

[Database Design](#_heading=h.b2spqjksyasv) 13

[**Implementation**](#_heading=h.vji3y1ov0jj3) **14**

[Code Design](#_heading=h.lnxbz9) 14

[Algorithm](#_heading=h.35nkun2) 15

[UI/UX Design](#_heading=h.1ksv4uv) 15

[Performance Enhancements](#_heading=h.44sinio) 18

[Release & Deployment](#_heading=h.2jxsxqh) 18

[Conclusion](#_heading=h.z337ya) **18**

[Final Project Impressions](#_heading=h.3j2qqm3) 18

[Obstacles and Triumphs](#_heading=h.1y810tw) 19

[Next Steps](#_heading=h.4i7ojhp) 19

[Software Maintenance](#_heading=h.2xcytpi) 19

[Overall Thoughts](#_heading=h.1ci93xb) 19

# Introduction

## Project’s Mission

To develop a web application that can assist in the effort of creating an inclusive workplace environment. The web application will have two main features: The first feature is an inclusive language sentiment analyzer. And the second feature is a data acquisition and labeling interface. These features go hand in hand, as the data acquisition and labeling tool will contribute to the development of a more robust dataset for training the model in the first feature.

## Analysis of User Needs

Studies show that DEI supports growth and innovation in the workplace (Chaudhry, Paquibut and Tunio, 2021). One way to help promote and increase DEI in the workplace is through newsletters/emails and special observances. For these two tasks it is typical to generate informational artifacts with content that can unknowingly relay the wrong sentiment. One way to avoid this situation is by employing editors and legal advisors to vet and approve content to ensure they align with company creed prior to dissemination, though this can be time consuming and costly. Time and cost can potentially be reduced by using a machine learning model trained to distinguish inclusive text as having either a negative or positive sentiment. While there are many models that perform sentiment analysis the publicly available models use datasets with ostensibly explicit language, thereby making the model inefficient for use on inclusive language.

## User Flow

The typical user will first navigate to the [DEILA](https://deila.azurewebsites.net/) website hosted using Azure web services. From there they will see the homepage which includes content describing the website and its features. For feature one the user will navigate to the Sentiment Analysis tab and once on this page will copy and paste whatever text they want to have classified. After submitting the text, the page will report it as being either positive or negative. Feature two is located on the Dataset page which can be navigated to by clicking on the dataset tab in the main menu. On the dataset page will be a table listing the current contents of the dataset used to train the model for the first feature. A user can create, edit, update, and delete new records for the dataset on this page.

# 

# Planning

## Software Requirements

High Level Requirements 1

**1.1**

Users should be able to use the web application interface to use the machine learning model to perform sentiment analysis on inputted English text.

**1.2**

Users should be able to access the latest version of the diversity equity and inclusion dataset using the web application interface and perform CRUD operations on it.

**1.3**

Deploy and host the DEILA website (frontend and backend) using Azure Compute by August 26th, 2022.

**1.4**

Deploy the inclusive language sentiment analyzer using Azure Compute or HuggyFace by August 26th, 2022.

Main Menu Functional Requirements 2

**2.1**

The main menu should have four options: Dashboard, Language Analyzer, Dataset Manager and Collapse.

**2.2**

When the Dashboard option is selected the user should be routed to a page where the subsequent actions are described in the Dashboard Functional Requirements Section.

**2.3**

When the Language Analyzer option is selected the user should be routed to a page where the subsequent actions are described in the Language Analyzer Functional Requirements Section.

**2.4**

When the Manage Dataset option is selected the user should be routed to a page where the subsequent actions are described in the Manage Dataset Functional Requirements Section.

**2.5**

When the collapse option is selected the main menu should collapse and free up space on the main content area. The main menu will still be visible, and the four options will still be accessible but instead of having a text title you should only see the icons associated with each option.

Dashboard Functional Requirements 3

**3.1**

The main content area of the dashboard page should include a description of the project mission.

**3.2**

The main content area of the dashboard page should include a description and link to the following related websites; the inclusive coding festival main page, the URL to the huggyface dataset and the GitHub URL for the code source of this project.

**3.3**

The main content area of the dashboard page should also have a quick about me section.

Language Analyzer Functional Requirements 4

**4.1**

The main content area of the language analyzer page should have a brief description of how to submit content to be analyzed.

**4.2**

The main content area of the language analyzer page should have a text box area that will allow users to copy and paste text for analysis. The text box area should have basic text editor features such as bolding and underlining words.

**4.3**

The main content area of the language analyzer page should have a submit and cancel button. When the user clicks submit button, the text will be sent to the service where the inclusive language analyzer was deployed to, and a sentiment classification will be returned. When the users click the cancel button the text in the textbox area will be cleared.

**4.4**

The main content area of the language analyzer page should have a results section that will display either positive or negative dependent on the model classification. The result should be highly visible and visually pleasing.

