REPORT FOR
ONLINE DICTIONARY
(A PROJECT BUILT BY UB-CSE-611)

GUIDED BY:

- PROF. JINJUN
- AMIR

BUILT BY:

- SAMAR PRATAP SINGH - 50469740
- AKHIL REDDY - 50465462
- HAMZA HAFIZ - 50478874
- CHRIS JACINTH YADALA - 50478073
- ROBIN MATHEW - 50478005

Motivation

In today's digital age, online dictionaries have become an indispensable tool for individuals seeking quick and reliable information about words and their meanings. However, one major issue that plagues many online dictionaries is the overwhelming presence of advertisements. These advertisements often overshadow the primary purpose of the dictionary, which is to provide concise and accessible definitions to users. As a result, users find themselves bombarded with intrusive ads, making it challenging to navigate through the clutter and extract the information they initially sought.

The incessant presence of advertisements in online dictionaries not only hampers the user experience but also disrupts the flow of learning. Imagine a student or a language enthusiast trying to look up a word and understand its usage. Instead of finding a clean and straightforward interface with the desired information, they encounter a barrage of flashy ads that distract and confuse. This unnecessary clutter not only slows down the learning process but also hinders the comprehension and retention of new vocabulary. As a consequence, the user's original intention to expand their knowledge and understanding of words gets overshadowed by the overwhelming commercialization of the online dictionary platform.

Recognizing this problem, there is a growing need for an online dictionary that places the user's experience and learning at the forefront. By creating a dictionary that focuses solely on delivering clear and concise word definitions without the intrusion of disruptive advertisements, users can finally access the information they seek without unnecessary distractions. Such a user-centric approach would not only enhance the learning experience but also foster a more positive relationship between users and online dictionaries. By providing a clean and uncluttered interface, users can efficiently navigate through the dictionary, absorb information, and broaden their vocabulary with ease. Ultimately, the goal is to prioritize the user's needs and ensure their journey through the dictionary remains smooth and productive, free from the frustrations caused by excessive advertisements.

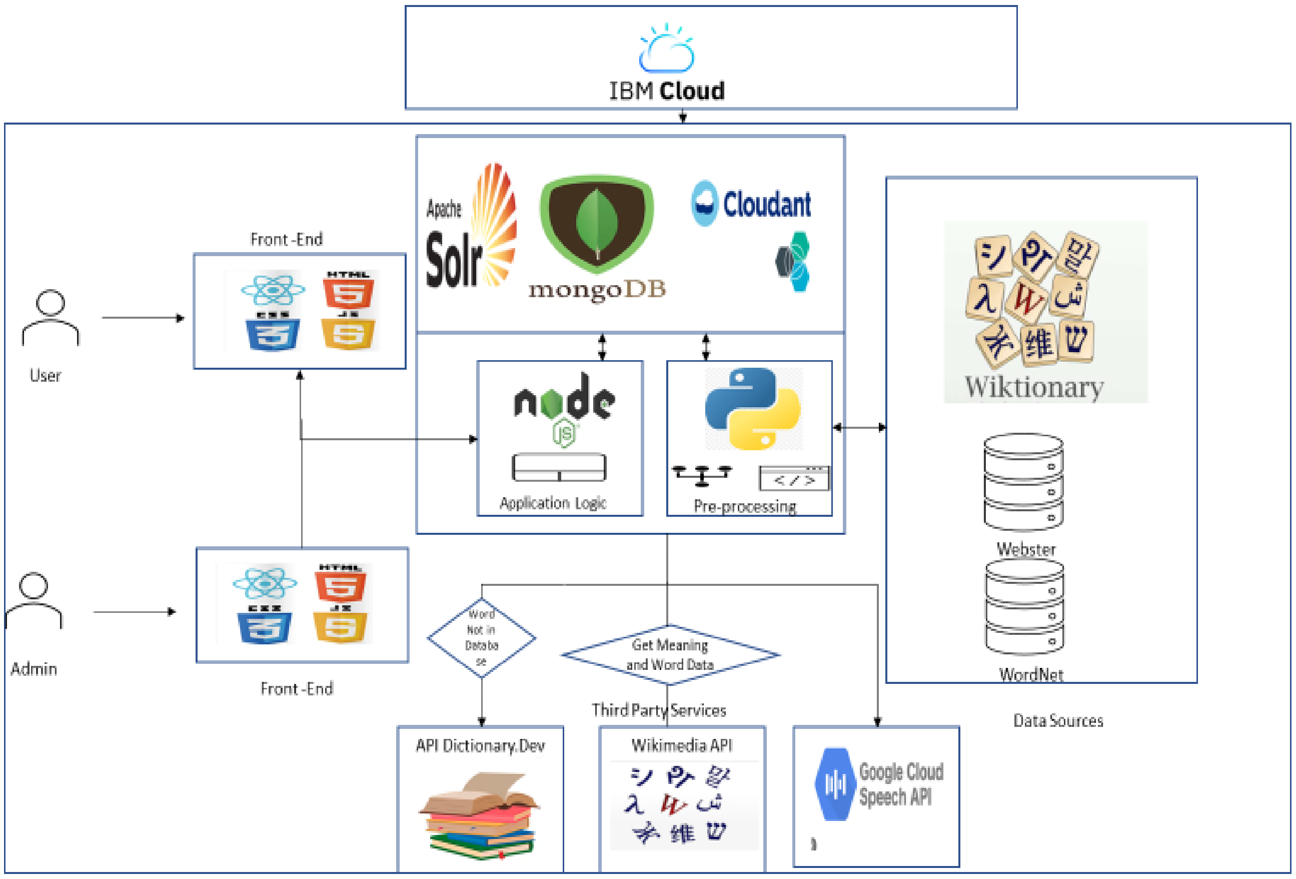
Introduction:

