InsightStream: Navigate the News Landscape **(React Application)**

Introduction:

Project Tile: InsightStream: Navigate the News Landscape

Team Members: NAVEEN KUMAR S (Team leader),
MOHANNATH S (Team member),
KISHORE S (Team member),
KISHORE R(Team member).

Project Overview:

Purpose:

Description:

Welcome to the cutting-edge frontier of news exploration with InsightStream! Our revolutionary web application is meticulously crafted to transcend the boundaries of traditional news consumption, catering to the diverse interests of both avid news enthusiasts and seasoned information professionals. With an emphasis on an intuitive user interface and a robust feature set, InsightStream is poised to redefine the entire news discovery and consumption process.

Designed with a commitment to user-friendly aesthetics, InsightStream immerses users in an unparalleled journalistic adventure. Navigate seamlessly through a vast expanse of news categories with features such as dynamic search, effortlessly bringing you the latest and most relevant stories from around the world.

Feature:

From those seeking the latest headlines to seasoned news connoisseurs, InsightStream embraces a diverse audience, fostering a dynamic community united by a shared passion for staying informed. Our vision is to reshape how users interact with news, presenting a platform that not only delivers breaking stories but also encourages collaboration and sharing within the vibrant news community.

Embark on this informative journey with us, where innovation seamlessly intertwines with journalistic tradition. Every click within InsightStream propels you closer to a realm of global happenings and perspectives. Join us and experience the evolution of news consumption, where each feature is meticulously crafted to offer a glimpse into the future of staying informed.

Elevate your news exploration with InsightStream, where every headline becomes a gateway to a world of information waiting to be discovered and understood. Trust InsightStream to be your reliable companion on the journey of staying connected with the

pulse of the world.

Scenario Based Intro:

Suppose you're rushing home after work, phone clutched in your hand. Today's been a whirlwind, and you have no idea what's happening in the world. Suddenly, you remember InsightStream , the innovative app you downloaded that promised to revolutionize your news experience. Intrigued, you open the app. Images flash across the screen – breaking headlines, in-depth articles, diverse categories. This isn't your typical news feed. InsightStream feels...different. Intrigued, you tap a category and dive in, ready to explore the future of staying informed.

Architecture:

Component Structure:

The user experience starts with the InsightStream web application's UI, likely built with a framework like React or Vue.js for a smooth, single-page experience. This UI interacts with an API client specifically designed for InsightStream. This client handles communication with the backend, but with a twist: it leverages Rapid API, a platform providing access to various external APIs. This suggests InsightStream might integrate external data feeds or functionalities through Rapid API, enriching the user experience without building everything from scratch.

Project Goals and Objectives:

The primary objective of InsightStream is to establish a user-friendly platform tailored for individuals who are passionate about staying informed, exploring diverse news topics, and accessing the latest updates.

Our key goals include:

✔ **User-Friendly Experience:** Develop an interface that is intuitive and easy to navigate, ensuring users can effortlessly access, save, and share their preferred news articles. ✔ **Comprehensive News Management:** Provide robust features for organizing and managing news content, incorporating advanced search options for a personalized news experience.

✔ **Technology Stack:** Employ cutting-edge web development technologies, such as React.js, to ensure an efficient and enjoyable user interface.

State Management:

✔ **News from API Sources:** Access a vast library of global news spanning various categories and interests, ensuring a well-rounded coverage of current affairs. ✔ **Visual**

News Exploration: Discover breaking stories and explore different news categories through curated image galleries, enhancing the visual appeal of news discovery.
✔ **Intuitive Design:** Navigate the application seamlessly with a clean, modern interface designed for optimal user experience and clarity in information presentation. ✔ **Advanced Search Feature:** Easily access news articles on specific topics through a powerful search feature, providing users with tailored news content based on their interests.

Routing Structure

Using **React Router**, the routing structure is simple:

1. **Setup**:

Define routes using `<Routes>` and `<Route>` components.

```
<div className="App">
    <NavbarComponent />
    <Routes>
      <Route exact path="/" element={<Home/>} />
      <Route exact path="/category/:id" element={<CategoryPage/>} />
      <Route exact path="/news/:id" element={<NewsPage/>} />
    </Routes>
    <Footer />
</div>
```

SET INSTRUCTION:

PREREQUISITES AND INSTALLATION:

Here are the key prerequisites for developing a frontend application using

React.js: ✔ **Node.js and npm**:

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

- Download: https://nodejs.org/en/download/
- Installation instructions: https://nodejs.org/en/download/package-manager/

✔ **React.js**:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to

build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

- Create a new React app:

```
npx create-react-app my-react-app
```

Replace my-react-app with your preferred project name.

