Web Tool for Phonemes - Phonix

Jeyendra Srinivas Datta Kanaparthi 50483770 jeyendra@buffalo.edu	Ram Kashyap Cherukumilli 50468970 ramkashy@buffalo.edu	Kriti Chapagain 50487430 kriticha@buffalo.edu	Haritha Kunta 50488070 harithak@buffalo.edu	Pratiksha Shirsath 50478912 shirsath@buffalo.edu

Overview:

The Web Tool for Phonemes creates a powerful word database with metadata and phoneme details, featuring an API and user-friendly interface. It offers specific phonetic queries, minimal maximal pairs, consonant patterns, articulation manners, and insightful phoneme statistics, serving speech-language pathologists, aiding children with pronunciation, supporting educators, and facilitating word-related exploration for general users

Introduction:

Effective communication is fundamental to both personal and professional interactions, yet many individuals face significant challenges in pronouncing specific words or sounds. These difficulties can arise from a variety of sources, profoundly impacting an individual's confidence and ability to communicate effectively. Pronunciation is a complex aspect of language that involves various phonetic elements, such as the place and manner of articulation, voicing of sounds, and the structure of phonemes. For language learners, non-native speakers, or individuals with speech disorders, mastering these intricate components of speech can be a daunting task. Mispronunciation can lead to misunderstandings or an inability to be understood, which can be frustrating and disheartening. Recognizing these challenges, PhoniX emerges as a comprehensive web tool designed to empower individuals facing pronunciation difficulties. It serves as an innovative solution for speech pathologists, language educators, and learners, offering a range of features to explore and understand the complexities of phonetics and pronunciation. By leveraging PhoniX, users can gain a deeper understanding of phonetic principles, practice pronunciation with targeted exercises, and overcome the barriers to clear and confident communication.

Motivation:

Phonix empowers individuals with speech difficulties and equip speech pathologists with better tools. We believe everyone deserves clear communication and overcoming speech challenges can profoundly impact lives. For individuals, we provide accessible, engaging tools to make speech improvement easier and more enjoyable. For speech pathologists, we offer innovative tools to personalize treatment plans, freeing up time for building patient relationships and delivering personalized care. Ultimately, we aim to remove communication barriers, build confidence, and empower both individuals and speech pathologists to achieve their full potential.

Use Cases :

SLPs (Speech-Language Pathologists):

1. Development of Phonemic Awareness: SLPs can utilize the app to help children with speech disorders develop phonemic awareness, which is essential for clarity of speech and language learning.
2. Articulation Practice: The app can provide targeted practice for clients with articulation disorders, allowing SLPs to customize exercises for specific phoneme difficulties.

Teachers/Educators :

1. Improving Reading and Spelling Skills: Teachers can incorporate the app into the academic curriculum to assist students understand the relationship between phonemes and spelling, thereby improving reading and spelling skills.

2. Language Learning Support: For ESL (English as a Second Language) students, the app can be a helpful tool in mastering English phonetics, which can help with pronunciation and comprehension.

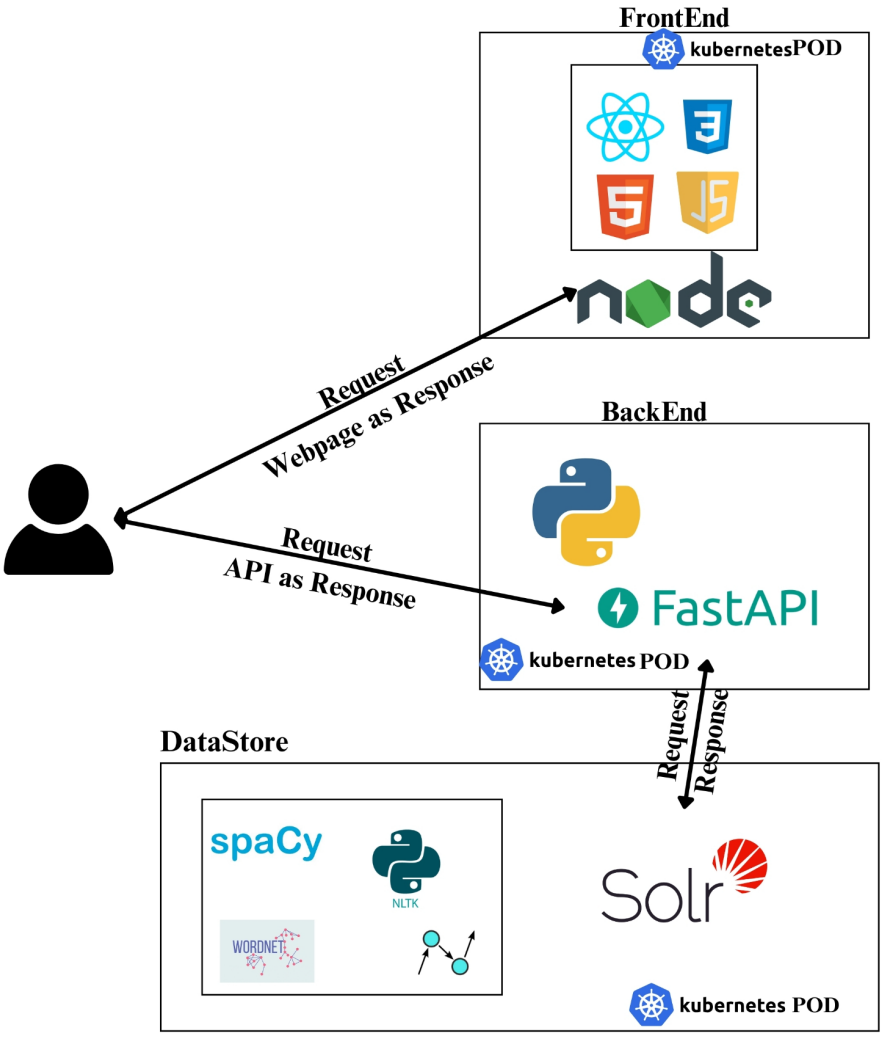
General Public:

1. Self-Directed Learning and Practice: People who want to improve their pronunciation or learn a new language can use the app for self-directed learning and practice.
2. Parental Support for Early Learning: Parents can use the app to help their children build their early language and reading skills by making learning phonemes pleasant and engaging.

Proposed Solution:

Our application features a fully functional website offering an intuitive and responsive user experience. It also has efficient APIs to quickly fetch data from an indexed database, ensuring fast and accurate responses for its users. With a robust backend, the application offers quick responses and ensures both stability and the ability to scale effectively.

Architecture Diagram:



Each of the component is explained in brief below:

User:

The starting point of interaction with the application. Users access the system typically through a web interface or mobile application. They send requests (such as queries or data inputs) and receive responses, experiencing the front-end features and functionalities.

Web Application:

The web application is where users interact directly with your system. It's built using HTML and CSS for structure and styling, React for a dynamic and responsive user interface, and Node.js for server-side processing. This combination ensures a robust, efficient, and visually appealing user interface. React's component-based approach allows for efficient updates and state management, while Node.js handles back-end tasks and communicates with the FastAPI.

FastAPI:

FastAPI acts as the intermediary layer handling the business logic. It receives requests from the web application, processes these requests, and interacts with Apache Solr. FastAPI is known for its high performance and easy-to-use features, making it a great choice for building efficient and scalable APIs.

Apache Solr:

Apache Solr is the search platform used in this architecture. It's responsible for indexing and searching large volumes of data quickly and efficiently. When FastAPI forwards a query, Solr searches through its indexed data and returns the relevant results. It is known for its powerful full-text search, hit highlighting, faceted search, and real-time indexing capabilities.
XLab University Server:

XLab Server :

This server is the central infrastructure that hosts the application's backend components, such as FastAPI and Apache Solr. It is responsible for various functions, including data storage, running application services, and managing network traffic. The server's role is crucial for maintaining the application's overall performance, security, and availability.

Each component plays a specific role in ensuring that the application functions efficiently, providing users with a seamless and effective experience.