We are proud to present the online dictionary, designed with a simple user interface that prioritizes ease of use and focuses on giving the user's what they need. Unlike other dictionaries cluttered with advertisements, our platform is completely ad-free, ensuring an uninterrupted learning experience. Moreover, we have placed a strong emphasis on maximizing the retention of words and their meanings. By providing concise and accurate definitions, audios for the words, along with contextual examples, we aim to facilitate a deeper understanding and long-term retention of vocabulary for our users. With our user-friendly interface and commitment to delivering high-quality content, we strive to offer an online dictionary that truly empowers users to expand their language skills effortlessly.
Use Cases built are:

1. **Extensive Word Database**: Our online dictionary boasts an impressive collection of over 125,000 carefully curated words, ensuring that users have access to a wide range of vocabulary at their fingertips.
2. **Comprehensive Meanings & Examples**: Each word in our dictionary is accompanied by many detailed and accurate meanings, providing users with clear explanations and promoting a deeper understanding of the language.
3. **Origin-based Differentiation**: We go beyond basic definitions by highlighting the differentiation of meanings based on the word's origin.
4. **Integration with Google Audio**: To provide users with a comprehensive language learning experience, our online dictionary integrates with Google Audio, offering a vast collection of audio recordings in multiple languages. Users can explore different accents and pronunciations, broadening their understanding of global linguistic variations.
5. **Natural Audio Pronunciations**: With a repository of over 45,000 natural audio recordings, users can listen to authentic pronunciations, helping them develop accurate pronunciation skills and better grasp the nuances of spoken language.
6. **Contextual Sentence Examples**: Our extensive database includes more than 600,000 sentences sourced from reputable news outlets like US News, enabling users to explore words in relevant contexts and enhance their comprehension and usage.
7. **Word of the Day**: We engage and intrigue users by featuring a daily word that expands their vocabulary, introducing them to fascinating and lesser-known terms that enrich their language skills.
8. **Trending Words**: Stay up-to-date with the language trends as we provide a dedicated section listing the most searched words, allowing users to discover popular and relevant terms that are making waves in the linguistic landscape.

9. **User Contribution**: We empower users to actively participate in enhancing our dictionary by providing a seamless workflow for suggesting new words, ensuring a collaborative and continuously evolving platform.
10. **Statistics and Insights**: Gain valuable insights into the extensive data we have accumulated, such as word frequency, usage patterns, and user statistics, enabling users to explore from a statistical perspective.
11. **Admin Portal Workflow**: Our dedicated admin portal enables manual addition of new words based on user suggestions, ensuring a reliable and up-to-date dictionary that reflects the needs and contributions of our user community.
12. **IPA (International Phonetic Alphabet) Support**: We understand the importance of accurate phonetic representation, which is why our dictionary includes IPA symbols alongside word entries. This feature assists users in pronouncing words correctly and enhances their phonetic awareness.
13. **API Integrations**: Our online dictionary goes beyond its extensive database by incorporating API integrations with reliable external sources. This ensures that even if a word is not present in our database, users can still access relevant word data from trusted sources, expanding the range of words and information available to them.
14. **Swagger Hub for Direct API Consumption**: We empower developers and language enthusiasts by providing direct access to our dictionary's APIs through Swagger Hub. This enables users to leverage our APIs for various applications, such as building language-learning apps, integrating word data into their projects, or creating custom language tools.

Architecture Diagram



Challenges:

1. **Data Gathering and Exploration:** The initial challenge we faced was gathering relevant data for our specific problem. We explored various sources such as WordNet, Webster, and Wiktionary, eventually finding Wiktionary to be suitable for our needs. However, the data was vast and contained a lot of irrelevant information. Understanding Wiktionary data itself was a challenge because it is very confusing to interpret, and it requires linguistic knowledge. Preprocessing the data was also a challenge due to the huge inconsistencies in the data.
2. **Preprocessing and Understanding Word Context:** Preprocessing the data presented its own set of challenges. We had to delve into understanding how the context of words is defined, including the realization that words often have multiple meanings. Additionally, we discovered that words can have multiple etymologies, adding another layer of complexity to our preprocessing efforts.
3. **Data Structure and Storage:** Determining the appropriate data structure to hold the vast amount of data required careful consideration. Storing words may seem straightforward, but the granularity of information, such as parts of speech and their variations, required meticulous planning. We chose to store the data as JSON objects and utilized MongoDB as our database solution.
4. **Real-time Integration with Google Audio:** Ensuring real-time integration with Google Audio for providing pronunciations in different accents posed a technical challenge. We worked on establishing seamless integration to offer users a comprehensive language learning experience with accurate and diverse audio resources.
5. **Parsing and Indexing News Articles:** Extracting data from extensive news article sources required robust parsing and preprocessing techniques. Python proved valuable in preprocessing the data, and we leveraged Solr for efficient indexing to enable quick and accurate retrieval of information from news sources.
6. **Workflow for Adding Words:** Developing an intuitive workflow for adding new words required careful consideration. We devised a state management system that enables efficient addition of words based on their current state, ensuring a streamlined process for user contributions.
7. **Admin-Driven Word Addition:** Enabling administrators to add words seamlessly with a single click required extensive research on APIs and online sources. We parsed Wiktionary data comprehensively and integrated it in real-time with the Wiki Media API to facilitate efficient and accurate word additions.
8. **Deployment and Containerization:** Deploying our application on Code Engine proved to be a unique challenge that required a deep dive into understanding Docker and its deployment capabilities. We studied and implemented containerization to ensure smooth deployment and scalability.