**4.5**

The language analyzer should have a visible notification to update the user on the model classification service whether it is processing or complete.

Dataset Manager Functional Requirements 5

**5.1**

The main content area of the Dataset Manager page should display a paginated table preset to view 10 rows and the table will have the following five columns Title, Content, Bases, URL and Sentiment. The table should also be sortable by category and sentiment.

**5.2**

The table in the Dataset Manager should allow you to change the number of rows displayed to either 10, 25, 50 and 100.

**5.3**

Each data entry row of the dataset table should have an edit action button. The edit link will send you to an entry specific page that has editable fields for the title bases sentiment and content of the currently selected data entry.

**5.4**

The edit entry page of the Dataset Manager should have four fields, Title, Content, Bases and Gender. The Title and Content areas should an input and text box respectively. The bases and gender fields should be dropdowns. Each field should be prepopulated with the current value of the field in the database.

**5.5**

The edit entry page of the Dataset Manager should have three buttons, Cancel, Reset and Save. Cancel will return the user to the main Dataset Manager page. Reset will return the fields to their original values and Save will persist the changes to the database.

**5.6**

Each data entry row of the dataset table should have a delete action button. When the user clicks the delete button it will remove the entry from the database.

**5.7**

Each data entry row of the dataset table should have an expandable section to hide and display the main content of the article.

**5.8**

The main content area of the Dataset Manager page should have a highly visible button that says add article that when clicked will route the user to a sperate page the editable fields needed for a user to submit a new article to the diversity equity and inclusion dataset.

**5.9**

The Add article page of the Dataset Manager will allow users to submit a new article to the diversity equity and inclusion dataset by providing a URL and associating a basis and sentiment for the article from that URL. The rest of the data such as content and title will be scrapped from the provided link.

**5.10**

The Add article page of the Dataset Manager should have three buttons, Cancel, Reset and Submit. Cancel will return the user to the main Dataset Manager page. Reset will clear all the fields and submit will save the inputted article to the database if it is a valid URL if it isn’t a notification will appear notifying the user that the URL could not be scrapped.

**5.11**

All CRUD operations should illicit a visible notification on each page informing the user on the status of the operation and or any errors.

Use Cases

**Actors**

For the following use case specifications there will be two actors the first is a user wanting to have text classified as being negative or positive this will be a model user and the second actor will mainly interface with the dataset this will be a data curator.

| Use Case: Submit Text for Classification |
| --- |
| Primary Actor: model user |
| Secondary Actor: |
| Precondition: Diversity Equity and Inclusion related text has been inputted to the textbox on the Sentiment Analysis page. |
| Postcondition: Message informing the user of the classification given by the gpt2 fine tuned model. There will also be a scaling feature to display how negative or positive the result is based on the logit values. |
| Main Success Scenario:   1. Classify text service provided by DEILA Web API takes in text and returns logit values. 2. Classification message is displayed on the webpage. |
| Alternative Course: |
| Exception Course:   1. Empty or null string sent to classification service. 2. Webpage displayed empty content message. |

| Use Case: Add New Diversity Equity and Inclusion Article to the Dataset Manually |
| --- |
| Primary Actor: data curator |
| Secondary Actor: |
| Precondition: Manual article add is selected in the article dashboard page and information for each field is added to the webpage (title, content, origin, sentiment, basis). |
| Postcondition: New article is added to the diversity equity and inclusion dataset. |
| Main Success Scenario:   1. Post events triggered and serviced by the DEILA Web API, article content is saved as a new entry in the database. 2. Notification to alert users of successful addition to dataset. |
| Alternative Course: |
| Exception Course:   1. Invalid fields inputted i.e., unsupported characters in text, title was too long. 2. Notification to alert users that the request was invalid, and no edits were made to the selected data entry. |

| Use Case: Modify an Existing Dataset Entry |
| --- |
| Primary Actor: data curator |
| Secondary Actor: |
| Precondition: Article exists in dataset and is navigated to by clicking the edit button of the row on the table on the dataset manager page. |
| Postcondition: Article from dataset has new edited content stored in the database and is updated on the DEILA front-end Web App. |
| Main Success Scenario:   1. User selects a field on the edit article page and edits one of the items: Either changes the sentiment or basis dropdown values or the origin, title, and content text fields. 2. The content changes and is visible on the DEILA front-end Web App. 3. User clicks the save button. 4. Put events triggered and serviced by the DEILA Web API. Service validates incoming changes. 5. If valid the article that was selected and edited on the DEILA front-end Web App is modified in the DEILA database. 6. Notification to alert users of successful updates to the dataset. |
| Alternative Course:  1a. User clicks the cancel button instead of the save button after editing article fields.  2a. DEILA Web App navigates back to the Dataset Manager dashboard page.  1b. User clicks reset instead of the save button after editing article fields.  2b. Field content will revert to the values stored in the database. |
| Exception Course:   1. Invalid field updates i.e., unsupported character text or too long of a title. 2. Notification to alert users that the request was invalid, and no edits were made to the selected data entry. |