Technical Details

Data Aggregation:

Data is collected from multiple source like Webster dictionary, CMU dictionary, Wordnet and Conceptnet the Script

We used a Json file inorder to maintain the data. The Reasons are

1. Easy to convert to Python data variables.

2. Easy to index them as Solr Documents.

Sample JSON Data Format:


```
{
 "word":"abacus",
 "phoneme":["AE","B","AH","K","AH","S"],
 "POA":["-","bilabial","-","velar","-","alveolar"],
 "MOA":["-","plosive","-","plosive","-","fricative"],
 "VOA":["-","voiced","-","voiceless","-","voiceless"],
 "H":["near-open","-","open-mid","-","open-mid","-"],
 "B":["front","-","back","-","back","-"],
 "R":["unrounded","-","unrounded","-","unrounded","-"],
 "freq":1017068,
 "POS":["NOUN"],

"category":["non_agentive_artifact","tablet","calculator","consumer_durable","tangible_thing","non
_powered_device"],
 "phoneme_length":6,
 "phoneme_str":"AE,B,AH,K,AH,S"
}
```


1. *Word*: what is the word
2. *Phoneme*: It is a Multi-valued, It contains what are the phonemes that are present for the word in an order.
3. *POA*: Position Of Articulation, It contains an array that has the articulation for every phoneme. Only Consonants have Position of Articulation, '-' is placed in for the Vowel sound
    a. It has values {"dental:alveolar", "simultaneous postalveolar and velar", "uvular", "labiodental", "palatal", "palato alveolar", "velar", "postalveolar", "dental", "velopharyngeal", "epiglottal", "pharyngeal", "nasal", "labial-palatal", "postalveolar", "bidental", "labial-velar", "glottal", "alveolo-palatal", "alveolar", "bilabial", "retroflex"}
4. *MOA*: Manner Of Articulation, It contains an array that has the articulation for every phoneme. Only Consonants have Manner of Articulation, '-' is placed in for the Vowel sound
    a. It has values {"lateral flap", "flap", "implosive", "lateral affricate", "ejective", "approximant", "stop", "percussive", "click", "fricative", "fricative release", "trill", "fricative;approximant", "nasal release", "plosive", "nasal", "lateral click", "lateral approximant", "tap", "lateral fricative", "affricate", "release", "lateral release"}
5. *VOA*: Voice Of Articulation, It contains an array that has the articulation for every phoneme. Only Consonants have Voice of Articulation, '-' is placed in for the Vowel sound
    a. It has only values {"voiceless","voiced"}
6. *H*: Highness, Only for Vowel Sounds.
    a. It has only values {"close-mid", "open", "mid", "close", "open-mid", "near-close", "near-open"}
7. *B*: Backness, Only for Vowel Sounds.
    a. It has only values {"back", "near-back", "near-front", "central", "front"}
8. *R*: Roundness, Only for Vowel Sounds.
    a. It has only values {"rounded", "unrounded", "more rounded", "less rounded"}
9. *freq*: It is the frequency value taken from google web trillion corpus.
    a. It signifies how frequent the word is. The higher the value the word is, the more frequent it is.
    b. Used to display the most frequent words in day to day life
10. *Category:* Category is an array of strings that contains the category for which the word belongs to.
11. *POS*: Parts of Speech. It contains an array of parts of speech Possible Values are {} → todo
12. *phoneme_len*: It contains an Integer that contains the length of phonemes it contains.

13. *phoneme_str*: It contains a single String that contains all the phonemes of the word in comma(,) separated.

- ## POS Data Collection:

With POS tagging, each word is tagged with the appropriate part of speech.
The script consists of a two-tier approach, consisting of data from both Spacy and Wordnet.
The data goes through three steps:

-Loading the data: Words are loaded from a JSON file, with structured format for input.

-Tagging: Each word is passed through the Wordnet. If the wordnet returns an empty set, it is passed through Spacy's POS tagging to generate the POS for each phoneme of the word.

-Output data: The output is a JSON file where each word is annotated with its POS tags and the source of these tags (WordNet or spaCy).

WordNet categorizes words under different parts of speech depending on their usage and meaning, for identifying all possible POS tags for a given word.The system queries WordNet for each input word to retrieve all associated POS tags. These tags include nouns (n), verbs (v), adjectives (a and s - satellite adjectives), and adverbs (r).
When WordNet returns no data for a word, then it performs spaCy's tagging, which is based on machine learning models trained on large text corpora. This provides the most probable POS tag for the word. It is a powerful and efficient NLP library that can determine the POS of a word within the context of a sentence. In cases where WordNet does not yield results (e.g., for neologisms or less common words), spaCy's POS tagger is employed. SpaCy's strength lies in its contextual accuracy and the breadth of its POS tags, covering a wide range of grammatical categories beyond WordNet's scope.

- ## Articulation Data Collection :

The script is designed to enhance the dataset by providing detailed phonetic and articulatory information for each word.
This requires 2 files for the script to operate:

1. IPA-Phoneme Mapping Data File (ipa_phoneme.json)
   This mapping from Phoneme to IPA can be found here.
2. Articulation Data for Phonemes (Arti_data.json):
   This data is acquired from Github. The data is validated by a speech specialist.

It operates in three main stages:

1. **Data Loading:** Initially, it loads words and their phonetic representations from the main data file in JSON format. This file contains the basic phoneme data for each word, which serves as the foundation for further processing.
2. **Phonetic Conversion and Articulation Tagging:** In this crucial stage, the script transforms the phoneme data into International Phonetic Alphabet (IPA) symbols using another JSON file ("ipa_phoneme.json"). It uses "Arti_data.json" to tag each phoneme with specific articulatory properties such as Place of Articulation (POA), Manner of Articulation (MOA), and Voicing (VOA) for consonants and Height, Backness and Rounding for Vowels.
3. **Output Generation:** The final step involves compiling this enriched data into a new JSON file ("data3.json"). Each word in this file is annotated with its IPA representation and a set of articulatory tags, offering a comprehensive phonetic and articulatory profile.

- ## Frequency Data Collection:

  Word Frequency is the field which provides the user with the option to view words based on relevancy to their day to day lives. It is considered as one of the sorting parameters for the results shown.
  This script is used to add word frequency information to our dataset:
  1. Reading the Google Web trillion Corpus file: The script reads the CSV file named 'unigram_freq.csv' accessed from Kaggle using Pandas. This file contains words and their corresponding frequency counts.
  2. Creating a Word Frequency Dictionary: The script iterates through each row in the above read file, extracting the word and its count, and stores this information in a dictionary which is used ahead.
  3. Updating Data File with Word Frequencies: For each item in the JSON data, the script checks if the word is present in the frequency dictionary. If found, it updates the word with its frequency ('freq') and sets the source of this frequency data. If the word is not found, it sets the 'freq' value to 0 and leaves the source field empty.

- ## Category Data Collection:

  The script first gathers category data from two sources: ConceptNet and WordNet. Words with an "Is-A" relationship in ConceptNet were prioritized for categorization.

  1. **ConceptNet Processing:**
     The script in Data-Scripts/work_conceptnet.py in then delves into the specific "Is-A" relationships for each word. It parses each relation line, extracting the "Is-A" relation type and the related word. This information is then stored in a dictionary, building a map of each word's connections.
  2. **WordNet Processing:**
     The script in Data-Scripts/Word_category.py loads the WordNet corpus. It iterates through each entry in the JSON data. For words lacking "Is-A" relationships in ConceptNet, the script turns to WordNet. It retrieves synsets, which are sets of synonymous words representing a concept. By exploring the hypernyms of each synset (words higher in the semantic hierarchy), the script uncovers broader categories for the target word.
  3. Then combines the extracted categories from both ConceptNet and WordNet for each word, prioritizing ConceptNet's connections for initial classification. It then fills in any missing categories using WordNet hypernyms.

- ## Phoneme Data Collection:
  With all words we have aggregated from various sources, we used speechbrain pretrained grapheme to phoneme model to generate phonemes for all the words, the script to generate phonemes is present in Datascripts/Add_PhoneticData.py

- ## Words Collection:

  1. **Sources : Webster Dictionary, WordNet, ConceptNet, CMU Dictionary**
  2. **Webster Dictionary:** The words that belong to webster dictionary are scraped from the raw dictionary file from this repo, collected words with more than 2 letters to avoid junk https://github.com/matthewreagan/WebstersEnglishDictionary
  3. **Wordnet:** The words that belong to WordNet are scraped directly from the site
  4. **Conceptnet:** The words from the conceptnet are downloaded using the below file https://github.com/commonsense/conceptnet5/blob/master/testdata/reference/assoc/assoc.csv
  5. **CMU :** The words from CMU Dict are used from the NLTK library from available in python

Implementation details:

- **Solr:**
  1. We used Solr as a our datastore to store the data for the following reason
     a. **Type of data:** Data contains many fields and we search for a word that satisfies some constraints in the data. So our search needs to be fast. Solr internally uses inverted search that helps to retrieve data fast.
     b. It also supports Phonetic search, Multi-language support and Faceted search. We Present used only Faceted search. For future reference the other features can help.
  2. We need a core to index all the data.
     a. Command to create a Solr core "phonemes" name of the core.

```
solr create -c phonemes
```

  3. We need to use the python Script in Data-Scripts → solr → create_index.py
     a. This script is used to create a schema in the "phoneme" solr core
     b. The script requires the "pysolr" python package.
     c. In the solr object creation in the python script make sure you insert the correct url along with the core for solr
        - Example here : http://xlabk8s2.cse.buffalo.edu:30007/solr/phonemes
  4. Add data to Solr Core.
     a. Make Sure the Json file that contains all data is in the same path. With the name "alldata.json"
     b. Run the script to insert the data to Solr
     c. The script has a prerequisite for the "pysolr" package.
- Fast-Api:
  1. Used Fast-Api to expose the API
  2. Phoneme_extraction.py is the file that contains the Code to handle API's using FastApi
     a. fastapi , pysolr are the prerequisite packages needed to be installed
  3. While starting the app make sure you use 0.0.0.0 to start the app. Command is placed in CMD in the Dockerfile.

```
uvicorn Phoneme_extraction:app --host 0.0.0.0 --port 8000
```

API Documentation:
All the APIs are written in the PhonemeExtraction.py file. You need to create a virtual environment and install Pysolr, FastAPI.