9. **User Interface Design:** Building a clean and user-friendly interface was a crucial aspect of our project. We developed multiple prototypes of the UI to ensure that users can easily access and comprehend the information they need, providing a seamless and intuitive experience.
10. **Database Migration to Cloudant:** As our project evolved, we faced the challenge of migrating our database from MongoDB to Cloudant. This required careful planning and execution to ensure a smooth transition while maintaining data integrity and accessibility.
11. **Fallback Mechanism:** In cases where a word is not present in our database, we needed to implement a fallback mechanism to provide users with alternative sources of information. Extensive research and evaluation of available APIs led us to integrate with the API provided by dictionary.dev, ensuring comprehensive coverage and a reliable fallback option.

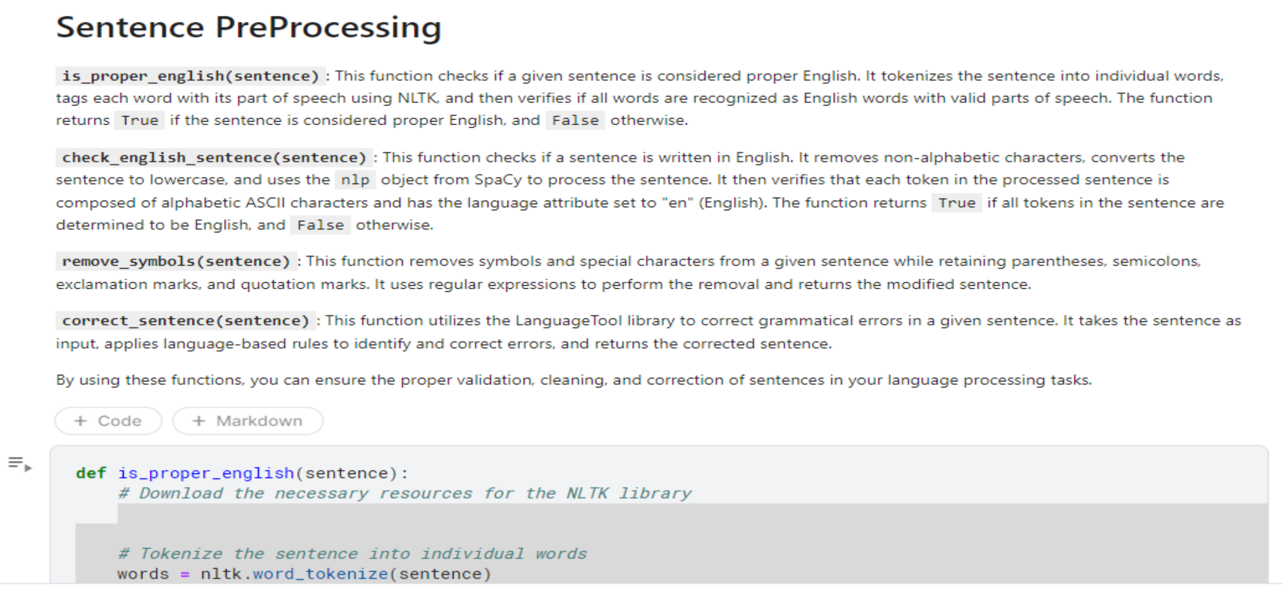
Preprocessing

During the preprocessing phase, we employed various Python packages such as NLTK and SpaCy to preprocess the textual data obtained from WordNet and Wiktionary. Since the data from Wiktionary was contributed by individuals and required formatting to match our chosen data structure, extensive preprocessing was necessary. We used NLTK to preprocess WordNet words and leveraged multiple Python libraries to handle the complexities of the textual data from Wiktionary.
The link for collection of data source is : Wiktionary English Data
Below I will give a brief description of the files used for preprocessing , for details refer to the markdown in the jupyter notebooks. The files can be found in dataPreprocessing directory.

Pre-processing WordNet Wiktionary:

PreProcessingWordNetWiktionary.ipynb is the notebook. It is responsible for preprocessing and dumping the preprocessed data to a file which we can import later to mongodb or cloudant. Detailed explanation for every block is in the markdowns.

Proccessed File: Data



Sentence PreProcessing

`is_proper_english(sentence)` : This function checks if a given sentence is considered proper English. It tokenizes the sentence into individual words, tags each word with its part of speech using NLTK, and then verifies if all words are recognized as English words with valid parts of speech. The function returns `True` if the sentence is considered proper English, and `False` otherwise.

`check_english_sentence(sentence)` : This function checks if a sentence is written in English. It removes non-alphabetic characters, converts the sentence to lowercase, and uses the `nlp` object from SpaCy to process the sentence. It then verifies that each token in the processed sentence is composed of alphabetic ASCII characters and has the language attribute set to "en" (English). The function returns `True` if all tokens in the sentence are determined to be English, and `False` otherwise.

`remove_symbols(sentence)` : This function removes symbols and special characters from a given sentence while retaining parentheses, semicolons, exclamation marks, and quotation marks. It uses regular expressions to perform the removal and returns the modified sentence.

`correct_sentence(sentence)` : This function utilizes the LanguageTool library to correct grammatical errors in a given sentence. It takes the sentence as input, applies language-based rules to identify and correct errors, and returns the corrected sentence.