- Navigate to the project directory:

```
cd my-react-app
```

- Running the React App:

With the React app created, you can now start the development server and see your React application in action.

- Start the development server:

```
npm start
```

This command launches the development server, and you can access your React app at http://localhost:3000 in your web browser.

✔ **HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

✔**Version Control**: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

• Git: Download and installation instructions can be found at: https://git-scm.com/downloads

✔ **Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from https://code.visualstudio.com/download • Sublime Text: Download from https://www.sublimetext.com/download
• WebStorm: Download from https://www.jetbrains.com/webstorm/download

To install and run the Application project from google drive:

Follow below steps:

✔ **Get the code:**

• Download the code from the drive link given below:

https://drive.google.com/drive/folders/1tDoSwd-1I3HsPJ9_92MnZTUtteeda-hL?usp=sharing

Install Dependencies:

• Navigate into the cloned repository directory and install libraries:

```
cd news-app-react
npm install
```

✔ **Start the Development Server**:

• To start the development server, execute the following command:

```
npm start
```

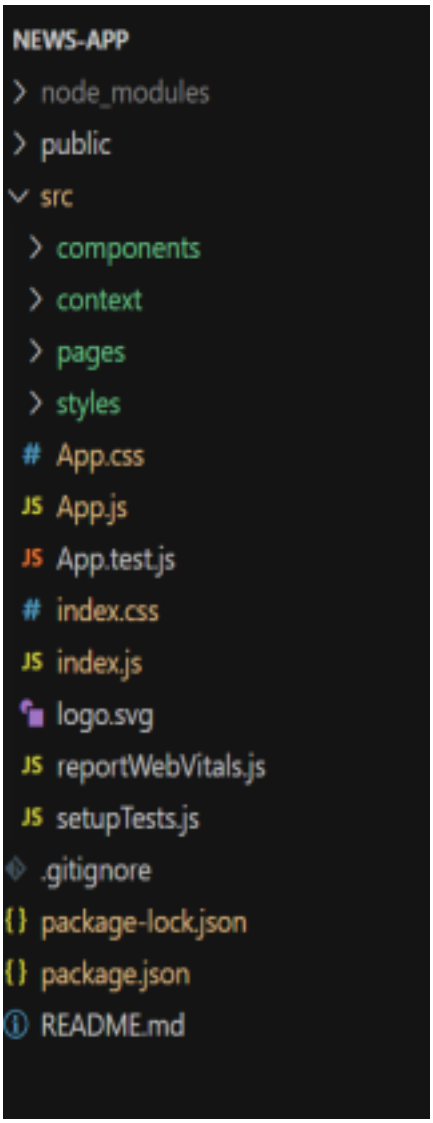
Access the App:

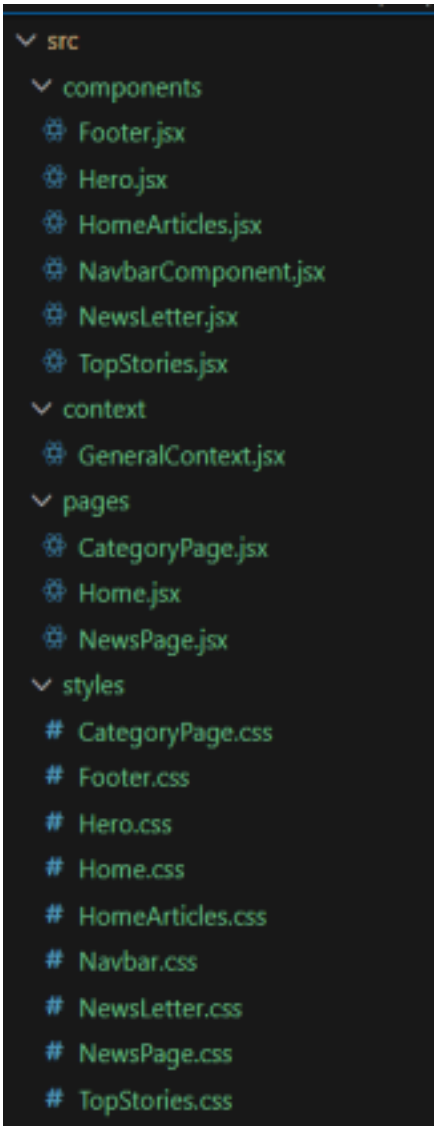
• Open your web browser and navigate to http://localhost:3000.