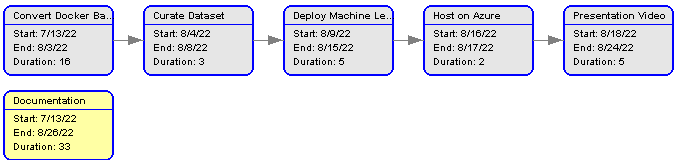
| Use Case: Delete an Existing Dataset Entry |
| --- |
| Primary Actor: data curator |
| Secondary Actor: |
| Precondition: User clicks the delete button associated with the row of an unwanted data entry. |
| Postcondition: data entry is removed from the DEILA database and DEILA front-end Web APP data store. |
| Main Success Scenario:   1. Click the delete or trash can icon on the row of the unwanted data entry. 2. Delete event triggered and serviced by the DEILA Web API. If the data entry with the id sent to the endpoint exists, it is removed from the DEILA database. 3. Notification to alert users of a successful data entry deletion. |
| Alternative Course: |
| Exception Course:   1. Data entry with id sent from DEILA front-end Web App does not exist in the database. 2. Notification to alert users of an unsuccessful attempt to delete data entry. |

| Use Case: Classification Service Notification |
| --- |
| Primary Actor: model user |
| Secondary Actor: |
| Precondition: Endpoint to retrieve classification was ping and a result was sent back. |
| Postcondition: Messaged displayed on Sentiment Analysis page. |
| Main Success Scenario:   1. Logits from DEILA Web API service returned. 2. Code modifies the display based on whether the results were negative or positive. 3. Code uses the difference between the two logit values to determine how negative or positive the result is. 4. DEILA front-end Web App. |
| Alternative Course: |
| Exception Course: |

| Use Case: Add New Diversity Equity and Inclusion Article to the Dataset Manually Using Web Crawler |
| --- |
| Primary Actor: data curator |
| Secondary Actor: |
| Precondition: User selects add by crawling option rather than manual article add and information for each field is added (origin, sentiment, basis) |
| Postcondition: New article is added to the diversity equity and inclusion dataset. |
| Main Success Scenario:   1. Post event triggered and serviced by the DEILA Web API. Service determines if the URL is a valid URL. 2. If valid the service will scrape/crawl the site and retrieve the title and content of the web article and save it to our dataset. 3. Notification to alert users of successful addition to dataset. |
| Alternative Course: |
| Exception Course:   1. Invalid fields inputted i.e., invalid URL, unsupported characters in text, title was too long. 2. Notification to alert users that the request was invalid, and no edits were made to the selected data entry. |

| Use Case: View Main Content Tables on Dashboard Page of Particular Feature |
| --- |
| Primary Actor: data curator |
| Secondary Actor: |
| Precondition: User selects one of the options on the menu: Dataset Manager, Basis Key. |
| Postcondition: Dashboard displays a table with contents from the DEILA database. |
| Main Success Scenario:   1. Get events triggered and serviced by the DEILA Web API. Service retrieves all contents in table and stores it in DEILA front-end Web App datastore. 2. DEILA front-end Web App retrieves datastore content and populates the Vue elements with the contents of the datastore. |
| Alternative Course: |
| Exception Course: |

## Roadmap



# Design

## Software Stack

The Software stack is composed of three main components deila.backend, deila.frontend, and DEILAdb. Using Azure Compute, I was able to deploy the Software Stack using three unique resources. The first resource is an Azure SQL Database. The second is a Web App which is used to host the backend and the third is a second Web App used to host the frontend.

**Frontend**

The frontend web application was scaffolded using Visual Studio 2022 Standalone TypeScript Vue Project Template. NPM’s serve is used for local development and Azure Compute services are used for production hosting. The codebase is implemented using TypeScript and Vue along with a dozen or more packages which are listed along with their version in the package.json. A few notable packages are Axios to make HTTP requests to the backend API and Vuex for state management.

**Dataset Curator Backend**

The dataset curator backend web application was scaffolded using Visual Studio 2022 ASP.NET Core Web API template. Local development done using Visual Studio 2022 and Azure Compute services are used for production hosting. The codebase is implemented in C# along with a handful of additional packages such as Entity Framework (for a code first approach) and Swagger (to test the API without the need for a working frontend).

**Machine Learning Backend**

The machine learning backend web application was scaffolded using a tianglo FastAPI tutorial. Local development is done on visual studio code and Azure Compute services are used for production hosting. The codebase is implemented in Python along with a dozen or so packages such as Torch and FastAPI to name a few, a list of the all packages needed to run this resource can be seen in the start-up.sh and requirements.txt file.