By using these functions, you can ensure the proper validation, cleaning, and correction of sentences in your language processing tasks.

```
def is_proper_english(sentence):
    # Download the necessary resources for the NLTK library


    # Tokenize the sentence into individual words
    words = nltk.word_tokenize(sentence)
```

Parsed Wiktionary:

ParsedWiktionary.py is the notebook which will preprocess the entire Wiktionary data. It has the same functionality as previous file but it will export all the words in wiktionary into a file whether relevant or irrelevant.

Processed File : Data

Add Accepted Words:

AddAcceptedWords.py is used to retrieve words from database which are in **accepted** stage and would use the parsed wiktionary data in previous step to extract the word structure that we have already formatted according to our data structure and then load it in database. Also it would change state of the words added to **added.**
Since we moved to real time adding of words this is not in use and if put to use change the endpoint to the cloudant database or the database in use. Details of function and blocks in markdown.

Example Sentence Word Linking:

Solrsentencewordlinkage.py This file is responsible for linking each word with its corresponding sentences indexed in Solr. The goal is to establish a connection between words and the sentence IDs associated with them, enabling efficient querying of sentences for a given word.

To achieve this, the file loads all the words present in our database. The specific implementation details and workflow can be found in the accompanying notebook or code documentation.

In order to establish the link between words and sentences, the file utilizes Spacy, a natural language processing library. Spacy is employed to check the similarity between sentences and identify related or similar content. This similarity check plays a crucial role in associating sentences with their respective words.

Overall, the file's purpose is to create a connection between words and the corresponding sentence IDs in Solr, leveraging Spacy for sentence similarity analysis. This linkage enhances the efficiency and effectiveness of querying sentences based on specific words, enabling more accurate and relevant search results.

Solr Preprocessing:

The news dataset can be processed and uploaded to solr using the notebook file here.

Front-End

With a strong focus on maintainability and scalability, we have implemented a modular structure for our application. By breaking down our application into various components, we have achieved a highly modularized architecture. Each component represents a specific functionality or UI element, allowing for easier management and reuse of code. Below are the components with their functionality defined in detail.

Home.js:

Within home.js, there are the following components that are rendered:

On home page:

> **Header.js:**
>
> Within header.js, there are the following components being rendered:
>
> **MaininfoCard.js and InfoCard.js:**
> MainInfordCard contains the code for word of the day and trending words.
> Trending words is the list of words which are searched most often. The words are in a listed format. There is a Show More button which keeps showing five more words and the trending list keeps on going. Each trending word shows its meaning next to it. OnClick function is enabled which will open the meaning of the word directly.
> The Word of the day is one word selected at random from the Trending words list. There is a separate list which maintains the word of the day recorded so that it does not repeat. The word of the day has one meaning and one usage.

On meaning page:

WordSummary.js:

Pos.js:

Definitions.js:

On Statistics page:

Statistics.js :

Admin Authentication – LoginPage.js

Here we have two fields for username and password which when entered and clicked on submit will check if both the fields are filled, if not will notify the user to fill both fields and then check for the details which entered. If the login credentials are wrong, it will again notify the user as entered wrong details. Admin must enter the correct details in the login page in order to access the admin page.

About.js

We have a separate About page for this project which can be accessed by clicking the about us button on the home page. This page has some static information which talks about why we did this project and how much data we have in our database and how we are different than the other existing online dictionaries. The CSS for this page is in the About.css file.

Logo in header.js

We have a cse logo for this project which is at the top left of the home screen. It is stored in the icons folder of the assets folder which is inside the src folder. Necessary styling for this is present in the Header.css file in the same path.

Home.js: This is the home directory of our application. It is the parent component for our application and all child components are being rendered from here.

Some important functions used in this file are:

1. Async function WordHandler: This function fetches the data from the backend, and is also responsible for sending the language change code from the dropdown menu of the meanings page, to the backend.
2. Other functions include handling of different components such as Statistics, AddWord, WordNotFoud, About. These components are conditionally rendered, based on the 'OnClick' handler function which is tied to the buttons displayed on the screen.

Within home.js, there are the following components that are rendered:

1. On home page:

- **Header.js:** This is one of the main child components of our app, and houses several functions such as:
    - Search box with submit button
    - Search suggestions: We have implemented search suggestions as a dropdown, based on what the user types in the search box. The search uses a .filter()

function to filter the results and display them alphabetically. The results are then handled by the onInputChangeHandler and OnSubmitHandler function which has features such as using arrow keys of the keyboard to navigate the search results. The OnSubmitHandler handles the searched word and forwards this word as a prop to its child components.

  - Buttons displayed such as Statistics, AddWord, About.

Within header.js, there are the following components being rendered:

  - **Statistics.js:** The page shows all the corpus data available with us. The data is presented as bar graphs. The bar graph is made using react Plotly library. The graph contains the total words, total audios, total examples in the form of graph. We have also added one field which shows the total number of words searched and the total number of unique words searched which is updated dynamically. This component displays general statistics of the website such as 'Total Queries Handled', 'Total unique words searched', and shows a bar graph of our database with values: Total words, Total audios, Total Examples, Total Definitions.
  - **AddWord.js**: Incase our database does not have any new words that the user has searched, we have added the functionality to send this word for review to the admin. This word request is forwarded to the admin page, where the admin decides in a two-phase step to either accept or reject the requested word. If the word is accepted, it is added to our dictionary.