• You should see the applications homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the application on your local machine. You can now proceed with further customization, development, and testing as needed.

Folder structure:





In this project, we've split the files into 4 major folders, *Components, Context, Pages and Styles.* In the pages folder, we store the files that acts as pages at different URLs in the application. The components folder stores all the files, that returns the small components in the application. The context Api will be coded in the context folder. All the styling css files will be stored in the styles folder.

Project Flow:

Project demo:

Before starting to work on this project, let's see the demo.

Demo link:

https://drive.google.com/file/d/1i8y09FiMk7QM0akH3my10OBWqXH-8dNh/view?usp=sharing Use the code in:

https://drive.google.com/drive/folders/1tDoSwd-1I3HsPJ9_92MnZTUtteeda-hL?usp=sharing

Milestone 1: Project setup and configuration.

- **Installation of required tools**:

  To build InsightStream, we'll need a developer's toolkit. We'll use React.js for the interactive interface, React Router Dom for seamless navigation, and Axios to fetch news data. For visual design, we'll choose either Bootstrap or Tailwind CSS for pre-built styles and icons.

  Open the project folder to install necessary tools. In this project, we use:

  - o React Js
  - o React Router Dom
  - o React Icons
  - o Bootstrap/tailwind css
  - o Axios

- For further reference, use the following resources
  - o https://react.dev/learn/installation
  - o https://react-bootstrap-v4.netlify.app/getting-started/introduction/ o https://axios-http.com/docs/intro
  - o https://reactrouter.com/en/main/start/tutorial

Milestone 2: Project Development

- ❖ Setup the Routing paths

  Setup the clear routing paths to access various files in the application.

```
<div className="App">
    <NavbarComponent />
    <Routes>
      <Route exact path="/" element={<Home/>} />
      <Route exact path="/category/:id" element={<CategoryPage/>} />
      <Route exact path="/news/:id" element={<NewsPage/>} />
    </Routes>
    <Footer />
</div>
```

- ❖ Develop the Navbar and Hero components
- ❖ Code the popular categories components and fetch the categories from ***newsapi***.
- ❖ Also, add the trending news in the home page.
- ❖ Additionally, we can add the component to subscribe for the newsletter and the footer.

❖ Now, develop the category page to display various news articles under the different categories.

Important Code snips:

Fetching Top/Trending news

With the API request, we fetch the trending news articles.

The code snippet shows a function written in Python called fetchTopNews that fetches news articles from an API. Here's a breakdown of the code:

Async function fetchTopNews:

The code defines an asynchronous function named fetchTopNews. An asynchronous function is used to handle asynchronous operations, such as making API requests that take time to complete.

try...catch block:

- The try...catch block is used to handle the API request.
- The try block contains the code that attempts to fetch data from the API using axios.get.
- axios is an external Python library for making HTTP requests. If you don't already use Axios in your project, you'll need to install it using a package manager like pip.
- The .get method makes a GET request to the specified URL.

API URL:

The URL used in the API request is 'https://newsapi.org/v2/everything?q=popular&apiKey=37306aca596542f0a8402978de3d4224'.

This is likely a specific API endpoint that returns popular news articles. You might need to replace this URL with the actual endpoint you want to use depending on the API you're using. Replace '37306aca596542f0a8402978de3d4224' with a placeholder instructing users to replace it with their own API key.

Error Handling (catch block):

The catch block handles any errors that might occur during the API request. If there's an error, it's logged to the console using console.error(error).

Setting State (then block not shown):

The .then method (not shown in the code snippet) is likely used to process the fetched data after a successful API request.

In this case, it likely updates a state variable named topNews (based on the function name fetchTopNews) with the fetched news articles. This state variable might be used to display the news articles in a user interface.

Fetching news by search/category

With the specific category or search keyword, we use API request to fetch all the news articles related to that.

The code snippet shows a function called get_news_articles that fetches news articles from a news API. Here's a breakdown of the code:

Imports:

The code starts by importing the requests library. The requests library is a popular Python library for making HTTP requests. If you don't already have it installed in your project, you can install it using pip install requests.