**Database**

The database was created as mentioned in the backend section using Entity framework and a code first approach. It runs on SQL Server which was deployed in the azure web portal. It consists of two tables whose attributes are listed below.

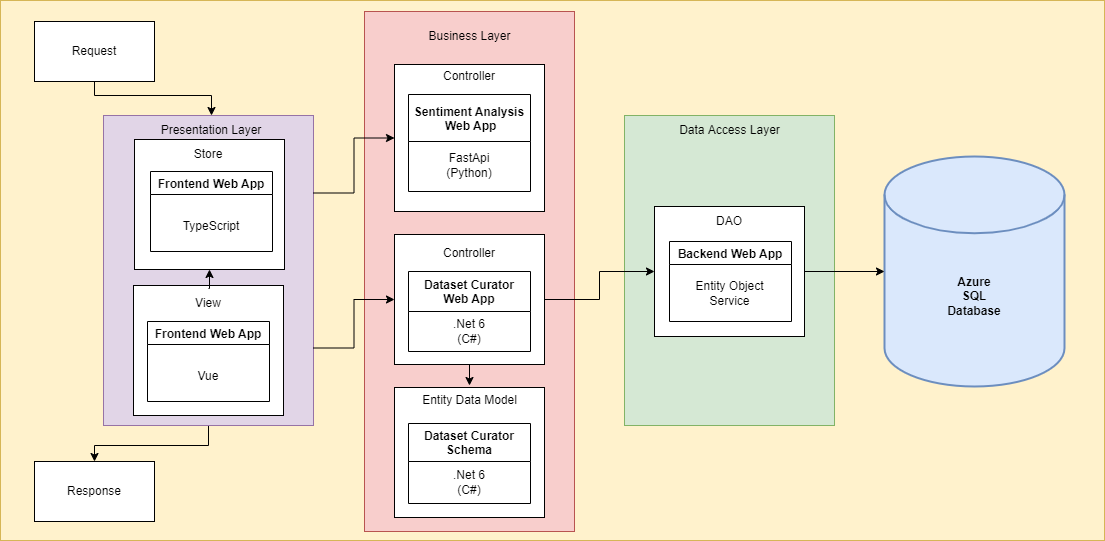
The Article table is composed of the following attributes.

| Property | Description | Required | Column Type |
| --- | --- | --- | --- |
| Id | A unique identifier. | Yes | int |
| Title | Title of Article. | No | nvarchar(max) |
| Content | Main content of the article. | No | nvarchar(max) |
| Sentiment | An identifier to indicate whether an article elicits a negative (0) or positive (1) feeling in the reader. | No | bit |
| BasisId | ID which is tied to an identifier in the basis table. | No | int |
| Origin | The article's source, typically a URL. | No | nvarchar(max) |

The Basis table is composed by the following attributes

| Property | Description | Required | Column Type |
| --- | --- | --- | --- |
| Id | A unique identifier. | Yes | int |
| Title | Name describing the basis. A basis is essentially a diversity category this nomenclature was taken from No Fear Act reporting. |  | nvarchar(50) |
| incidents | Number of reports under a specific basis. This field is intended for future use. |  | int |

## Software Ecosystem Map



## 

## Database Design

**Article**

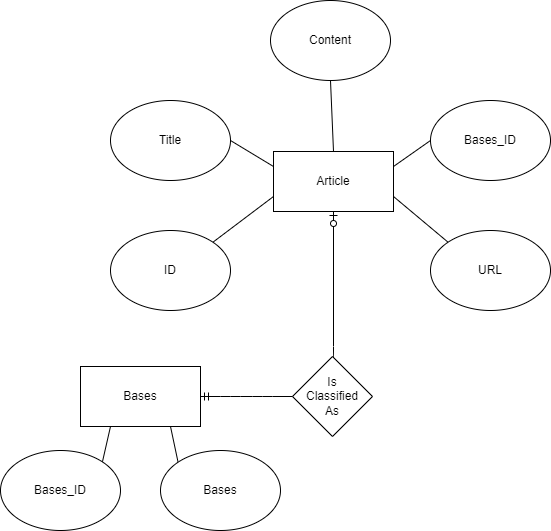
An Article is identified by an id and has the attribute title, content, origin, basis, and sentiment

**Basis**

A Basis is identified by an id and has the attributes title and incident

**Relationship between Article and Basis**

Each Article must be associated with a basis and more than one article can be associated with the same basis.



# 

# Implementation

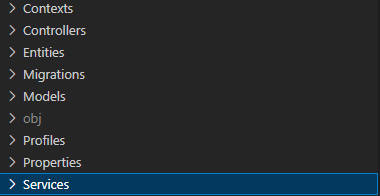
## Code Design

For my code design I tried to follow an object oriented flexible MVC approach. For the frontend I have separated the model under the interface folder the view in the views folder and the controller in the store folder.