- **MaininfoCard.js and InfoCard.js:** These components are rendered on the home page and have two cards displayed: Word of the Day and Trending Words. MainInfordCard contains the code for word of the day and trending words. Trending words is the list of words which are searched most often. The words are in a listed format. There is a Show More button which keeps showing five more words and the trending list keeps on going. Each trending word shows its meaning next to it. OnClick function is enabled which will open the meaning of the word directly. The Word of the day is one word selected at random from the Trending words list. There is a separate list which maintains the word of the day recorded so that it does not repeat. The word of the day has one meaning and one usage.
-
  - **Word of the Day:** This card shows a random word from the list of most searched words. It is updated every day and a new word is selected from this list.
  - **Trending Words:** This card shows an interactive list of most searched words in our dictionary. The user can click on these words and is redirected to the meaning of the clicked word

2. **On meaning page:**

- **WordSummary.js:** This component displays the word as well as its IPA pronunciation. Furthermore, there is an audio button that sources the audio from the Google Text-to-Speech API and the user is also able to select different accents such as American English, Indian (Hindi), Japanese, Danish, British English. The word displayed is the prop passed from header.js from the searched word in the search box.

- **Pos.js:** This component renders the multiple meanings of the word that has been searched. Since each word might have different meanings as well as parts of speech, we have formatted this component to display the meaning of each part of speech, separately. Furthermore, each meaning also has its own pronunciation which we have sourced from Wiktionary. These are natural sounding, real audios. Also, each meaning has its own example usage sentence, based on the part of speech.
- **Definitions.js:** This component houses the general example usages/sentences of the searched word. It is a non-exhaustive list of examples which are sourced from news articles on the web. This enables the user to view the appropriate real world use case of each word.

Database

We have a non-relational database, MongoDB, with three collections: word_data, new_words, word_logs,word_of_the_day. These collections are used to store application-related data. In mongodb we are using collections and in cloudant we are using partitions to reduce query response time

Word_data:

This collection or partition will store the words and its corresponding data that will be presented to the user when he searches for a word.

The below provided JSON document represents a single entry in the "word_data" collection, which stores word-related data in MongoDB. Here's an explanation of the structure:

- - "_id": The unique identifier for the word document.
- - "word": The actual word being stored, e.g, "abime".
- - "usage": An array containing information about the word's usage.
  - - "pos": Part of speech (POS) of the word, e.g, "noun".
  - - "etymology_text": The text explaining the origin or etymology of the word.
  - - "etymology_number": A potential field for an etymology identifier.
  - - "definitions": An array of definitions associated with the word.
    - - "definition": An object containing the definition and its source.
      - - "gloss": The definition of the word.
      - - "source": The source from which the definition is obtained, e.g, "Wiktionary".

    - - "examples": An array of example sentences demonstrating the usage of the word.
    - - "audio": An array for storing audio-related information.
    - - "source": The source of the word's data e.g "Wiktionary".
- -"sentence_ids": Array of sentences linked which were scraped from news sources.
- "type": Just the collection name. Used for partitioning in cloudant.

This structure allows you to store detailed information about a word, including its usage, definitions, etymology, and sources.

Word_logs:

Everytime a word has been searched the following will be added to the logs collection or partition.

The below provided JSON document represents a single entry in the "word_logs" collection, which stores logs related to word operations in MongoDB. Here is an explanation of the structure:

- - "word": The word that the log entry refers to, in this case.
- - "wordFound": A boolean value indicating whether the word was found or not.
- - "date": The date when the log entry was created, in the format "YYYY/MM/DD" example, it is "2023/5/12".
- - "meaning": The meaning or definition of the word.
- - "pos": The part of speech (POS) of the word.
- - "count": The count of occurrences or frequency of the word. It is represented as a long number.
- -"type" : word_logs
- -"trendingWord": false

This structure allows you to track and log various details related to word operations, such as word searches, meanings, part of speech, occurrence count, and whether a word is trending or not. For word of the day, a random word would be chosen from past data and make its trendingWord to true and date to today's date for making it word of the day.

Review Words:

This collection or partion allows to add new words to the database and keep track of their state, such as whether they are in new, rejected, or successfully added. It also includes additional information like occurrence count and the meaning of the word. These details are valuable for managing new words in our database.

Here is an explanation of JSON with additional details:

- "_id": The unique identifier for the word entry, typically represented using an ObjectId. It serves as a primary key to distinguish each word entry within the database.
- "word": The word being added, such as "nearlywed". It represents the word being added to the database.
- "state": The state field is used to keep track of the current state of the word. It helps in managing the word's lifecycle and can have values like "Added," "New," "Rejected," or any other relevant states.
- "count": The count field represents the frequency or number of occurrences of the word. It is typically represented as a numeric value.
- "meaning": The meaning or definition associated with the word. It provides a description of the word's significance or interpretation.

ENDPOINTS & APIS:
These endpoints provide different functionalities for interacting with the application's word data, including retrieving data, managing new word additions, accepting or rejecting words, and adding logs and audio data. The details of these API's and usage can be found on our swaggerhub page which is SwaggerHub

1. Endpoint: /getData
- HTTP Method: GET
- Linked Function: getWordData
- Description: Retrieves word data from the database.

2. Endpoint: /getNewWords
- HTTP Method: GET
- Linked Function: getNewWords
- Description: Retrieves newly added words from the database.

4. Endpoint: /getTrendingWords
- HTTP Method: GET
- Linked Function: getTrendingWords
- Description: Retrieves a list of trending words from the database.

5. Endpoint: /getWordOfDay
- HTTP Method: GET
- Linked Function: getWordOfDay
- Description: Retrieves the "Word of the Day" from the database.