All the APIs use query parameters which are inputs for the functions

1. **word_details(/api/word/{word}):**
   Input: word
   Output: This Function returns all the details of the word(Phonemes, Articulation Manners, Categories etc)
2. **phoneme_details(/api/phoneme/):**
   Input: include, exclude, sort_parameter, sort_order
   Output: This function returns all the words that have phonemes from include string which doesn't contain any of the phonemes exclude string. The results are sorted by frequency in decreasing order by default, they can also be sorted by either number of phonemes in a word or alphabetical order of words if passed as parameters

3. **get_phonemes_by_position(/api/phoneme_pos/):**
   Input: initial, middle, final, sort_parameter, sort_order
   Output: This function returns all the words that have initial phoneme from initial string, middle phoneme from middle string, final phoneme from final string. Middle can be any position other than first and last. The results are sorted by frequency in decreasing order by default, they can also be sorted by either number of phonemes in a word or alphabetical order of words if passed as parameters. This function also has find_indices helper function which assists in returning the indices that needs to be highlighted for the UI

4. **minimal_details(/api/minimal/):**
   Input: phoneme1, phoneme2, length
   Output: This function returns all the minimal pairs of input length given the two differing phonemes. It uses a returnIndex helper function which returns the index that is differing that needs to be highlighted for the UI

5. **minimal_maximal_word(/api/minimal_maximal_word/):**
   Input: word
   Output: This function returns all the minimal and maximal pairs of a given word. This function uses 3 helper functions: placeDiff, mannerDiff, voiceDiff which are useful for finding if a pair is a minimal pair or a maximal pair. It also uses a returnIndex helper function which returns the index that is differing that needs to be highlighted for the UI

6. **get_minimal_maximal_cat(/api/minimal_cat/):**
   Input: category
   Output: This function will return all the minimal and maximal pairs in a given input category from length 3 to 20. The main helper function minimal_maximal_cat_helper calculates minimal and maximal pairs of a given category and certain length. The minimal_maximal_cat_helper also uses placeDiff, mannerDiff, voiceDiff helper functions for differentiating minimal and maximal pairs

7. **poa_detail("/api/poa/"):**

   Input: place of articulation, manner of articulation, voice of articulation, height, backness, round, sort_parameter, sort_order
   Output: This API will return all the words based on any combination of articulation manners for consonants and vowels. The results are sorted by frequency in decreasing order by default, they can also be sorted by either number of phonemes in a word or alphabetical order of words if passed as parameters

8. **category_details(/api/category/):**

   Input: category, sort_parameter, sort_order
   Output: This API returns all the words from a chosen category. The results are sorted by frequency in decreasing order by default, they can also be sorted by either number of phonemes in a word or alphabetical order of words if passed as parameters

9. **get_words_by_pattern(/api/words_by_pattern):**

   Input: Pattern, sort_parameter, sort_order
   Output: This function returns all the words based on the consonant pattern that is taken from a string. It uses a regular expression pattern to find the matching words. The results are sorted by

frequency in decreasing order by default, they can also be sorted by either number of phonemes in a word or alphabetical order of words if passed as parameters

10. **get_words_by_cluster(/api/words_consonant_cluster):**

    Input: cluster, sort_parameter, sort_order
    Output: This function returns all the words out of the consonant cluster that is taken as a string. It uses a regular expression pattern to find the matching words. The results are sorted by frequency in decreasing order by default, they can also be sorted by either number of phonemes in a word or alphabetical order of words if passed as parameters

- **Node.js**:
  - Whenever User Hits the URL. They make call to Node Js and Nodejs is responsible for sending the webpages
  - When The user receives web pages based on the actions, specific Api calls are made to FastAPI

Setup Proxy:
The requests from frontend to API are routed using proxy-middleware
This proxy middleware uses target URL: **http://xlabk8s2.cse.buffalo.edu:30008** which is the starting point to all API calls from the frontend.
Proxy Setup can be found in SetupProxy.js.

Firstly, install http-proxy-middleware in the frontend folder using

```
npm install http-proxy-middleware
```

Minimal And Maximal Pair Algorithm

The main idea behind the minimal pair algorithm is that the words will have the same length, therefore finding words with the same length which differ by only one phoneme is the crux of the algorithm.

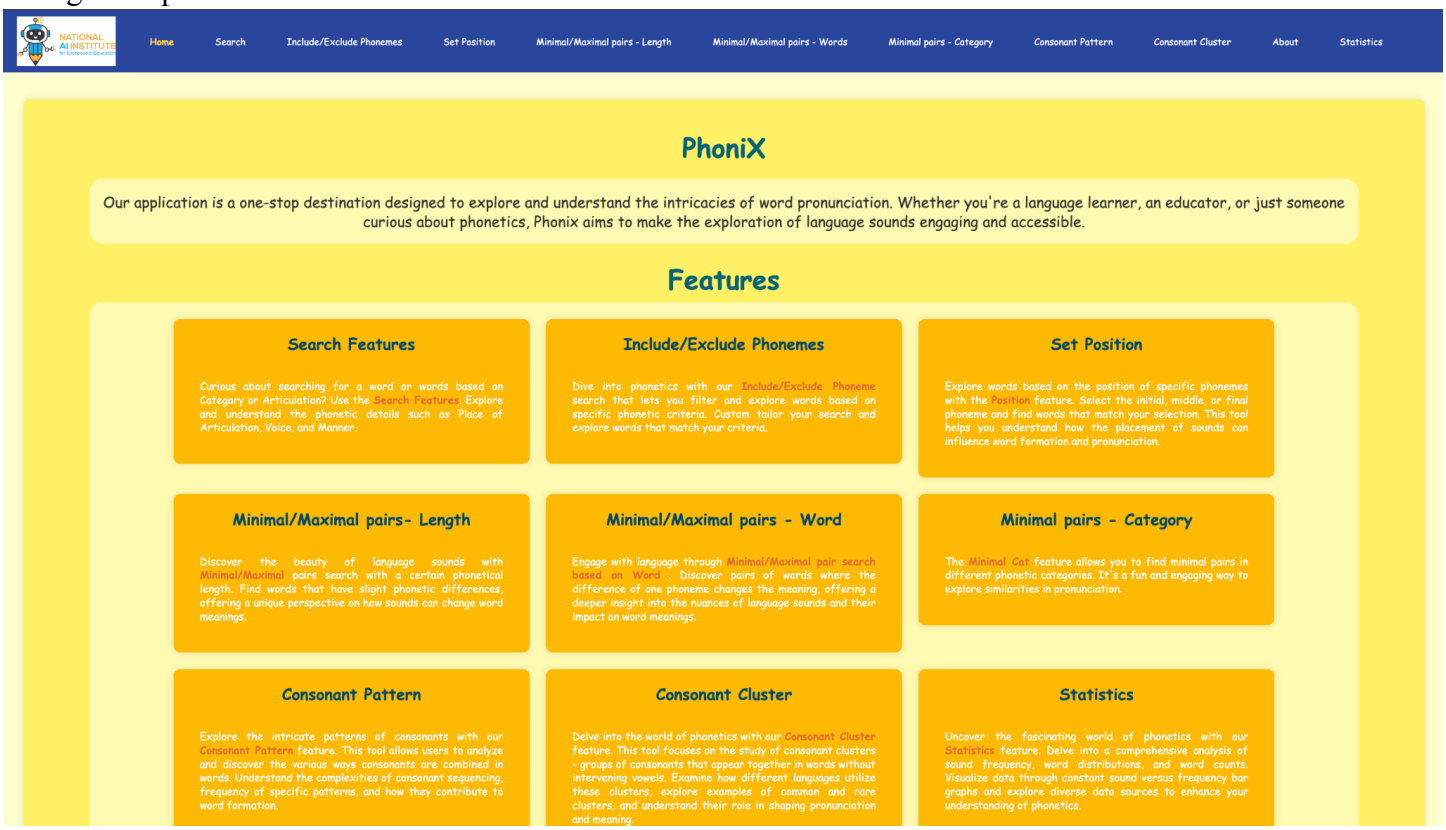
1. Queried using a regular expression to find out all the words with phoneme1 or phoneme2 with a given length and stored them in an array. 2. Used this array to store phoneme strings of each word, their articulation manners for making the api response clear.
3. Using a regular expression, stored all the words which contain only phoneme1 in a hash set
4.Iterated over the array of all the words with phoneme2 and replaced phoneme2 with phoneme1 to check if the word is present in hashset
5.If the word is present in the hashset it means the words are differing by only those phonemes. So they are minimal pairs.
6.Using articulation manners and placeDiff,mannerDiff, voiceDiff functions classified them into maximal pairs if there are any

