Contents

[Code Comments (pending) 1](#_Toc164240277)

[Project flow diagram: 1](#_Toc164240278)

[Environment Configuration: 1](#_Toc164240279)

[Dependencies / Steps to install: 1](#_Toc164240280)

[Deployment Process: 1](#_Toc164240281)

[Versioning 1](#_Toc164240282)

[Schema Detail 2](#_Toc164240283)

[Entry / Exit point 2](#_Toc164240284)

[Debug Logs 2](#_Toc164240285)

[External APIs: 2](#_Toc164240286)

[React Components 6](#_Toc164240287)

[Other files: 17](#_Toc164240288)

## Code Comments (pending)

## Project flow diagram:

You can view flow diagram from [here](https://drive.google.com/file/d/1u8Dmfkg9EgYTGum5mKaG7Vbk_2Oqxw0b/view?usp=sharing)

## Environment Configuration:

The project utilizes a .env file to manage environment variables, specifically REACT\_APP\_OPERATION. This variable determines how tweet data is retrieved. In development mode (Dev), the application fetches data from a predefined URL, cleans and processes it before returning the results. This allows for mock data and easier development workflows. In production mode, the current implementation retrieves an empty list. This separation ensures sensitive information like API URLs are not stored directly in the code and facilitates configuration for different environments.

**Transitioning Between Dev and Prod** You can seamlessly transition the application from development to the production environment by making a simple change in your .env file. Update the REACT\_APP\_OPERATION variable as follows:

* **For Development:** REACT\_APP\_OPERATION=Dev
* **For Production:** REACT\_APP\_OPERATION=Production

## Dependencies / Steps to install:

**Frontend**

* Clone the project from [GitHub](https://github.com/MuneebMuhammad/social-sensing-frontend.git) repository.
* Install dependencies using **npm i**
* Run **npm start** command

**Backend**

## Deployment Process:

This project leverages GitHub Actions for a streamlined deployment process. Upon merging code into the main branch, the workflow automatically builds the application and deploys it to [Vercel](https://social-sensing-frontend.vercel.app/searchpage) for hosting. This automation ensures consistent, rapid updates and simplifies the process for maintaining a live version of the application.

## Versioning

Release management has not been implemented yet.

## Schema Detail

**User Model for MongoDB**

This document describes the User model for a MongoDB database created using Mongoose. The model defines the structure of each user document within a collection.

**Properties:**

* **Required Fields:**
  + firstName (String): Stores the user's first name.
  + lastName (String): Stores the user's last name.
  + email (String): Stores the user's email address.
  + password (String): Stores the user's password (hashed for security).
* **Optional Fields:**
  + companyName (String): Stores the user's company name (if applicable).
  + resetPasswordToken (String): Stores a temporary token used for password resets.
  + resetPasswordExpires (Date): Stores the expiration date for the password reset token.

**Saved Search Model for MongoDB**

This document describes the Saved Search model for a MongoDB database created using Mongoose. The model defines the structure of each saved search document within a collection.

**Properties:**

* **userId** (ObjectId): References a user document in the User collection using its unique identifier. This establishes a relationship between saved searches and the users who created them.
* **name** (String): Stores the name of the saved search, allowing users to identify it easily. This field is mandatory.
* **date** (Date): Stores the date and time the search was saved. Defaults to the current date and time.
* **region** (String): Stores the geographic region associated with the saved search. Defaults to "none" if not specified.
* **hashtags** (Array of Strings): Stores a list of hashtags included in the saved search criteria.
* **labels** (Array of Strings): Stores a list of user-defined labels for further categorizing the saved search.

## Entry / Exit point

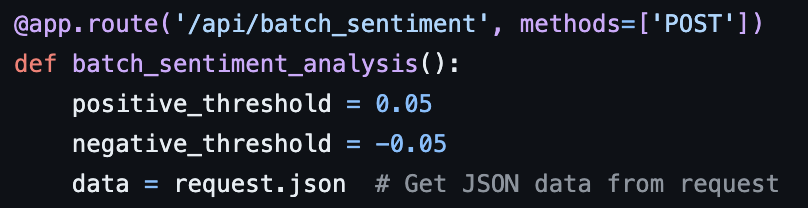
You can view entry / exit points through the [component tree](https://drive.google.com/file/d/1u8Dmfkg9EgYTGum5mKaG7Vbk_2Oqxw0b/view?usp=sharing).