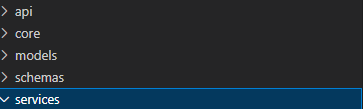
Frontend Folder Structure

The dataset curator backend uses a code first approach for the Data Access Object pattern. Using entity I have my models defined in the models folder my entities defined in the entities folder and I pass these objects along between the database and endpoints through the services.



Dataset Curator Backend Structure

Lastly the machine learning model backend uses a similar paradigm the dataset curator except services in this project are helper functions or business logic and the schemas are the objects passed to the endpoints which are under the api folder, the models are objects that are utilized in the business logic of the code.

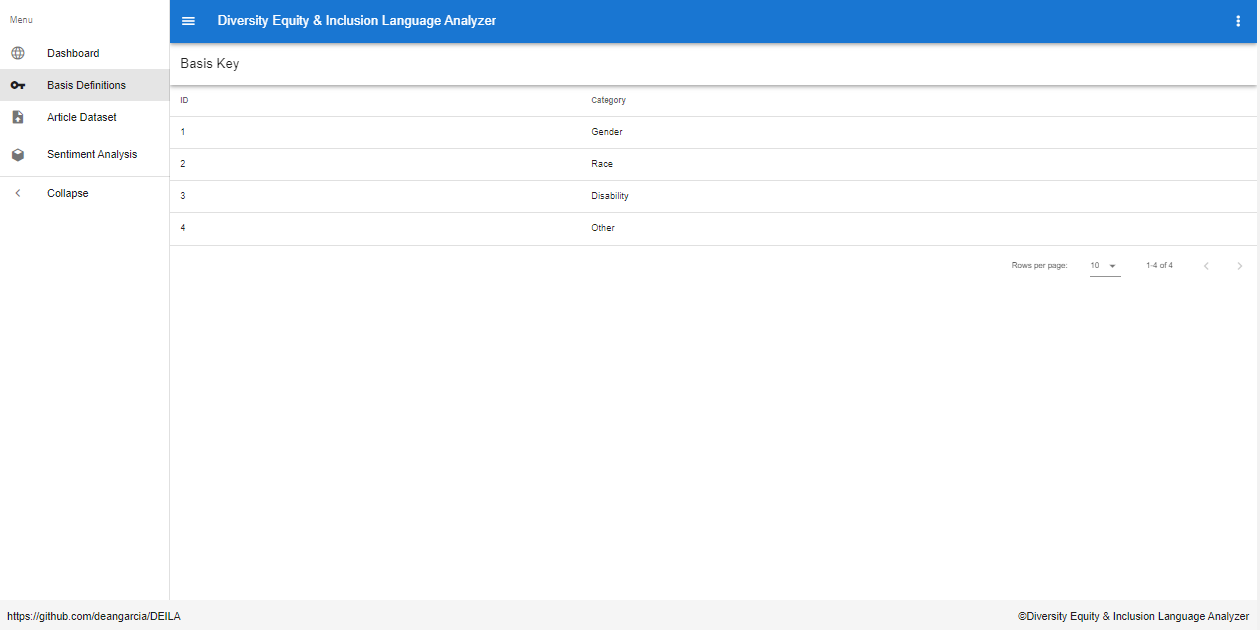


Machine Learning Backend Structure

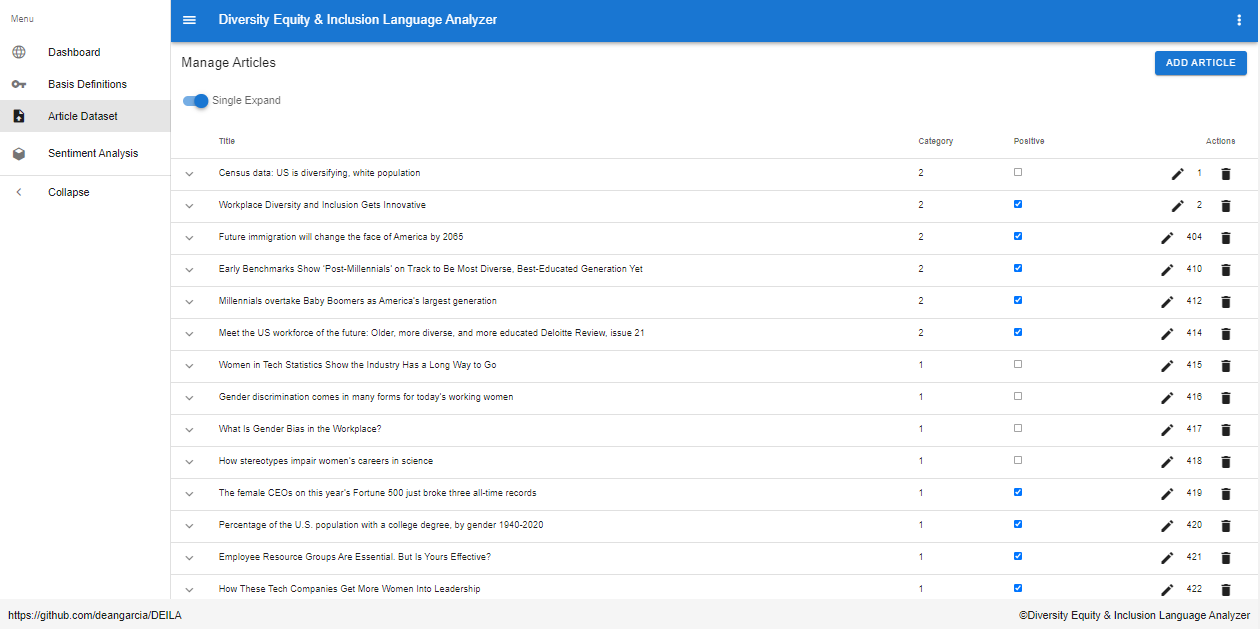
## Algorithm

The application uses GPT-2 transformer based model.