Features:

1. Home:

Home page is the landing page of the website. It consists of a feature card with a brief description of each pages. Each of these features is accessible through a button-like interface on the Home page. The users can navigate to different feature-specific pages by clicking on these buttons, which are designed as feature cards. This user-friendly design ensures that even those new to phonetics can easily access and benefit from the

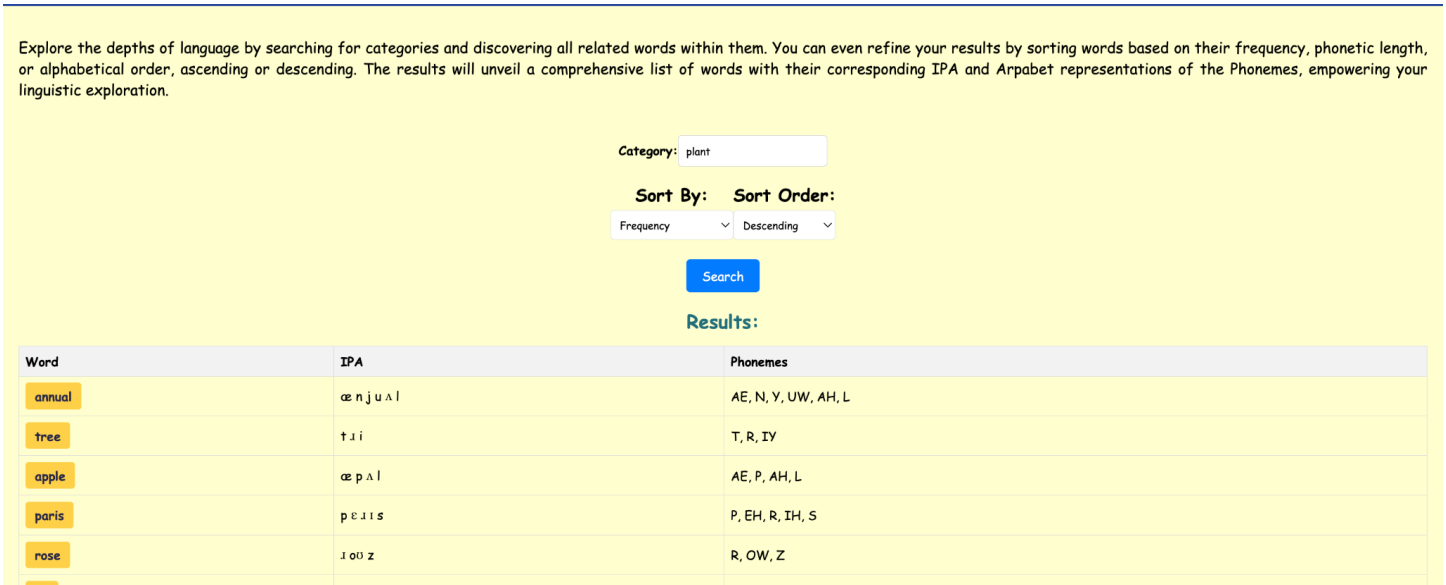
website's diverse functionalities. Users can easily find and access the tools they need without navigating through complex menus.



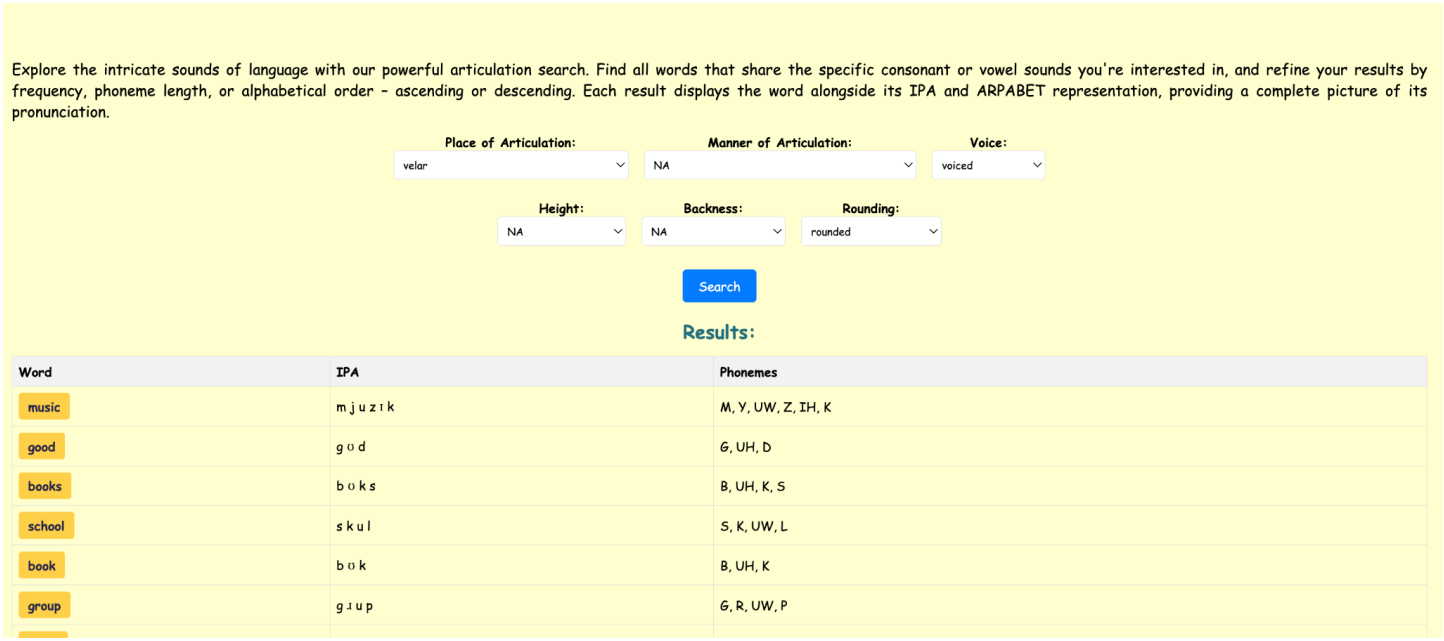
2. Search:

The Search Bar is the primary tool where users input words they want to search for. They can customize their search based on their requirements. The Search Bar consists of three different features:

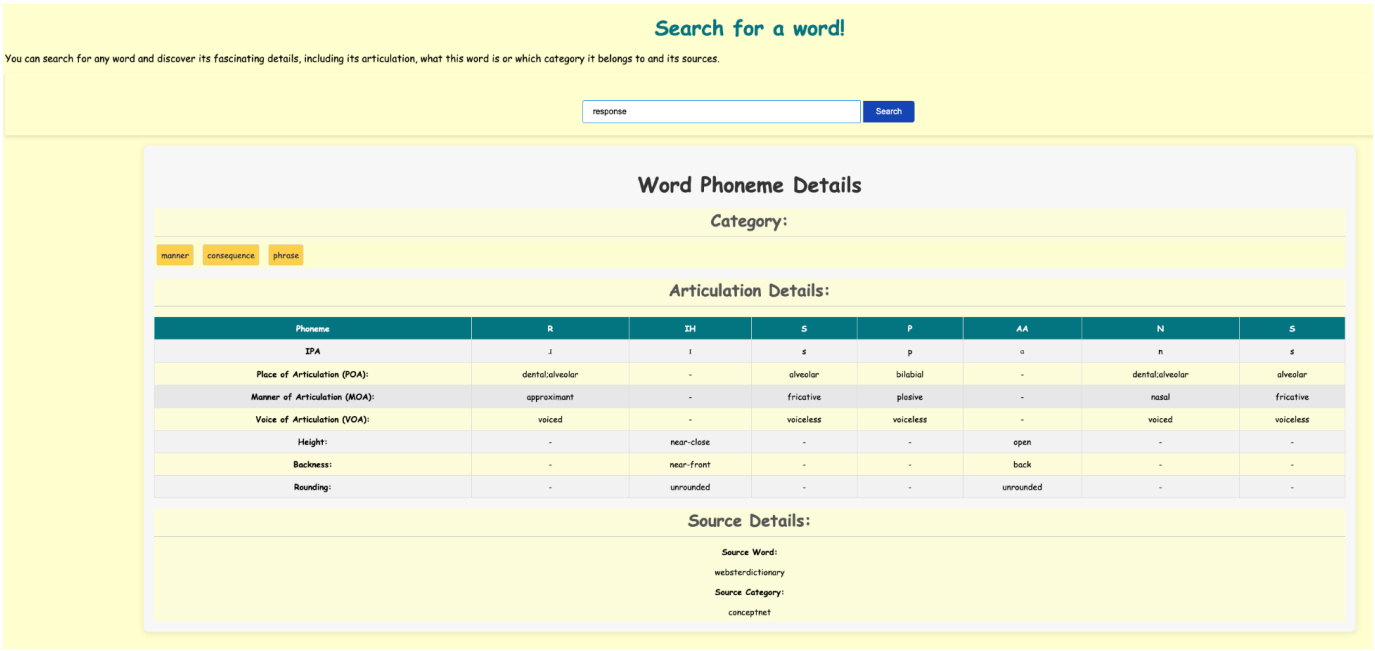
- **Search by Category:** The user can search for a category to fetch all the words under it with their corresponding IPA and Phonemes along with sorting based on Phonetic length, frequency, alphabetical order.



- **Search by Articulation:** This feature allows users to search words based on specific articulation characteristics, such as place of articulation (like bilabial, velar), manner of articulation (such as nasal, stop, fricative), voicing (voiced or voiceless sounds), Height, rounding, and backness.



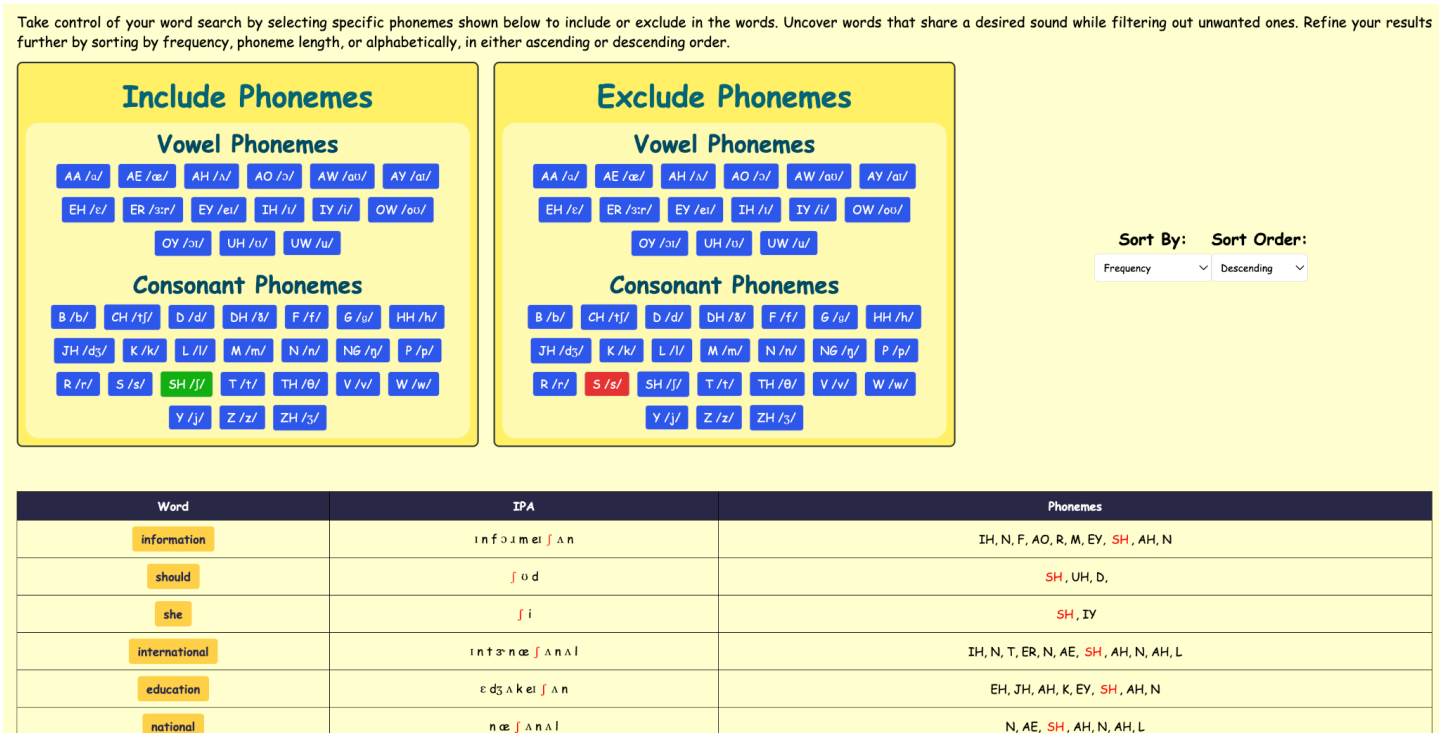
- **Search by Word:** This feature provides the phonetic details, Articulation Details, and Source of a word.



3. Include-Exclude:

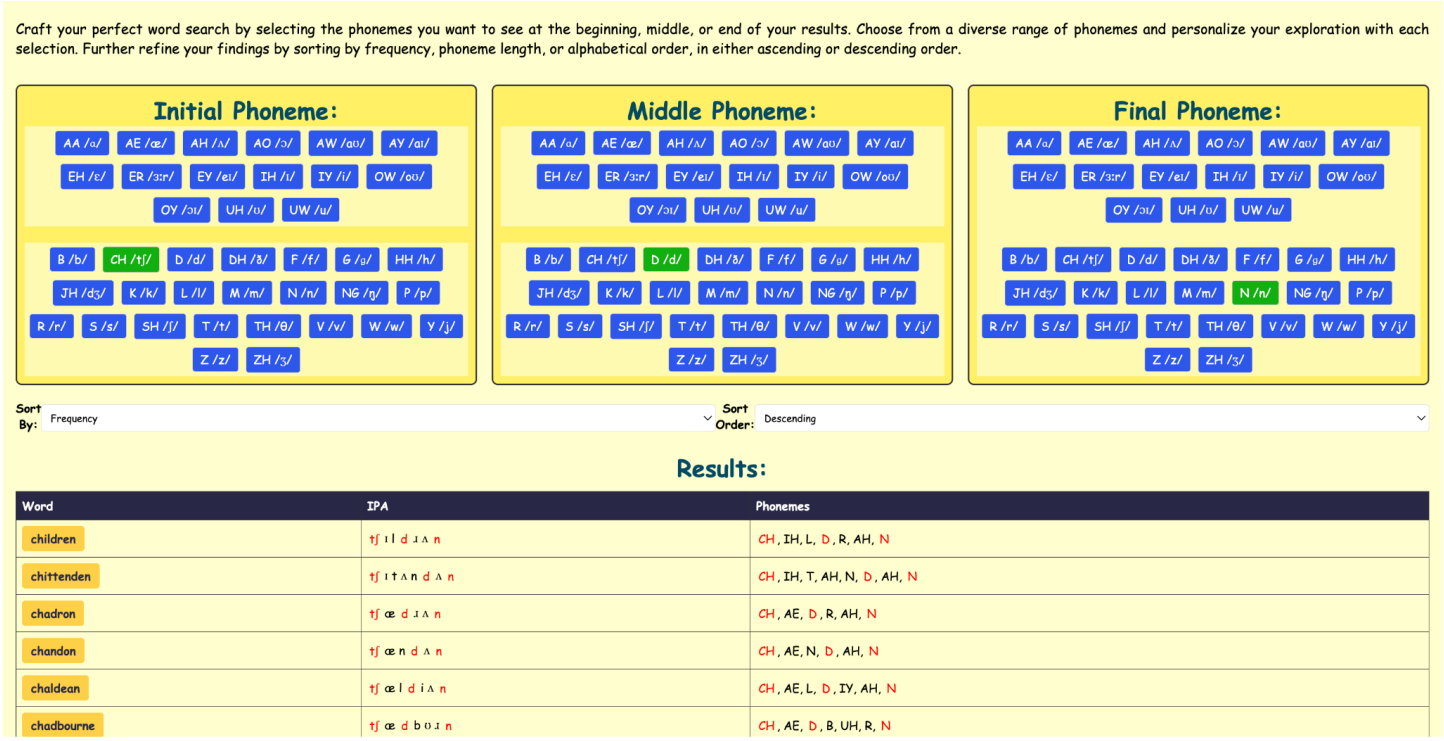
This feature lets you control your word discovery by focusing on specific sounds. Choose the phonemes you want to hear (include) and block out those you don't (exclude). For example, when the user has difficulty with ‘sh’ phonemes but not ‘s’, then you can select the phonemes you want to include and exclude respectively . The

results contain words that share your desired sound while eliminating unwanted ones. The results can be further sorted based on frequency, phoneme length, or alphabetical order, in ascending or descending order.