## Debug Logs

Logging system has not been implemented yet.

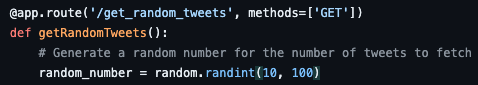
## External APIs:

1. Sentiment Analysis



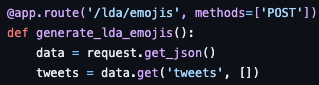
* 1. Description: Returns the sentiment of each text in a batch of text
  2. URL: <https://deploy-check-azure.vercel.app/api/batch_sentiment>
  3. Method: POST
  4. headers: {"Content-Type": "application/json"}
  5. body: tweetsText: array of strings each representing a tweet’s text
  6. response: array of string each representing a tweet’s sentiment
  7. Used at: ‘’src/contexts/dummyData.js”

1. Random Tweets



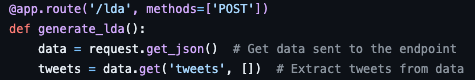
* 1. Description: Get random tweets from corpus in mongoDB
  2. URL: <https://lda-iwz8.onrender.com/get_random_tweets>
  3. Method: GET
  4. Response: array of object each representing a single tweet’s information
  5. Used at: ‘’src/contexts/dummyData.js”

1. Emoji Cloud



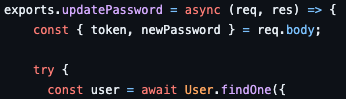
* 1. Description: Get word-cloud for emojis from tweet’s text
  2. URL: <https://lda-iwz8.onrender.com/lda/emojis>
  3. Method: POST
  4. body: tweets: array of string each representing a tweet’s text
  5. response: frequency and count pairs for each emoji
  6. Used at: “src/components/top-themes/emojicloud.js”

1. Word Cloud



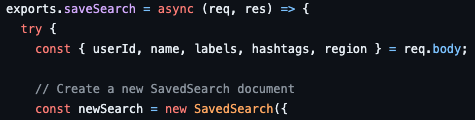
* 1. Description: Get word-cloud from tweet’s text
  2. URL: <https://lda-iwz8.onrender.com/lda>
  3. Method: POST
  4. body: tweets: array of string each representing a tweet’s text
  5. response: frequency and count pairs for each word
  6. Used at: “src/components/top-themes/workcloud.js”

1. Update Password



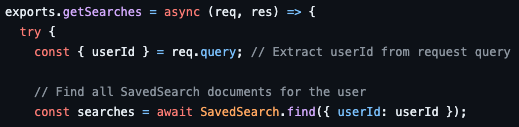
* 1. Description: Change the user password in forget password
  2. URL: <https://mongodb-server-eta.vercel.app/user/updatePassword>
  3. Method: POST
  4. body: newPassword, token
  5. Used at: “src/components/ChangePassword/ChangePassword.js”
  6. Response: Confirmation message

1. Save User Search



* 1. Description: Saves the user query in the database
  2. URL: <https://mongodb-server-eta.vercel.app/search/saveSearch>
  3. Method: POST
  4. body: userId, name, labels, hashtags, region: “None”
  5. Used at: “src/components/saveSearchModal/SaveSearchModal.component.js”
  6. Response: Confirmation message

1. Get User Searches



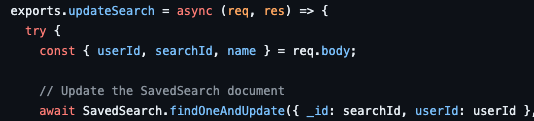
* 1. Description: Retrieves all the searches saved by a particular user
  2. URL: <https://mongodb-server-eta.vercel.app/search/getSearches?userId=id>
  3. Method: GET
  4. Used at: “src/components/mySearchesModal/MySearchesModal.component.js”
  5. Response: list of searches

1. Delete User’s Saved Search



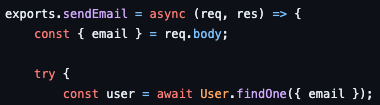
* 1. Description: Delete a particular saved search of a user
  2. URL: <https://mongodb-server-eta.vercel.app/search/deleteSearch>
  3. Method: DELETE
  4. Body: userId: userId, searchId: id
  5. Used at: “src/components/dashboard/Dashboard.component.js”
  6. Response: Confirmation message