6. Endpoint: /acceptRejectWord
- HTTP Method: POST
- Linked Function: acceptRejectAddWord
- Description: Accepts or rejects a newly added word based on what admin will do and will automatically load the word data or associate its meaning with it automatically by integration with wikimedia api. This is the api which is responsible for state management and dynamically adding the word in the database if admin approves it.

8. Endpoint: /addNewWord
- HTTP Method: POST
- Linked Function: addNewWord
- Description: Adds a new word to the database.

9. Endpoint: /addWordLog
- HTTP Method: POST
- Linked Function: addWordLog
- Description: Adds a word log entry to the database.

10 Trigger:/ insertWordOfDay:
This is a trigger which will insert word of the day from previously searched top words in our database. Same trigger is present in cloudant also.

For cloudant we have not specifically built any api's and points . We are using built in data manipulation provided by **Cloudant** and have migrated the logic to our Node.js backend whose explanation we will see in backend.

Backend

Our backend is built using Node.js with the Express framework for routing. It serves requests from the front end and handles data from multiple sources. One of the main challenges we encountered was integrating data from different sources such as word meanings, example usages, and audio pronunciations. To overcome this, we leveraged the asynchronous nature of Node.js, allowing us to query these sources concurrently and improve overall efficiency. Throughout the codebase, we have implemented robust error handling techniques and thorough checks for edge cases to prevent application crashes even in the face of unforeseen errors.

We have established a standardized structure for sending data from the backend to the frontend. This is particularly important given the diverse origins of the data and the need to merge it seamlessly. The process starts by retrieving word and meaning data from IBM Cloudant database. If a meaning is found, the data is formatted according to the frontend requirements. Next, we fetch example usages of the word from a Solr text search engine. Prior to further processing, we perform some preprocessing tasks on the Solr data, such as removing special characters and filtering sentences based on length and specific keywords. These processed sentences are then appended to the response. Additionally, we process the International Phonetic Alphabet (IPA) information using a relevant Node.js module and include it in the response.

Routes

The routes folder contains all the necessary routes for our application. For example, the wordStats.js file includes routes for features like word of the day, trending words, and admin approvals for new words and meanings. The fetchWordLocal.js file contains routes for retrieving word pronunciation audio files.

Search

To support our search functionality, we utilized Solr, a text search engine. We processed approximately 87,500 articles from news sources such as US News and NY Times, using Python modules like sentence splitter for data preprocessing. These articles were divided into sentences and indexed in Solr, resulting in around 650,000 sentences from the corpus. To ensure fast and efficient access to the index, we hosted a Linux virtual machine on Google Cloud Platform (GCP) with ample RAM and storage capacity, minimizing latency. We also maintain source information, such as the website name, to cite specific websites when displaying example usage sentences to users.

Logs

Within our Node.js implementation, we have a logging mechanism that captures every word searched. This information is used to generate trending words, word of the day, and enables the potential development of analytics based on user search patterns.

We have implemented a function called handleDictionaryData to format data retrieved from the IBM Cloudant database. We also fetch relevant example usages from Solr and apply post-processing filters based on criteria like sentence length and word existence.

Another function, handleDictionaryAPI, is responsible for formatting data obtained from the dictionaryapi.dev website, which is an open-source and freely accessible API. We ensure consistency by formatting the API data according to our standard structure. We fetch example usages from Solr and append them to the response. Prior to querying the API, we utilize the IBM Cloudant database to minimize reliance on third-party sites.

For additional features, we implemented trending words functionality by sorting the word log based on frequency of occurrence, allowing us to display the top trending words. Similarly, we select a word of the day from the trending words list.

State Management for New Words

To facilitate an administrative review process, we have a two-stage workflow with different states representing initial review, final review, addition, and rejection of words. We provide an "add new words" route for users to suggest words for review by administrators. The "get new words" route fetches words that are currently in the review process across different states. The "admin word" route is the central component of the administrative process, handling state changes for words during initial and final reviews, as well as word acceptance or rejection. Additionally, the route fetches word meanings from Wiktionary to assist administrators in assessing the validity of the word, displaying this information in the final review page.

Google Pronunciation:

FetchwordLocal.js and index.js

In this project we are creating audio files for each word (with specific accent) exactly once. This file is created once a word is searched. To create this audio file, we are using two parameters, one is the word for which we need the pronunciation, and the second parameter is the accent. The default one we use here is “en-US” accent.

Using these values, we will be writing a ssml (Speech Synthesis Markup language) script. This markup language is like a mediator between the user and google audio. We can convey our requirements using these tags. Google text to speech is being used to generate the audios for the words searched.

First, we should create a google account and enable the text-to-speech and then get the Json file which is required to use this text-to-speech tool. I’m placing this file in the project backend directory, and it can be used in any other file by pointing towards it.

We also have 6 different types of accents for each word. They are en-US, en-GB, hi-IN, es-ES, ja-JP, da-DK. User is given an opportunity to select an accent from the dropdown. The audio is cached when user searches for a word and when we click on the audio button in the meanings page the stored audio file from the google_Audios folder will be retrieved. The storing name format for each word is word_accent.mp3 (example wonderful_en-US.mp3).

Since we are caching the audios the next time when the user search for the same word with the same accent the file is already present in the folder, so it is directly retrieved and doesn’t hit the Google's text-to –speech tool.

Admin authentication – index.js

Here when we get the request to get the details for the admin authentication, we send a json with the login name and password.

Deployment

We have used IBM Cloud Code Engine to deploy our website. Since code engine accepts docker images, we used Dockerfile to create images and then built the containers using the images.