4. Set Position:

This feature focuses on targeting specific phoneme locations. Choose the sounds you want to see at the beginning, middle, or end of your results. This lets you explore a diverse range of phonemes and uncover hidden patterns with each selection. The results can be further sorted based on frequency, phoneme length, or alphabetical order, in ascending or descending order.



5. Minimal-Maximal pairs:

Minimal pairs are two words that differ by only one phoneme in only one parameter of articulation. Maximal pairs are two words that differ by only one phoneme in more than one parameter of articulation.

- **Minimal and Maximal Pairs based on Word :**

  This feature focuses on identifying pairs of words that are minimal and maximal pairs.
  Use : Users could input a word, and the system would display a list of minimal pairs and a separate list of maximal pairs for that word.

Enter a word, and discover its minimal and maximal pairs, unveiling a network of words linked by subtle phonetic changes.

Minimal Pair And Maximal Pair for Word

catch Search

Matching Minimal Pairs:

Matching Maximal Pairs:

Minimal Pairs	Phonemes	Maximal Pairs	Phonemes
batch	B, AE, CH	cab	K, AE, B
cac	K, AE, K	cad	K, AE, D
cack	K, AE, K	caen	K, AE, N
cadge	K, AE, JH	cag	K, AE, G
caff	K, AE, F	cahn	K, AE, N
calf	K, AE, F	cal	K, AE, L
cap	K, AE, P	cam	K, AE, M
capp	K, AE, P	camm	K, AE, M
caq	K, AE, K	can	K, AE, N
cas	K, AE, S	cann	K, AE, N
cash	K, AE, SH	cav	K, AE, V

- **Minimal and Maximal Pairs based on Length:**

  This feature finds Minimal and Maximal word pairs based on the two phonemes selected and also the length of the words. This gives practice based on the phonemes selected and show results sorted based on the frequency for more relevant results.
  Use : Users could select the buttons for the two phonemes and enter the word length, and the system would present minimal and maximal pairs fitting the input.

Select a phoneme from each tile to define the differing sound and give the length, and unveil all words containing these variations as their minimal pairs. Explore the fascinating world of minimal pairs and witness the transformative power of a single sound change.

Phoneme 1 :

AA /ɑ/ AE /æ/ AH /ʌ/ AO /ɔ/ AW /aʊ/ AY /aɪ/
EH /ɛ/ ER /ɜɹ/ EY /eɪ/ IH /ɪ/ IY /i/ OW /oʊ/
OY /ɔɪ/ UH /ʊ/ UW /u/

B /b/ CH /tʃ/ D /d/ DH /ð/ F /f/ G /ɡ/ HH /h/
JH /dʒ/ K /k/ L /l/ M /m/ N /n/ NG /ŋ/ P /p/
R /r/ S /s/ SH /ʃ/ T /t/ TH /θ/ V /v/ W /w/ Y /j/
Z /z/ ZH /ʒ/

Phoneme 2 :

AA /ɑ/ AE /æ/ AH /ʌ/ AO /ɔ/ AW /aʊ/ AY /aɪ/
EH /ɛ/ ER /ɜɹ/ EY /eɪ/ IH /ɪ/ IY /i/ OW /oʊ/
OY /ɔɪ/ UH /ʊ/ UW /u/

B /b/ CH /tʃ/ D /d/ DH /ð/ F /f/ G /ɡ/ HH /h/
JH /dʒ/ K /k/ L /l/ M /m/ N /n/ NG /ŋ/ P /p/
R /r/ S /s/ SH /ʃ/ T /t/ TH /θ/ V /v/ W /w/ Y /j/
Z /z/ ZH /ʒ/

Enter Length

4 Search

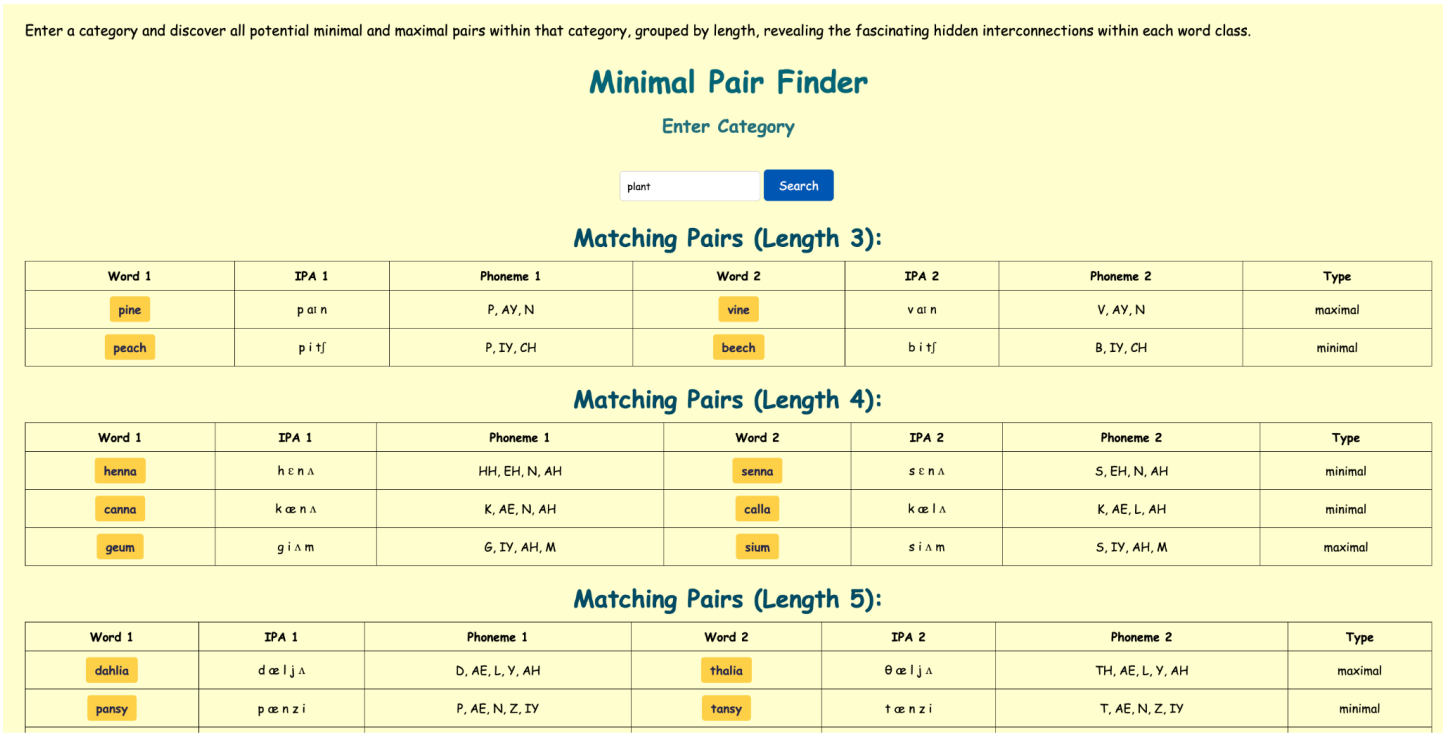
Word	IPA	Manner of Articulation / Height	Place of Articulation/ Backness	Voicing / Rounding
B	b	plosive	bilabial	voiced
P	p	plosive	bilabial	voiceless

Matching Minimal Pairs:

Word 1	IPA 1	Phoneme 1	Word 2	IPA 1	Phoneme 2
brice	b ɹ aɪ s	B, R, AY, S	price	p ɹ aɪ s	P, R, AY, S
boast	b oʊ s t	B, OW, S, T	post	p oʊ s t	P, OW, S, T
blees	b l i z	B, L, IY, Z	please	p l i z	P, L, IY, Z
bart	b ɑ ɹ t	B, AA, R, T	part	p ɑ ɹ t	P, AA, R, T
bress	b ɹ ɛ s	B, R, EH, S	press	p ɹ ɛ s	P, R, EH, S

- **Minimal and Maximal Pairs based on Category:**

Minimal and Maximal word pairs are categorized based on Categories like animals, plants, etc. The pairs would be similar words within a category.
Use: The UI could allow users to input a category, and then display minimal and maximal pairs within that category of all varying lengths.