1. Update User’s Saved Search



* 1. Description: Updates the content of a particular saved search of a user
  2. URL: <https://mongodb-server-eta.vercel.app/search/updateSearch>
  3. Method: PUT
  4. Body: userId, searchId, name
  5. Used at; “src/components/dashboard/Dashboard.component.js”
  6. Response: Confirmation message

1. Send Email



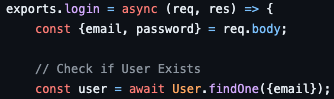
* 1. Description: Send verification email to User during sign up and forget password
  2. URL: [https://mongodb-server-eta.vercel.app/auth/sendEmail](https://mongodb-server-eta.vercel.app/search/updateSearch)
  3. Method: POST
  4. Body: email
  5. Used at: “src/components/CheckEmail/CheckEmail.js” and “src/components/ForgotPassword/ForgotPassword.js”
  6. Response: Confirmation message

1. Register User



* 1. Description: Sign up a user
  2. URL: <https://mongodb-server-eta.vercel.app/user/register>
  3. Method: POST
  4. Body: firstName, lastName, companyName, email, password
  5. Used at: “src/components/SignUp/signUp.component.js”
  6. Response: User document and token

1. Login User



* 1. Description: Log in a user
  2. URL: <https://mongodb-server-eta.vercel.app/user/login>
  3. Method: POST
  4. Body: email, password
  5. Used at: “src/components/SignIn/signIn.component.js”
  6. Response: token

## React Components

**Card**

**Description:**

This component renders a reusable card with a title, informative text, an optional percentage value, and icons that appear on hover.

**Path:** src\components\card\Card.js

**Input:**

* image: (string) URL of the image to display on the top left corner of the card.
* title: (string) Title text to display within the card.
* infoText: (string) Informative text displayed within the card.
* percentage: (string) Optional percentage value displayed in the bottom right corner.

**Output:**

The component renders a styled card with the provided props. The card includes:

* An image in the top left corner.
* A title displayed next to the top left image.
* Informative text displayed prominently within the card body.
* An optional percentage value and trend indicator displayed in the bottom right corner.
* Three hidden icons (maximize, logout, and more options) that become visible on hover.

**Chart**

**Description:**

Renders a line chart using Chart.js.

**Path:** src\components\chart\Chart.js

**Inputs:**

* title: Chart title text.
* data: Chart data formatted for Chart.js line charts.
* queryMatches: Array of query strings matched to the data.

**Outputs:**

* Styled card with a title and a line chart.

**Interactions:**

* Clicking a data point updates context values for top results filtering and navigates to the /topResults route.

**Context Usage:**

* Accesses and updates state values from TopResultsFilterContext.

**Colored Checkbox**

**Description**

This component renders a styled checkbox with a custom design.

**Path**: src\components\coloredCheckbox\ColoredCheckbox.styles.js

**Inputs**

* className: (string) An optional class name to be applied to the checkbox container.
* checked: (bool) A boolean value indicating the checked state of the checkbox.
* props: (object) Any additional props to be passed to the underlying hidden checkbox input.

**Outputs**

The component renders a styled checkbox element that:

* Displays a checkmark icon when checked.
* Changes background color based on the checked state (blue when checked, white otherwise).

**Comparison Card**

**Description:**

UI element designed to display side-by-side comparisons between different items or data points. It provides a visually appealing and structured layout for presenting information in a clear and concise manner.

**Path:** src\components\comparison-card\ComparisonCard.jsx

**Input:**

* title: (Required) A string value that defines the title of the comparison card, providing context for the data displayed within.

**Output:**

The component renders a visually formatted card with the following elements:

* **Title:** The provided title string displayed prominently at the top of the card.
* **Comparison Items:** A series of vertically arranged sections within the card, each representing a single item or data point for comparison. These sections include:
  + **Info Text:** A highlighted value displayed in a larger font size and a custom color (based on the item.color prop within the data).
  + **Muted Text:** A secondary text label providing additional information about the corresponding item, typically displayed in a smaller font size and a muted color.
* **Danger Icon:** A subtle danger icon represented by the /danger-circle.svg image positioned in the top-right corner of the card.