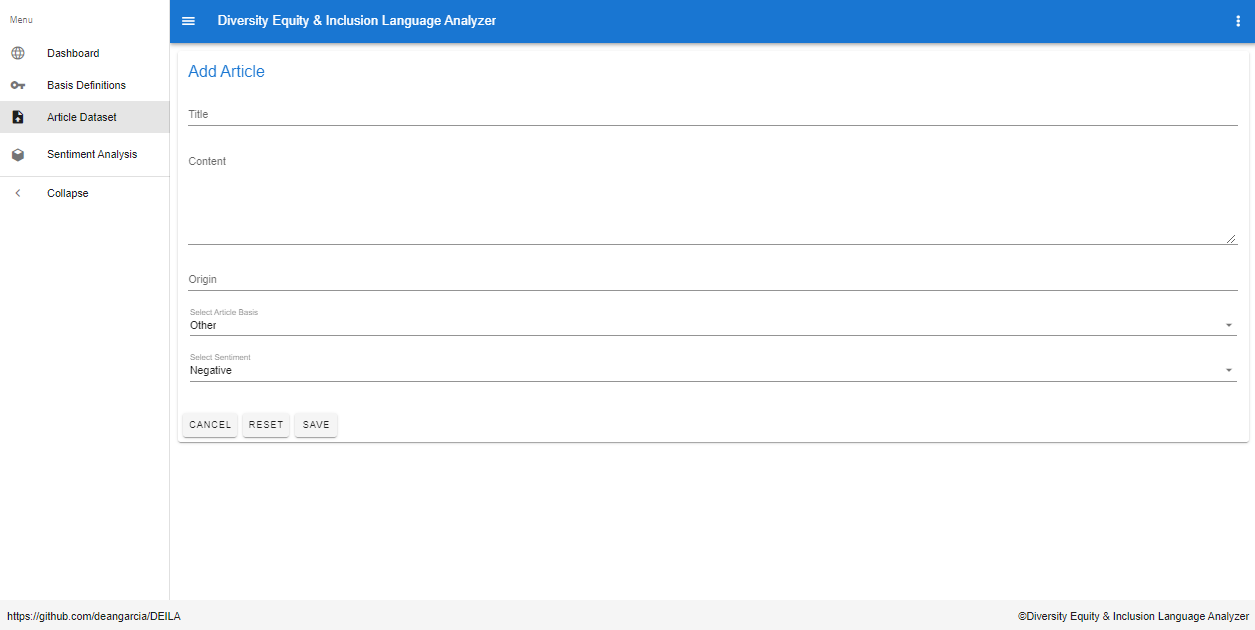
## UI/UX Design



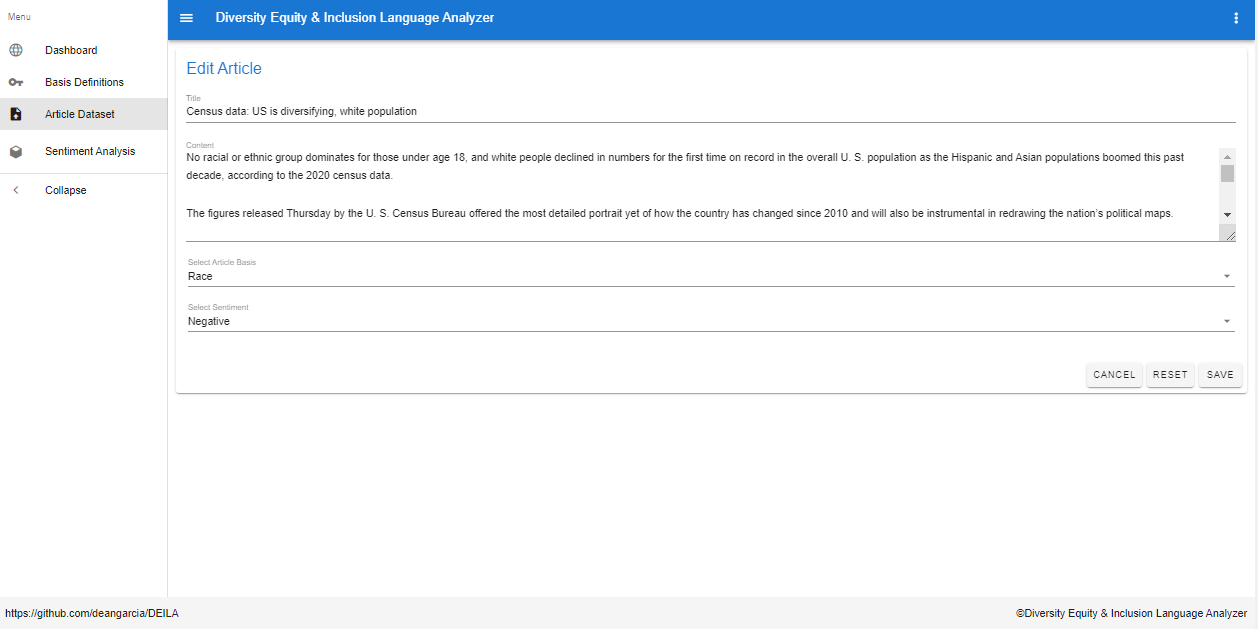
Dashboard



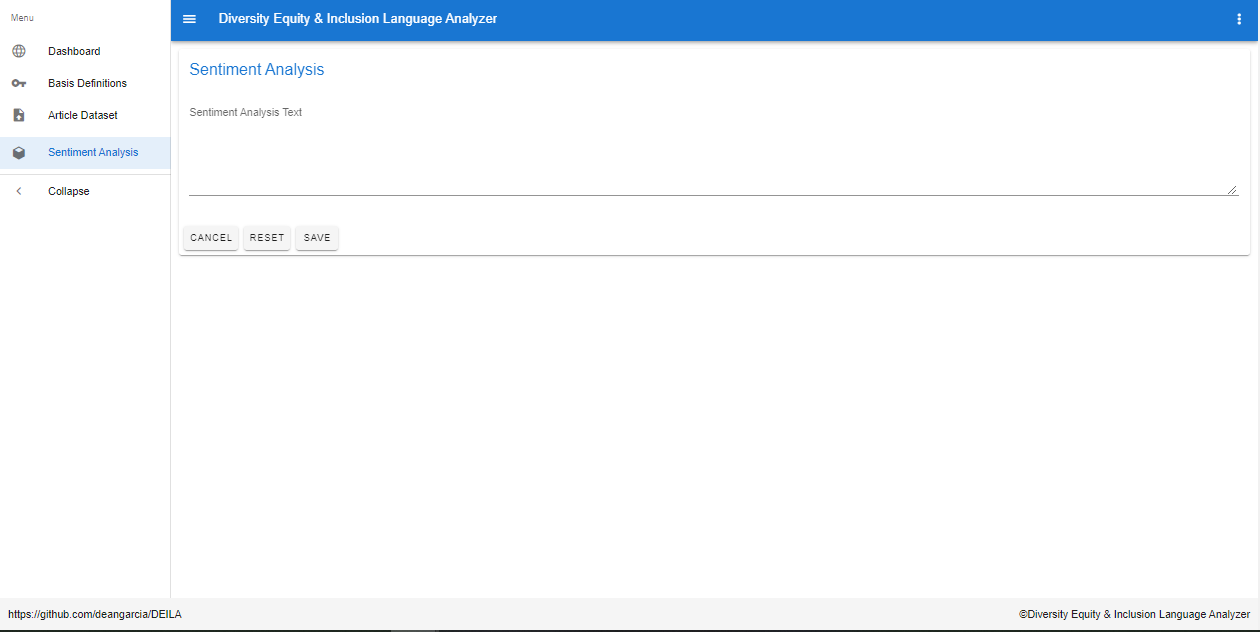
Article View



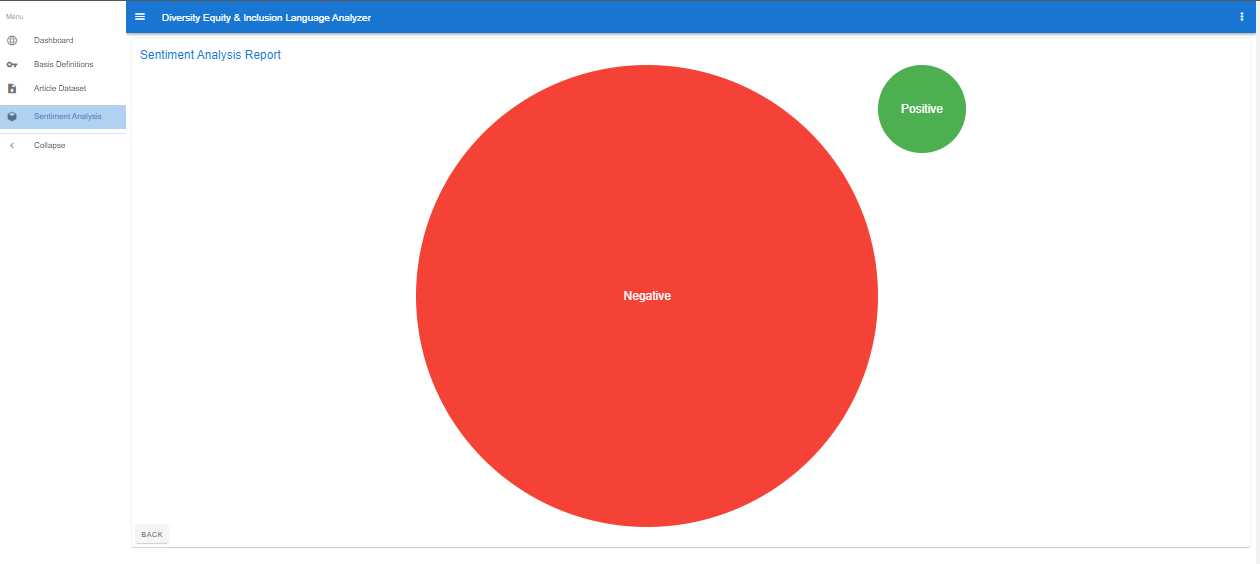
Article Add



Article Edit



Sentiment Analysis



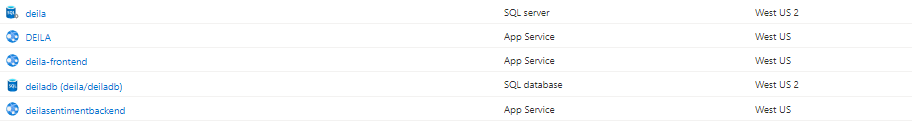
Sentiment Analysis Report

## Performance Enhancements

For the services portion of the dataset curator backend I created a generic repo which is implemented in the other schema specific repos to pass along the common crud operations.

## Release & Deployment

I deployed all of my features using azure resources, through either Visual Studio Code or Visual Studio 2022. I have a total of 5 resources in resource group I have title InclusiveCoding, the first title deila is my sql server which is where my database titled deiladb is running. I also have the dataset curator backend which is running as an .NET Core 6 app service and it is titled DEILA. It was deployed in Visual Studio Code 2022 using the publish feature integrated into Visual Studio. The frontend is another .NET Core 6 app service titled deila-frontend and was deployed using the deploy to web app feature integrated into Visual Studio code after installing