Enter a category and discover all potential minimal and maximal pairs within that category, grouped by length, revealing the fascinating hidden interconnections within each word class.

Minimal Pair Finder

Enter Category

plant Search

Matching Pairs (Length 3):

Word 1	IPA 1	Phoneme 1	Word 2	IPA 2	Phoneme 2	Type
pine	p aɪ n	P, AY, N	vine	v aɪ n	V, AY, N	maximal
peach	p i tʃ	P, IY, CH	beech	b i tʃ	B, IY, CH	minimal

Matching Pairs (Length 4):

Word 1	IPA 1	Phoneme 1	Word 2	IPA 2	Phoneme 2	Type
henna	h ɛ n ʌ	HH, EH, N, AH	senna	s ɛ n ʌ	S, EH, N, AH	minimal
canna	k æ n ʌ	K, AE, N, AH	calla	k æ l ʌ	K, AE, L, AH	minimal
geum	g i ʌ m	G, IY, AH, M	sium	s i ʌ m	S, IY, AH, M	maximal

Matching Pairs (Length 5):

Word 1	IPA 1	Phoneme 1	Word 2	IPA 2	Phoneme 2	Type
dahlia	d æ l j ʌ	D, AE, L, Y, AH	thalia	θ æ l j ʌ	TH, AE, L, Y, AH	maximal
pansy	p æ n z i	P, AE, N, Z, IY	tansy	t æ n z i	T, AE, N, Z, IY	minimal

6. Consonant Pattern:

The "Consonant Pattern" feature is an innovative tool designed to assist users in exploring and discovering words based on specific consonant and vowel arrangements. This feature functions through a user-friendly interface where users can select buttons representing consonants and vowels, crafting a pattern of their choice.

Use : The feature offers an interactive interface with buttons representing consonants and vowels. Users can select these buttons to form a pattern. For example, a user could choose a pattern like "CVCV" (where 'C' stands for consonant and 'V' for vowel).

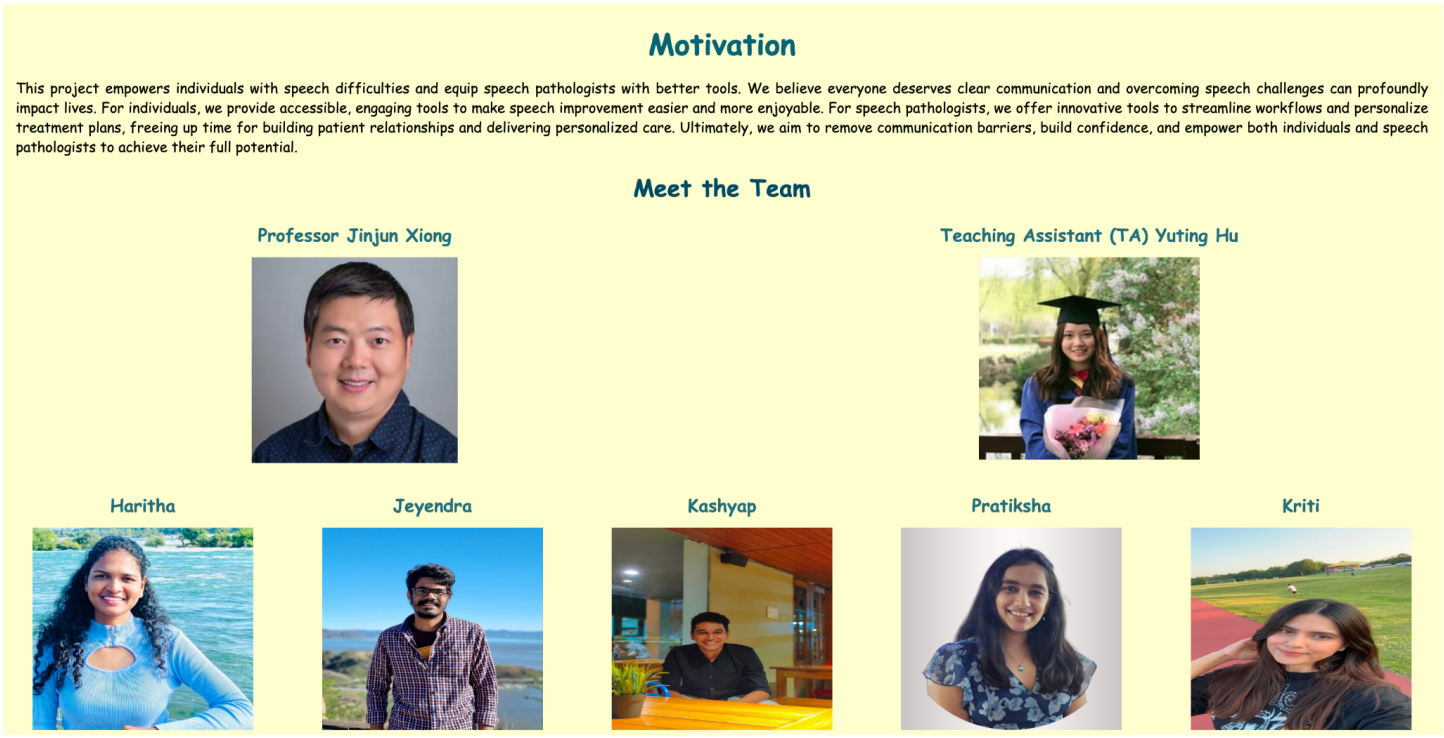
7. Consonant Cluster:

The "Consonant Cluster" feature is a specialized tool designed to enhance the exploration of consonant combinations.

Use: Users can interact with the user interface by clicking a button to add consonants to a cluster. For each consonant you add, select your preferred consonant from the dropdown menu that appears.
The word with the selected consonant combination will be presented to the user.

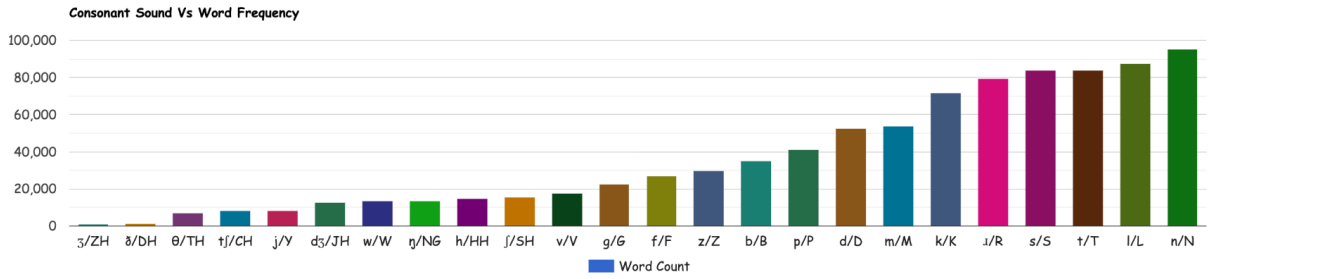
6. About:

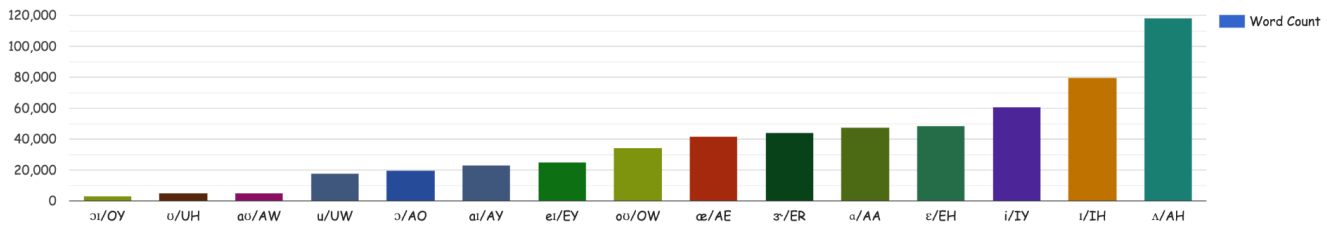
This page consists of a short brief on motivation of the project including the introduction to the team.