API Key:

The line API_KEY = 'YOUR_API_KEY' defines a variable named API_KEY and assigns it a placeholder value 'YOUR_API_KEY'. You should replace this with a placeholder instructing users to replace it with their own API key obtained from the news API provider they want to use.

Function Definition (get_news_articles):

The code defines a function named get_news_articles that takes two parameters:

- query: This parameter is likely a string representing the search query for news articles.
- source: This parameter is likely a string representing the news source (e.g., 'bbc-news', 'cnn').

Building the API Request URL:

The line url = f'https://newsapi.org/v2/everything?q={query}&apiKey={API_KEY}' constructs the URL for the API request using a formatted string literal (f-string).

The URL includes the following parts:

- Base URL: https://newsapi.org/v2/everything
- Query parameters:
- q: This parameter is set to the query argument passed to the function.
- apiKey: This parameter is set to the API_KEY variable, which should contain the user's API key.

Making the API Request (requests.get):

The line response = requests.get(url) sends a GET request to the API URL constructed earlier. The requests.get function from the requests library is used to make the HTTP request. The response from the API is stored in the response variable.

Error Handling (try...except block):

- The try...except block is used to handle potential errors during the API request.
- The try block contains the code that attempts to fetch data from the API using requests.get(url).
- The except block handles any exceptions that might occur during the request, such as network errors or invalid API responses. In this case, it prints an error message to the console using print(f'Error fetching news articles: {e}').

Running the Application:

After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js

Here are some of the screenshots of the application.

Hero components

In the hero component, the trending news articles are displayed. It is to highlight them. Apart from that, the search bar is also available to search for various articles and categories.

Component Documentation

1. **Key Components**

Home:

Purpose: Displays the main landing page of the application.

Props: None.

```
function Home() {
  return <h1>Welcome to the Home Page</h1>;
}
```

About:

Purpose: Displays information about the app or company.

Props: None.

```
function About() {
  return <p>This is the About Page</p>;
}
```

UserProfile:

Purpose: Displays user information based on a dynamic `id` parameter.

Props:

`id`: A dynamic route parameter used to fetch and display user details.

```
function UserProfile({ id }) {
  return <div>User Profile for ID: {id}</div>;
}
```

Dashboard:

Purpose: Displays a dashboard with navigation to other features like settings.

Props: None.

```
function Dashboard() {
  return <div>Welcome to the Dashboard</div>;
}
```

---

2. Reusable Components

Button:

Purpose: A reusable button component for various actions across the app.

Props:

`text`: The label displayed on the button.

`onClick`: The function to call when the button is clicked.

`type`: Button style (`primary`, `secondary`, `danger`).

```
function Button({ text, onClick, type }) {
  return <button className={type} onClick={onClick}>{text}</button>;
}
```

Card:

Purpose: A reusable card component for displaying content in a box with a title and description.

Props:

`title`: The title displayed on the card.

`content`: The content or description inside the card.

```
function Card({ title, content }) {
```

```
  return (
    <div className="card">
      <h3>{title}</h3>
      <p>{content}</p>
    </div>
  );
}
```

InputField:

Purpose: A reusable input field component for forms.

Props:

`label`: The label displayed for the input field.

`value`: The value of the input.

`onChange`: Function to handle changes to the input.

```
function InputField({ label, value, onChange }) {
  return (
    <div>
      <label>{label}</label>
      <input type="text" value={value} onChange={onChange} />
    </div>
  );
}
```

These reusable components are modular and can be customized with different props to fit the needs of different parts of the app, making them adaptable across the

project.

State Management

Global State:
Global state is managed using **React Context** or **Redux**. State flows across the application by providing a central store for values like authentication status, user info, and app settings. These values can be accessed and updated by any component.

```
// Example with React Context
const AppContext = React.createContext();
```