**Date Range**

**Description:**

Renders a styled button that functions as a date range picker input. It displays either the selected date or a default "Select Date" message. Clicking the button triggers the onClick callback function, presumably to open a date range picker UI.

**Path:** src\components\date-range\DateRange.component.jsx

**Inputs:**

* value (optional): A string representing the currently selected date range in a format suitable for display.
* onClick (function): A callback function that executes when the button is clicked.

**Outputs:**

* Renders a styled button element with the provided value or the default text "Select Date."

**Edit Compare Keyword Modal**

**Description:**

Renders a modal dialog for editing a compare keyword within your application. It allows users to modify both the label (displayed name) and the underlying query associated with the keyword.

**Path:** src\components\editCompareKeywordModal\EditCompareKeywordModal.component.js

**Inputs:**

* show (boolean): Prop passed from a parent component to control the visibility of the modal (true for open, false for closed).
* handleClose (function): A callback function provided by the parent component to handle closing the modal.
* currentHashtag (string): The initial label and query value for the keyword being edited. This is presumably passed from the parent component.

**Outputs:**

* Renders a modal dialog with two input fields:
  + Label: Allows users to edit the displayed name for the compare keyword.
  + Query: Allows users to modify the underlying search query associated with the keyword.
* Provides a "Save" button to trigger saving the edits.

**Search Items**

**Description:**

* Renders an individual saved search item within a list or collection of searches.
* Grants users abilities to view details, edit the name, delete the search, or reuse it for searching.

**Path:** src\components\MySearchesItem\MySearchesItem.component.js

**Inputs:**

* item (object): Represents a single saved search, containing properties like name, hashtags, date, and labels.
* handleEditSearch (function): A callback function to handle renaming a saved search.
* handleDeleteSearch (function): A callback function to handle deleting a saved search.

**Outputs:**

* Renders a visual item comprising:
  + The saved search's name.
  + Hashtags related to the search.
  + The date the search was saved.
  + Buttons for editing, deleting, or initiating a new search using those same parameters.

**Searches Modal**

**Description**

Renders a modal that displays a user's saved searches. It allows users to view, edit, and delete their saved searches. The component also provides pagination functionality to navigate through a large number of saved searches.

**Path** src\components\mySearchesModal\MySearchesModal.component.js

**Inputs**

* show: A boolean value that determines whether the modal is visible (true) or hidden (false).
* handleClose: A function that handles closing the modal.
* saveSearches: An array of objects containing the user's saved searches..
* handleEditSearch: A function that is called when a user edits a saved search. This function should take the search object as an argument.
* handleDeleteSearch: A function that is called when a user deletes a saved search. This function should take the search object's id as an argument.

**Outputs**

The component renders a modal with the following elements:

* A header that displays the title "My Searches".
* A body that displays a list of the user's saved searches using the MySearchesItem component.
* A footer that contains:
  + A dropdown menu to select the number of searches displayed per page.
  + A pagination component to navigate through multiple pages of saved searches.

**Onboarding Card**

**Description**

Renders a card layout commonly used in onboarding processes. It displays an image on the left side and content on the right side. It also includes a header section with a logo and a language selector dropdown.

**Path** src\components\onBoardingCard\onBoardingCard.component.js

**Inputs**

* children: This prop allows you to pass any React component(s) to be rendered within the right side content area of the card. This provides flexibility to customize the content displayed in the onboarding card.

**Outputs**

The component renders a card with the following elements:

* Left side: An image specified by the onboardingImage import.
* Right side:
  + Header:
    - Logo: An image specified by the waleeLogo import.
    - Language selector dropdown: A dropdown menu.
  + Content area: This area will be populated by the React component(s) passed through the children prop.

**Refresh Button**

**Description:**

This component renders a styled button that can be used to trigger a refresh action in your React application. It provides a visually appealing and interactive way for users to initiate a data refresh or any other relevant action.