We have built two different dockerfiles for our backend and frontend.

The backend dockerfile can be built using the docker build command with a tag:

docker build -t online-dict-backend:1.0 .

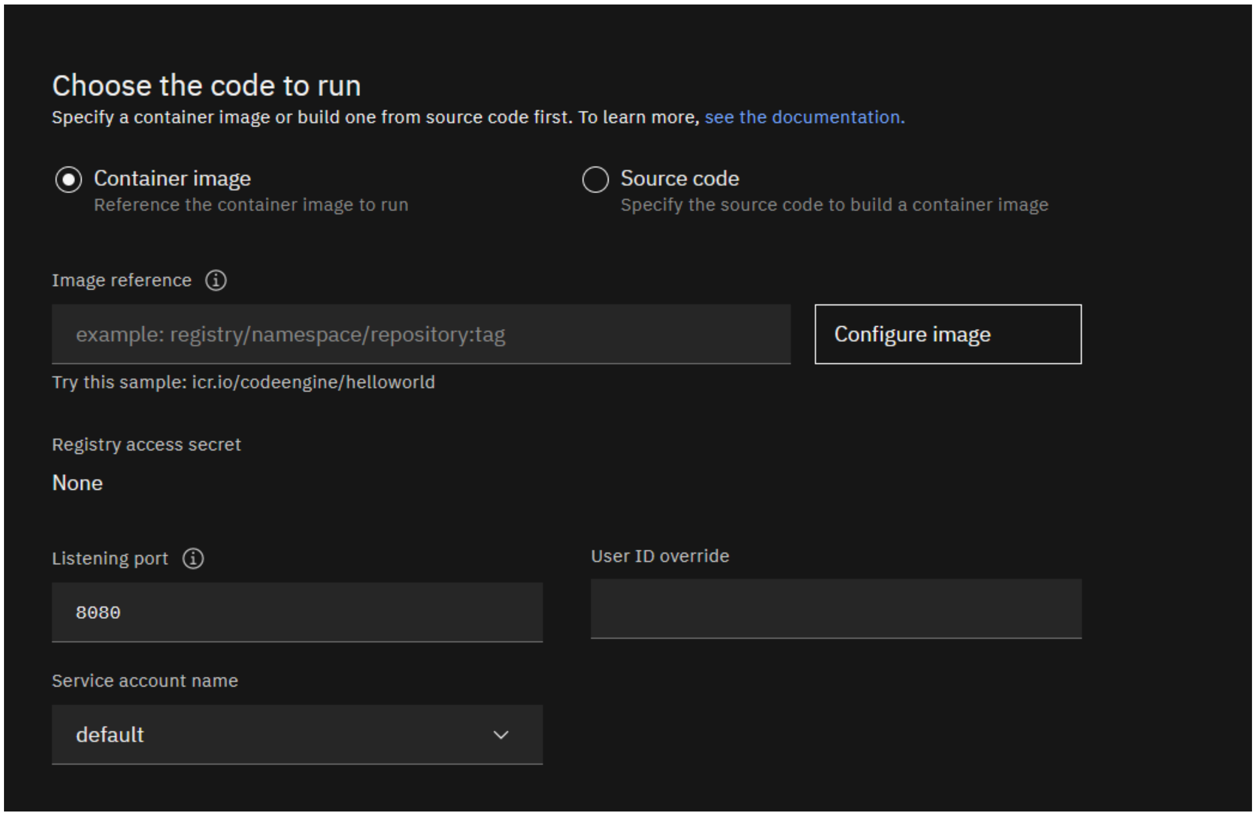
The built docker container can then be tagged with the command:

docker tag online-dict-backend:1.0 nightking80/online-dict-backend:1.0

The tagged container can then be pushed to docker hub:

docker push nightking80/online-dict-backend:1.0

After pushing the docker container to the docker hub, we can deploy it on code engine by referencing the docker image from the docker hub and changing the port number to 3001.



Choose the code to run

Specify a container image or build one from source code first. To learn more, see the documentation.

[✓ Container image] Reference the container image to run

Source code Specify the source code to build a container image

Image reference: example: registry/namespace/repository:tag

Try this sample: icr.io/codeengine/helloworld

Registry access secret: None

Listening port: 8080

User ID override:

Service account name: default

After the backend is deployed, the application URL can be copied and this URL should be pasted everywhere the front end is calling the APIs on the backend.

Similarly, the front end can also be deployed with port number set as 3000.

The cloudant database can be created using the data from parsed Wiktionary and uploading to the database with bulk operations.

Solr can be deployed on a VM and the URL of the VM can be inserted into the provided preprocessing script in order to insert all the sentences into solr instance.

To deploy a Cloudant resource on IBM Cloud and create a partitioned database, you can follow these steps:

1. **Log in to IBM Cloud:** Visit the IBM Cloud website and log in to your IBM Cloud account using your credentials.
2. **Create a Cloudant Service:** Once you're logged in, navigate to the IBM Cloud Catalog and search for "Cloudant" in the search bar. Select the Cloudant service from the available options.
3. **Configure the Cloudant Service:** On the Cloudant service page, choose the appropriate region and pricing plan based on your requirements. Provide a unique name for your Cloudant instance to identify it within your IBM Cloud account.
4. **Create a Partitioned Database:** After the Cloudant service is provisioned, you can access the Cloudant Dashboard. From the dashboard, click on your Cloudant instance to access the Cloudant management console.
5. **Configure Partitioning:** Once the database is created, you can configure partitioning for the database. Partitioning allows you to distribute your data across multiple shards for improved performance and scalability.
6. **Define Partition Keys:** Define the partition keys based on your data structure and access patterns. Partition keys determine how data is distributed across partitions. You can define multiple partition keys depending on your requirements.
7. **Load Data Using Curl or Requests :** Cloudant maximum size that can be uploaded is 8 MB

Discussion about the course:

The CSE 611 Project development course has given great insight into the Software development lifecycle.