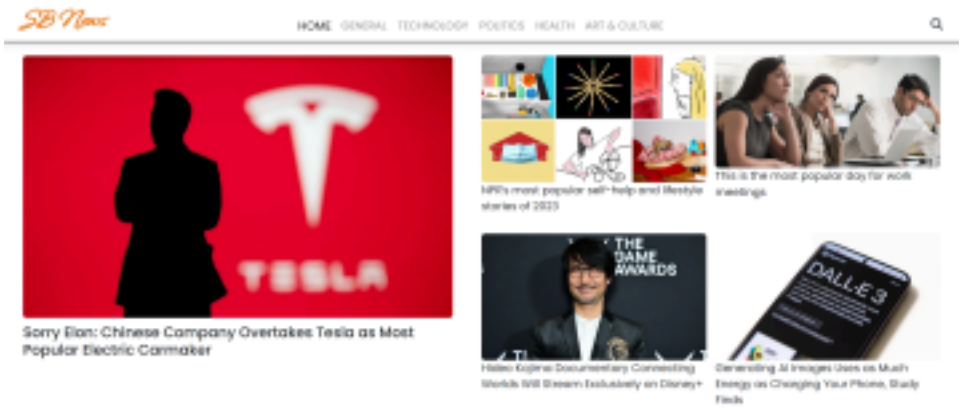
Local State:
Local state is handled within individual components using **React's** `useState` or `useReducer` hooks. This state is isolated to the component and is typically used for form data, toggles, or component-specific logic.

```
const [count, setCount] = useState(0);
```

User Interface

Screenshots or GIFs:
Currently, no visuals can be provided here, but UI features include:

Dashboard with charts, tables, and links to other sections.



Modals for user interaction, such as confirming actions or displaying extra information.

Styling

CSS Frameworks/Libraries:

Tailwind CSS: A utility-first CSS framework for rapid styling.

Styled-Components: A library for writing CSS inside JavaScript, allowing scoped and dynamic styling.

```
const Button = styled.button`
  background-color: blue;
  padding: 10px;
  color: white;
`;
```

Theming:
Custom theming is implemented using **CSS Variables** or **Styled-Components' ThemeProvider**, allowing for easy management of dark/light modes and consistent design patterns across the app.

```
 const theme = {
  colors: {
    primary: '#6200ea',
    secondary: '#03dac6',
  },
};
```

Testing Strategy

1. **Unit Testing**:

Purpose: Unit tests focus on testing individual components or functions in isolation to ensure they behave as expected.

Tools:

- **Jest**: Jest is a popular testing framework for JavaScript that can be used to run unit tests. It provides functionalities like mocking, spies, assertions, and snapshot testing.

- **React Testing Library**: For React components, React Testing Library is commonly used to render components in a simulated DOM environment and interact with them as a user would. This promotes writing tests that focus on component behavior rather than implementation details.

  Example:

  js
  Copy

```
import { render, screen } from '@testing-library/react';
import MyComponent from './MyComponent';

test('renders the component correctly', () => {
  render(<MyComponent />);
  expect(screen.getByText('Hello, World!')).toBeInTheDocument();
});
```

2. **Integration Testing**:

Purpose: Integration tests verify how multiple components or modules work together. These tests ensure that the interactions between various parts of the system (e.g., a form component and its associated API call) work as expected.

Tools:

- **Jest**: Jest is also commonly used for integration testing, especially when working with a test environment like `jest-environment-jsdom` for simulating browser behavior.
- **React Testing Library**: This can be extended to test integration points between components, ensuring the integration behaves as expected in a controlled environment.

  Example:

  js
  Copy

```
test('submitting the form calls the API', () => {
```

```
  render(<MyForm />);

  fireEvent.change(screen.getByLabelText(/name/i), {
target: { value: 'John Doe' } });

  fireEvent.click(screen.getByText(/submit/i));


  // Verify API call

  expect(mockApiCall).toHaveBeenCalledWith({ name: 'John
Doe' });

});
```

3. **End-to-End (E2E) Testing**:

Purpose: E2E tests ensure that the entire application works from the user's perspective. It tests the flow of the application, interacting with the UI, backend, and other systems.

Tools:

- **Cypress**: Cypress is a robust tool for end-to-end testing. It allows you to run tests in the browser and interact with elements as a user would, validating the application flow.
- **Puppeteer**: Puppeteer is another tool that can be used to automate browsers for end-to-end testing, although it is generally more suited for headless browser testing.

Example:

js
Copy

```
describe('User Flow', () => {

  it('should submit the form successfully', () => {

    cy.visit('/form');

    cy.get('input[name="name"]').type('John Doe');

    cy.get('button[type="submit"]').click();

    cy.contains('Thank you for your
submission').should('be.visible');

  });

});
```

Code Coverage

1. **Purpose**: Code coverage measures the percentage of your code that is executed while the tests are running. It helps ensure that your tests cover all critical paths, including edge cases.