**Path:** src\components\refreshBtn\RefreshBtn.component.js.

**Inputs:**

* The component doesn't accept any explicit props to customize its appearance or behavior. However, you can adjust the styles within the RefreshButton styled component definition in RefreshBtn.styles.js or RefreshBtn.styles.ts to modify its properties like:
  + background-color
  + border
  + cursor
  + font-size
  + color:hover (hover state color)

**Outputs:**

* The component renders a styled button element visually representing a refresh action. When clicked, it can be used to trigger a custom function or event in your application to perform the desired refresh logic.

**Result Card**

**Description:**

ResultCard, renders a visually appealing card displaying social media post information and relevant metrics. It takes in profile data and additional metrics as props and displays them in a structured format.

**Path:** src\components\Results\resultCard.js

**Inputs (Props):**

* profileData: An object containing information about the social media post, including:
  + profileImage: (string) URL of the profile picture
  + name: (string) Name of the profile
  + handle: (string) Username or handle of the profile
  + sharedImage: (string) URL of a shared image (optional)
  + content: (string) Text content of the social media post
  + matches: (number) Number of matches (relevance to a specific search?)
  + timePublished: (string) Time the post was published
  + location: (string) Location associated with the post (optional)
  + platform: (string) Social media platform where the post originated
  + sentiment: (string) Sentiment of the post ("Positive", "Negative", or "Neutral")
  + reach: (number) Reach of the post (number of people who saw it)
  + engagement: (number) Engagement with the post (likes, comments, etc.)
  + trending: (number) Whether the post is trending (optional)
* additionalMetrics: An object containing additional metrics about the post (optional):
  + shares: (number) Number of times the post was shared
  + hearts: (number) Number of likes or reactions on the post
  + users: (number) Number of users who interacted with the post (optional)

**Outputs:**

The component renders a styled card containing the following sections:

* Profile section with profile picture, name, handle, and optional shared image information
* Content section displaying the text content of the post
* Matches section (number of matches, purpose unclear without context)
* Additional information section with publication time, location (if provided), and platform
* Sentiment section with an icon representing the sentiment of the post ("Positive", "Negative", or "Neutral") and a label
* Metrics section with three columns displaying:
  + Reach (number of people who saw the post)
  + Engagement (number of likes, comments, etc.)
  + Trending score (optional)
* Small metrics section (optional) with icons and values for shares, likes (hearts), and users who interacted (optional)

**Saved Searches**

**Description:**

Renders a section displaying a user's saved searches within a social listening or analytics dashboard. It provides a quick overview of some saved searches and a button to open a modal for viewing and managing all saved searches.

**Path:** src\components\savedsearches\SavedSearches.component.js.

**Inputs:**

* style (optional): An object containing inline styles to be applied to the container element.

**Outputs:**

* Renders a styled container element with the following sections:
  + Heading: "Saved Searches"
  + SearchList: An unordered list displaying a limited number of saved search items
    - Each SearchItem displays:
      * SearchItemHeading: The name of the saved search (e.g., "Lahore")
      * Mentions count (e.g., "2k Mentions")
  + ViewAllButton: A button labeled "View All" that triggers the opening of the modal for managing all saved searches.
  + MySearchModal component (imported): This modal component is responsible for displaying and managing all saved searches. It's controlled by the showMySeachesModal state and receives props for handling close, editing, and deleting searches.

**Internal State:**

* showMySearchesModal (boolean): Controls the visibility of the MySearchModal component.
* saveSearches (array): An array of objects representing the saved searches.

**Save Search Modal**

**Description:**

Renders a modal dialog that allows users to save a set of search filters and queries within a social listening or analytics application. It provides a name input field, displays the selected queries for comparison, and offers buttons to save the search or close the modal.

**Path:** src\components\saveSearchModal\SaveSearchModal.component.js

**Inputs:**

* show (boolean): Determines the visibility of the modal (true for open, false for closed).
* handleClose (function): A function to be called when the user clicks the close button or the modal backdrop.
* addToDataset (function): A function provided by the parent component to handle saving the search data (name and filters).

**Outputs:**

* Renders a modal dialog with the following elements:
  + Modal header with title "Save Search" and a close button
  + Modal body:
    - Label "Name"
    - Input field for entering a name for the saved search
    - Label "Compared Queries"
    - List of the currently selected queries for comparison (fetched from context)
  + Modal footer:
    - "Save" button that triggers handleSaveSearch on click

**Internal State:**

* name (string): Stores the user-entered name for the saved search.