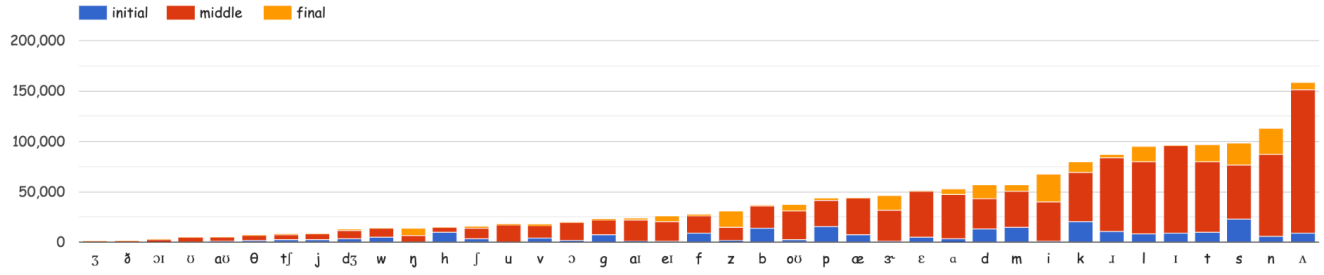


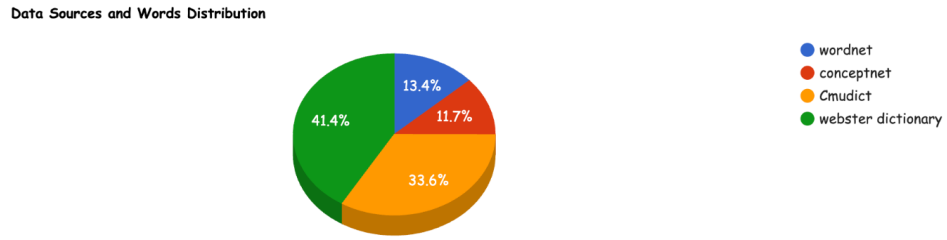
7. Statistics:

The Statistics page provides valuable insights into phoneme frequency, distribution, and source diversity of the data (Wordnet, Conceptnet, CMU Dictionary, Webster). Besides, It also provides the insights on word count of consonant sound, word frequency along with the position of the phonemes.









Deployment:

We used Kubrnetes inorder to deploy the application the deployment contains 2 phases

1. Docker image building.
    a. We have 2 images Node and FastApi
    b. We have folder named Deployment Files which has 2 sub-folders
        i. Node
            1. In the Frontend folder copy package.json, package-lock.json, src folder in to the Node folder and build them using docker
        ii. FastApi
            1. In the FastApi folder copy the Phoneme_extraction.py and build the image
2. Deploying in Kubernetes Cluster.
    a. First Deploy the Solr using the Solr.yaml
    b. Use the Scripts Create_schema.py and inser_data.py to create schema and insert the data to Solr
    c. Then Deploy node.yaml and fastapi.yaml
    d. Node js is service is exposed to 30009 and Fast api is exposed to 30008

Open Issues:

The current category sources, Conceptnet and Wordnet, rely on synsets, resulting in hierarchical categorization that includes both concrete and abstract categories. To enhance precision, there is a need for a more fine-grained classification system that can effectively distinguish between these categories for better results.

Future Scope:

1. **Multi Lingual Support:**
   When complete linguistic data, such as phonemes, articulation techniques, word frequency, categories, and language-specific elements of speech, are provided, the application will support various languages with ease. The underlying API logics remain unchanged, only the user interface needs to be changed.
2. **Sound Inclusion:**

Integrate audio features, offering sound for each phoneme and pronunciation for every word.

3. **Context Based Words:**
   Implement advanced natural language processing techniques to dynamically generate context-based sentences for user-searched words and context-based words with usage examples for category searches.

Contributions:

1. **Pratiksha Shirsath - 20%**
   a. Research about alternatives for IPA : IPHOD, Klatt, etc
   b. Research Paper about Minimal Pairs and Maximal pairs
   c. Phonemes and Articulation Data Study and Collection
   d. Collection of Vowel Parameter Data
   e. Writing file for mapping from IPA to Phoneme
   f. Script for mapping Phoneme to Articulation and its Data
   g. Word Frequency Data Collection and Script
   h. Frontend and Styling for Minimal Maximal Based on Length
   i. Frontend and Styling for Minimal Maximal Based on Category
   j. Frontend and Styling for Consonant Cluster
   k. Phoneme to IPA conversion and Highlighting in Minimal Maximal Based on Words, Search Feature
   l. IPA component in Search Feature, Consonant Cluster and Consonant Pattern.
   m. IPA and Phoneme Component Highlighting in Include Exclude and Set Position

2. **Jeyendra Srinivas Datta Kanaparthi - 21%**
   a. Work-Flow of Data Collection
   b. Data Unioning from Multiple Sources
   c. Research about CMU dictionary and Adding data.
   d. Conceptnet Parsing and adding categories from Conceptnet and Wordnet
   e. Basic algorithm for Minimal, Maximal Pair words Generation
   f. Writing Api for Position Based words retrieval
   g. Writing Api for fetching details of words
   h. Writing Api for Consonant Cluster
   i. Research about Solr and Setting Up the Solr (Core creation, Data Indexing)
   j. Writing Dockerfile for Nodejs and FastApi
   k. Writing Kubernetes Deployment files for Solr, Nodejs , FastApi
   l. Script to Collect Statistics.
   m. Front End for Statistics Page
   n. Front End for About Page

3. **Haritha Kunta - 20%**
   a. Scraped words from the Webster dictionary
   b. Scraped Parts of Speech from the NLTK-Brown Corpus.
   c. Research on Word2Vec - for visualizing phoneme vectors
   d. Data scraping for Categories from Brown Corpus, ConceptNet, and WordNet, ConceptNet API.
   e. Data Collection for Word Frequency - The Educator's Word Frequency Guide Book (suggested by Prof.), Google Ngrams, Lexicon
   f. Research on Minimal - Maximal Pairs (research paper)
   g. Frontend & styling for Search features and Search by Word pages & contributed in Search Word page.
   h. Frontend & styling for Search by Articulation
   i. Frontend & styling for Include-Exclude page

j. Frontend & styling for Set Position page
k. On all pages, Setup the navigation linking from word buttons to SearchDetails page
l. Dynamic search on Consonant cluster and Consonant pattern, Include Exclude, Set position pages
m. Worked on Sorting technique on Frontend on Include-Exclude, Set Position, Consonant Cluster, Consonant Pattern pages

4. **Ram Kashyap Cherukumilli - 21%**
    a. Research about Data Sources : wordnet for categories(synnets), webster dictionary
    b. Research about articulations, Minimal-maximal pairs,multiple oppositions
    c. Scraped Data from Webster Dictionary
    d. Script for generating phonemes based on words using Speech Brain g2p model
    e. Setup proxy as a middleware for connection between frontend and backend
    f. Algorithm for all Minimal Maximal pairs (category, word, differing phoneme)
    g. API design of include exclude phonemes
    h. API design of minimal pairs if two phonemes are given
    i. API design of minimal pairs if a category is given
    j. API design of minimal pairs if a word is given
    k. API design of to find all the words based on articulations
    l. API design to find all the words based on categories
    m. API design to find all the words based on parts of speech
    n. API design to find all the words based on Consonant Pattern
    o. Design for highlighting indices in set position API
    p. Front End for the minimal maximal pairs based on a word
    q. Front End for the Consonant Pattern
    r. Managed Kubernetes service deployments files for Solr, nodejs, FastAPI

5. **Kriti Chapagain - 18%**
    a. Research about Data Sources: Spacy, Wordnet
    b. Script for Parts of Speech
    c. Research on Word2Vec and Doc2Vec for analysis of Phonemes
    d. Proxy and No-CORS policy in the frontend
    e. Implemented Search Bar feature
    f. Implemented Word Phoneme Details
    g. Linking of words to the Search Bar
    h. Implemented Home Page
    i. Created Navigation Bar
    j. Search Handler
    k. React Router DOM - Routing of Pages
    l. Navigating Feature for the Website
    m. Phonemes representation in UI Buttons
    n. Styling of the website: UI, aesthetics, Color palette, Logo