2. **Tools**:

Jest: Jest provides built-in code coverage features by running tests with the `--coverage` flag. It generates reports that show which lines of code are covered by tests and which are not.

Istanbul/NYC: These tools can be used in conjunction with Jest or Mocha for generating detailed code coverage reports, highlighting the areas that need more attention.

Example:

bash
Copy
```
jest --coverage
```

3. **Techniques**:

Statement Coverage: Ensure every statement in your code is executed at least once by the tests.

Branch Coverage: Make sure each conditional branch (e.g., `if`, `else`) is tested.

Path Coverage: Validate that every potential path through the code is tested.

Function Coverage: Ensure that every function is called during tests.

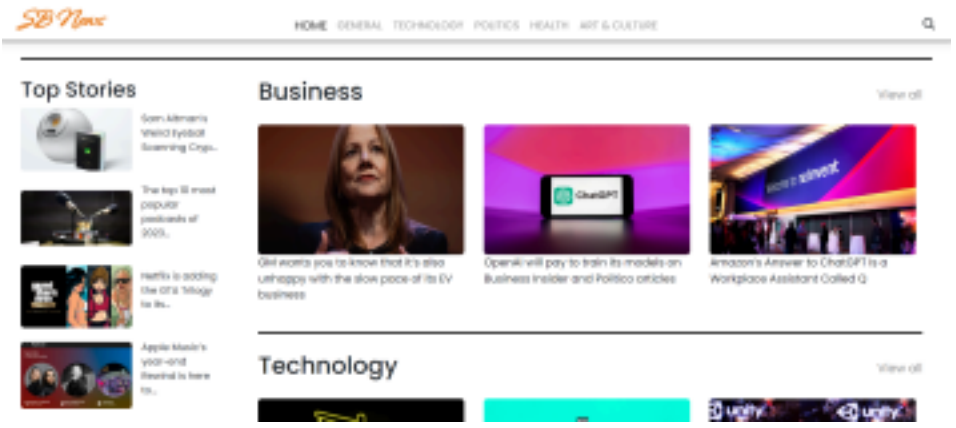
Example Coverage Report (Jest): After running Jest with coverage, you might see a report like this:

txt
Copy
```
Statements   : 95.5% (44/46)
Branches     : 90.0% (18/20)
Functions    : 100% (10/10)
Lines        : 94.7% (36/38)
```

Screenshots

Popular categories

In the hero component, the trending news articles are displayed. It is to highlight them. Apart from that, the search bar is also available to search for various articles and categories.

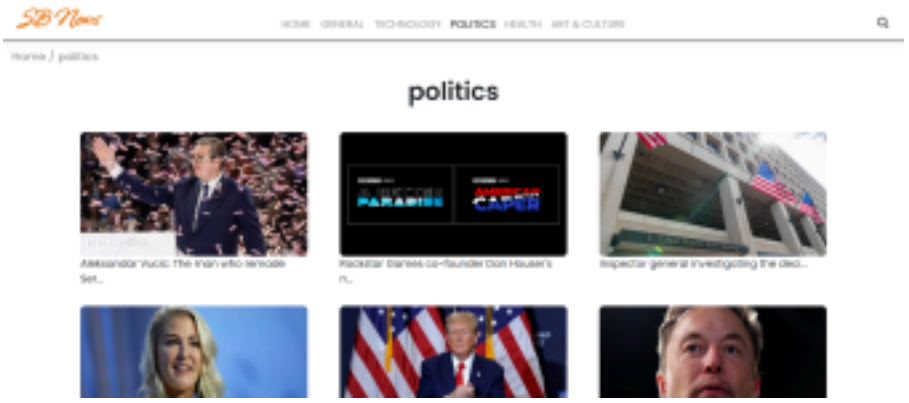


Newsletter

Staying informed is key! This section would act as a magnet for users who want to stay up-to-date on the latest news. A brief signup form with an email field would be presented, along with a clear call to action button like "Subscribe Now" or "Get Daily News Updates." With a simple click, users can join the InsightStream community and receive curated news delivered straight to their inbox.