During the project development course, I gained valuable insights and learned several important lessons that have enhanced my understanding of the software development process. Here are some key takeaways from the course:

1. Collaboration and Communication: Effective collaboration and communication are essential for successful project development. Working as part of a team, we learned the importance of clear and frequent communication among team members. Regular meetings, updates, and discussions helped us stay aligned, share ideas, and address any issues or challenges promptly.
2. Requirement Gathering and Analysis: Understanding and gathering accurate requirements is crucial for project success. We learned techniques for eliciting requirements from stakeholders, conducting thorough analysis, and documenting them clearly. This process helped us define the project scope, set realistic expectations, and avoid misunderstandings down the line.
3. Version Control and Collaboration Tools: The course introduced various version control systems and collaboration tools such as Git, GitHub. I learned how to use these tools to track changes, manage code repositories, collaborate with team members, and streamline project management tasks.
4. Testing and Quality Assurance: Ensuring the quality of software is essential, and I learned about different testing methodologies such as unit testing, integration testing, and user acceptance testing. I also gained an understanding of the importance of code reviews, continuous integration, and automated testing to catch bugs and improve overall software quality.
5. Documentation and Code Maintenance: Proper documentation is vital for project maintainability and knowledge sharing. I learned the significance of documenting code, APIs, and system architecture to facilitate future development and ease collaboration among team members. Additionally, I learned about best practices for code organization, modularization, and writing clean, readable, and maintainable code.
6. Time and Task Management: Managing project timelines and tasks efficiently is crucial for meeting deadlines and delivering projects successfully. The course provided insights into techniques such as creating project plans, setting realistic milestones, estimating task durations, and tracking progress. Effective time and task management helped us stay on schedule and manage workloads effectively.

Contributions:

1.Word Data Collection & Word Data Preprocessing-HAMZA: Understanding Confusing Wiktionary data and preprocessing the inconsistent dataset to a consistent suitable dataset. Imported the same In MongoDB.

2. MongoDB Data Import, API's, Endpoints & Triggers-HAMZA: Imported data to MongoDB, Created MongoDB API's, and Endpoints for data consumption as per our requirement. Handled all database related operations.

3.Swagger API-HAMZA: Created Swagger API Documentation to build a intuitive way for understanding our restful API's and interacting with it.

4 End To End Workflow and Scripting to Add a New Word-Hamza: Integrated with Wikimedia API and complex post processing of HTML data received from api to match the data format of our database.

5. **Setup of entire backend codebase in Node JS** - **Chris** : Handling website routing and calls to the different APIs like Cloudant, Solr, ApiDictionary.dev, MongoDB.

6. **Setup of SOLR instance - Chris** : Preprocessing huge news dataset. Utilized intelligent sentence reformatting tools to ingest best data in solr

7. Linking Solr to Words: Chris & Hamza: Linked Solr sentences to words to eliminate solr.

8. **Created post processing logic** – **Chris** : Example usage sentences formatted by removing special characters and removing unnecessary sentences.

9. **Handled deployment of project on the IBM Cloud** – **Chris** Building dockerfiles, understanding deployment strategies and keeping code up to date on cloud.

10. Google Audio Integration- AKHIL: Integrated with google audio and created support for six different accents.

11.**Admin Authentication Page and logo for the application** – **AKHIL**: The code for handling the admin authentication in both the backend and frontend. The logo for the application has been handled.

12. **Caching of Audio** – **AKHIL –** Caching of more than 20k unique term pronunciations to a folder for reusing them later.

13. **Testing the application** – **AKHIL –** Testing the whole application thoroughly and identified defects which have been reported to the team.

14 **Selenium Test Cases**- **Robin-** Created selenium test cases to automatically test front end.

15 **Mongo to Cloudant Migration- Hamza & Chris:** Imported all the data to cloudant. Created partitions to improve query , and migrated entire mongodb api logic to backend for cloudant.

16 **CSS For Complete Application** S**amar:** Implemented css to design a UI from which users could easily and concisely gain knowledge. Created CSS for 30 plus components.

Front- End:

Many people contributed to front end.D etailed contribution is mentioned below:

Hamza: Worked on setting up the entire application with basic layout and component Structure for different pages in the application. Integrated all the backend API's on front end. Managed the javascript for data processing and component rendering based on data in the front end.

Samar: Implemented   Javascript for all the components on front end. Worked on all the pages on the front end. Created all state management for pages.Integrated Google Audio and also manually tested front End.

Robin: Worked on Word of the day and trending words along with Samar. Completely Implemented statistics page reiteratively with integration with backend.

Feature wise contribution on the pages is mentioned below:

1. Word Of the Day: Robin, Samar.
2. Trending Words: Robin, Samar, Hamza.
3. Statistics: Robin, Akhil
4. Add a Word: Samar, Hamza
5. About: Akhil, Hamza
6. Admin Portal:
    - 6.1. Login: Akhil
    - 6.2. Initial Review: Samar, Hamza
    - 6.3. Final Review: Samar, Hamza
7. Word Search Suggestions: Samar
8. Word Search Logic: Hamza
9. Word Meanings Example Page: Hamza, Samar
10. Google Audio Front End: Samar, Akhil