the Azure extension. The machine learning backend is running on a Python 3.9 app service and is titled deilasentimentbackend it was deployed using the same method as the frontend (using the deploy to web app feature).



# Conclusion

## Final Project Impressions

I had a fun time developing this project learned a lot about deploying web apps and getting them to interact with each other. There were a lot of time constraints but overall I was able to accomplish my goals.

## Obstacles and Triumphs

One of the main obstacles I faced on this project was trying to deploy my FastAPI machine learning model on an azure app service. When trying to deploy to the free version my connection would time out when trying to run the start-up commands. I was able to learn how to install the dependencies using SSH but then had a hard time getting the application started. Another obstacle I faced was created getting data for my dataset, I did use a web crawler in some instances to pull article content but there was still a lot of cleaning to do. I had to remove adds or irrelevant content. Then I had to assign sentiments to the article entires, this added a layer of subjectivity that might be creeping into the model. Stil I was able to preserver and put in the leg work to complete 603 articles.

## Next Steps

The next step would be to continue growing the dataset so I can try retraining the model with more samples. Having a larger dataset will also make it possible to train another model using convolutional neural networks and or paradigms that require larger datasets. Create users profiles to preserve specific instances of the dataset the current dataset. I would also like to incorporate different fields in the dataset, like opinion-based columns so that the model can weight multiple sentiment columns each column would be an opinion from a different source. I would also like to fine tune a model that can put a special emphasis on special text (bold, underlined ect.). I would also like to incorporate some unit testing. Add a software maintenance plan. Continue to work on documentation. Deploy machine learning app service.

## Software Maintenance

There is no software maintenance plan currently in place.

## Overall Thoughts

I think I really pushed myself following through on completing this project. It is always good to have a goal to strive for and my ICF Mentor AMey Deshpande really helped me push past the difficulties of the project.