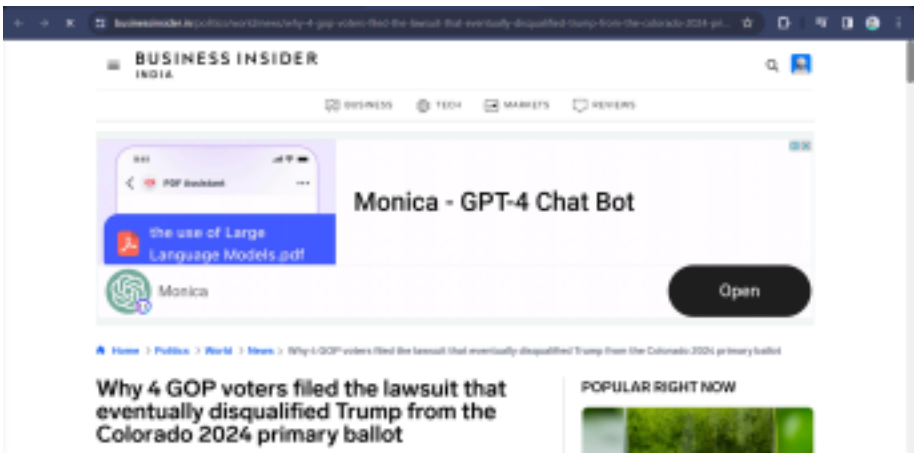
Category/Search result page

Finding the news you crave is effortless with InsightStream. This page displays a neatly organized list of articles matching your chosen category or specific search query. Each entry would provide a clear headline, a concise summary, and if available, an image to give you a quick glimpse into the story. To further refine your exploration, filters or sorting options might be available. Imagine narrowing down results by date, source, keyword, or other relevant criteria to pinpoint exactly what you're looking for.



Redirected Article page

This is where you dive deep! The article page proudly displays the complete news story, retrieved directly from the original source. To keep you engaged and exploring related topics, the page might also suggest additional articles based on the current story. These suggestions can open doors to a world of interconnected information, allowing you to become a well-rounded news connoisseur.



Project Demo link:

https://drive.google.com/file/d/1i8y09FiMk7QM0akH3my10OBWqXH-8dNh/view?usp=sharing

Known Issues

1. **Performance with Large Data Sets**:

- **Description**: Performance may degrade when rendering large lists or tables.
- **Cause**: Inefficient rendering or lack of virtualization.
- **Workaround**: Use libraries like `react-window` or `react-virtualized` for better handling of large datasets.

2. **Mobile Responsiveness**:

- **Description**: Some UI elements may not render correctly on older mobile devices or certain screen sizes.
- **Cause**: CSS media query issues or improper viewport settings.
- **Workaround**: Test across more devices and adjust media queries for legacy devices.

3. Modal Overlap:

- **Description**: Modals may overlap or stack incorrectly in rare edge cases.
- **Cause**: Incorrect handling of z-index or modal state management.
- **Workaround**: Ensure correct z-index management and reset modal states properly.

3. **Safari Rendering Bugs**:

- **Description**: Some components may not display correctly on Safari (e.g., stuttering animations, broken hover effects).
- **Cause**: Safari's inconsistent support for certain CSS properties and JavaScript APIs.
- **Workaround**: Use specific polyfills or CSS prefixes for Safari compatibility.

- **I18n Issues**:

- **Description**: Some language translations may break or not display correctly.
- **Cause**: Incomplete translations or poor handling of text direction (especially for RTL languages).
- **Workaround**: Complete translation files and improve RTL handling using libraries like

---

Future Enhancements

1. **New Components**:

- **Customizable Data Tables**: Add components with sorting, filtering, and pagination capabilities.
- **Dynamic Form Builder**: A form builder with drag-and-drop functionality and validation.

2. **Animations**:

- **Enhanced UI Transitions**: Smooth animations for page transitions and element fades to improve user experience.
- **Micro-interactions**: Add subtle animations for user interactions like button clicks, hover effects, or scrolling.

3. **Improved Styling**:

- **Dark Mode**: Implement a dark mode toggle for better accessibility and user preference.
- **Theming**: Enhance theming capabilities to allow users to customize the UI's color palette.

4. **Performance Optimizations**:

- **Code Splitting**: Implement better code splitting to improve load times.
- **Lazy Loading**: Introduce lazy loading for assets and components to reduce initial load times.

5. **Accessibility**:

- **ARIA Support**: Add or improve ARIA roles and attributes to make the app more accessible to screen readers.
- **Keyboard Navigation**: Enhance support for keyboard navigation throughout the app.