**Search Bar**

**Description:**

This component renders a search bar element for a social listening or analytics application. It provides a location dropdown, a search input field, and a search icon button.

**Path:** src\components\searchBar\SearchBar.component.js

**Inputs (Props):**

* style (optional): An object containing inline styles to be applied to the container element.
* onClick (function): A function to be called when the user clicks the search icon.
* onChange (function): A function to be called when the user types in the search input field.

**Outputs:**

* Renders a styled container element with the following elements:
  + WorldwideDropdown component (imported): component for selecting a location.
  + SearchInput: An input field with a placeholder "Search Hashtag, Brand or Event" for users to enter their search query.
  + SearchIcon: An image button.

**Sentiments Card**

**Description:**

This component renders a visually appealing card displaying sentiment analysis data for a specific topic. It displays a title, two sentiment categories (happy and sad) with icons, and their corresponding values.

**Path:** src\components\sentiments-card\SentimentsCard.js

**Inputs (Props):**

* title (string): The title of the sentiment analysis (e.g., "Social Media Sentiment").
* happy (number or string): The value representing the positive sentiment.
* sad (number or string): The value representing the negative sentiment.

**Outputs:**

* Renders a styled CardContainer with the following elements:
  + TopRow: Displays the title and a danger circle icon for negative sentiment.
  + BottomRow: Displays two sections for happy and sad sentiments:
    - An icon (smile for happy, frown for sad)
    - The corresponding value for each sentiment (happy and sad)

**Styled Components:**

* CardContainer: Defines the main card element with styles like flexbox layout, background color, border radius, and hover effects.
* TopRow: Styles a row for the title and icon.
* BottomRow: Styles a row for displaying happy and sad sentiments.
* Img: Styles the image elements for icons (danger circle, smile, frown).
* Title: Styles the title text with font size, weight, and color.

**Top Themes**

**Description:**

Renders a section for visualizing top themes within a social listening or analytics application. It provides a title, a danger circle icon for negative sentiment, and a dropdown menu to select the type of theme to display (Top Theme, Hashtags, Accounts, Bios, or Emojis). Based on the user's selection, it conditionally renders either a WordCloud component or an EmojiCloud component

**Path:** src\components\top-themes\TopThemes.js

**Inputs (Props):**

* timeRange (string or object):  represents the selected time range for the data visualization

**Outputs:**

* Renders a styled Container with the following elements:
  + OuterRow: A row with two sections:
    - Title "Top Themes" and a danger circle icon.
    - A dropdown menu (CustomSelect) to select the theme type.
  + WordCloudStyle: A styled container for conditionally rendering the word cloud or emoji cloud component.

**Trending Table**

**Description:**

This component renders a table displaying trending hashtags within a social listening or analytics application. It shows the hashtag name, a "New" tag for newly trending hashtags, or a percentage change for established trends. Additionally, it includes a header row indicating "Trending Hashtags" and the time period (e.g., Last 24 hours).

**Path:** src\components\trending\_table\trending\_table.component.js

**Inputs (Props):**

* style (optional): An object containing inline styles to be applied to the main table element.

**Outputs:**

* Renders a styled Table element with the following components:
  + TableHeader: Header section of the table.
    - HeaderRow: A row containing two header cells.
      * TableHeaderCellPrimary: Displays the primary header text "Trending Hashtags" with a trend icon.
      * TableHeaderCellSecondary: Displays the secondary header text indicating the time period (e.g., Last 24 hours).
  + tbody: Body section of the table containing table rows for each hashtag.
    - TableRow: Individual rows for each hashtag data.
      * TableCell: Cells for the hashtag name and its associated value (New tag or percentage change).

## Other files:

1. dummyData.js
   1. Description: Generates data for a keyword from mongoDB tweets corpus
   2. Path: src/contexts/dummyData.js
2. CompareKeyword.context
   1. Description: A react context to store the generated data regarding each keyword searched
   2. Path: src/contexts/CompareKeyword.context.js
3. SavedSearches.context.js
   1. Description: A react context to store the saved searches of a user
   2. Path: src/contexts/SavedSearches.context.js
4. TopResultsFilter.context.js
   1. Description: A react context to store the filters for showing the tweets result on Result page.
   2. Path: src/contexts/TopResultsFilter.context.js