**SJC: NEWS APP**

(REACT APPLICATION)

1. **Introduction:**

SJC is a revolutionary web application designed to redefine how people discover and consume news. It offers an intuitive interface, dynamic search, and a vast range of news categories for all types of users. Join SJC to embark on an informative journey and experience the future of news consumption.

**Description:**

Welcome to the cutting-edge frontier of news exploration with SJC! 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, SJC is poised to redefine the entire news discovery and consumption process.

Designed with a commitment to user-friendly aesthetics, SJC 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.

From those seeking the latest headlines to seasoned news connoisseurs, SJC 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 SJC 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 SJC, where every headline becomes a gateway to a world of information waiting to be discovered and understood. Trust SJC to be your reliable companion on the journey of staying connected with the pulse of the world.

**Team Members:**

1) AAKASH P (TEAM LEADER)

2) SARAVANAN S (TEAM MEMBER)

3) VIGNESH R (TEAM MEMBER)

4) SRIRAM S (TEAM MEMBER)

5) GOPI (TEAM MEMBER)

**2) PROJECT OVERVIEW :**

**Purpose of SJC 📰🌍**

SJC is a web application designed to redefine how people discover and consume news. The platform offers a user-friendly interface, dynamic search functionality, and diverse news categories, catering to both casual readers and dedicated news enthusiasts.

**Project Goals 🎯**

1. **Seamless News Access –** Provide a centralized platform to access global news across multiple categories in real time**.**
2. **User-Friendly Experience –** Ensure an intuitive and engaging UI, allowing users to effortlessly navigate, search, and filter news articles.
3. **Personalized Content –** Implement advanced search features and categorization to tailor news recommendations based on user preferences.
4. **Efficient Technology Stack –** Utilize React.js, APIs, and modern web technologies to deliver a smooth and scalable experience.
5. **Community Engagement –** Encourage users to explore, share, and discuss news stories, fostering an interactive news-reading culture.

**3)ARCHITECTURE:**

* + **Component Structure of SJC ⚛️📰**

The SJC news application follows a **modular React component structure**, ensuring reusability, scalability, and efficient state management. The project is organized into **four major folders**:

📂 **Components** – Reusable UI components  
📂 **Context** – Manages global state  
📂 **Pages** – Main pages for different routes  
📂 **Styles** – CSS files for styling

**🔹 Major Components & Their Interaction**

**1️.App.js (Root Component)**

* Serves as the **entry point** for the application.
* Wraps all components inside **React Router** for navigation.
* Uses **Context API** to provide global state management.

**2️.Navbar Component *(Global Component)***

* Displays the **app logo, navigation links, and search bar**.
* Uses **React Router** to navigate between pages.
* Calls handleSearch() to filter news based on user input.

**3️.Hero Component *(Home Page Banner)***

* Highlights **trending news articles** dynamically fetched from an API.
* Includes a **search bar** for quick news discovery.

**4️.Categories Component**

* Displays a **list of news categories** (e.g., Business, Sports, Technology).
* Clicking a category fetches relevant news articles via API.

**5️.NewsList Component**

* Fetches and displays **news articles** based on selected category or search.
* Uses **Axios** to make API calls.
* Maps through the response and renders NewsCard components.
  + **State Management in SJC 🛠️⚛️**

SJC uses **Context API** for state management, ensuring efficient data flow across components without unnecessary prop drilling.

**🔹 Why Context API?**

✔ **Lightweight & Built-in** – No need for external libraries like Redux.  
✔ **Global State Management** – Easily share data across components.  
✔ **Performance Optimization** – Uses useReducer to avoid unnecessary re-renders.

**🔹 How State Management Works in SJC?**

1️.**NewsContext.js (Global Store)**

* Manages the **global state** for:
  + Fetched news articles 📡
  + Selected category 🏷️
  + Search query 🔍
* Uses useContext and useReducer to update state efficiently.

2️.**App.js (Root Component)**

* Wraps all components inside NewsProvider, ensuring they can access the shared state.

3️.**Consuming Context**

* Components like **Navbar, NewsList, and ArticlePage** consume NewsContext using useContext().

**ROUTING :**

**Routing Structure in SJC 🚀🛤️**

SJC uses **React Router (react-router-dom)** for seamless **client-side navigation** between different pages. This ensures a **single-page application (SPA) experience** where users can browse news categories, search results, and full articles without full-page reloads.

**🔹 Installation of React Router**

Before using React Router, ensure it's installed:

**npm install react-router-dom**

**🔹 Routing Structure**

**SJC follows a structured routing approach using BrowserRouter, Routes, and Route components from React Router.**

**📁 Pages Structure**

**📂 src**

**┣ 📂 components**

**┣ 📂 pages**

**┃ ┣ 📜 Home.js**

**┃ ┣ 📜 Category.js**

**┃ ┣ 📜 SearchResults.js**

**┃ ┣ 📜 Article.js**

**┣ 📜 App.js**

**┣ 📜 index.js**

**┣ 📜 NewsContext.js**

**🛠 App.js – Main Routing Configuration**

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import Home from "./pages/Home";

import Category from "./pages/Category";

import SearchResults from "./pages/SearchResults";

import Article from "./pages/Article";

import Navbar from "./components/Navbar";

import Footer from "./components/Footer";

function App() {

return (

<Router>

<Navbar /> {/\* Persistent Navbar across all pages \*/}

<Routes>

<Route path="/" element={<Home />} />

<Route path="/category/:categoryName" element={<Category />} />

<Route path="/search/:query" element={<SearchResults />} />

<Route path="/article/:id" element={<Article />} />

</Routes>

<Footer /> {/\* Persistent Footer across all pages \*/}

</Router>

);

}

export default App;

**4) SETUP INSTRUCTION:**

**Prerequisites for SJC 🛠️**

Before setting up and running the **SJC News Application**, ensure you have the following software dependencies installed:

**🔹 Required Software & Dependencies**

**1️.Node.js & npm (Required for running React)**

✔ **Node.js** is a JavaScript runtime environment needed to run the React application.  
✔ **npm (Node Package Manager)** is included with Node.js and is required to install dependencies.

📥 **Download & Install:** [Node.js Official Site](https://nodejs.org/en/download/)  
📌 **Check Installation:**

BASH:

node -v # Check Node.js version

npm -v # Check npm version

**2️.React.js (Core Frontend Framework)**

✔ React is the JavaScript library used for building the SJC application.

📌 **Installation (If not installed):**

BASH:

npx create-react-app sjc-news-app # Creates a new React project

cd sjc-news-app

npm start # Starts development server

**3️.React Router DOM (For Client-Side Navigation)**

✔ Handles navigation between pages in the app.

📌 **Installation:**

npm install react-router-dom

**4️.Axios (For Fetching News API Data)**

✔ Used to send HTTP requests to fetch news articles.

📌 **Installation:**

npm install axios

* + **Installation Guide for SJC 🛠️**

Follow these steps to **clone the repository, install dependencies, and configure environment variables** to set up SJC on your local machine. 🚀

**🔹 Step 1: Prerequisites**

Before proceeding, ensure you have the required software installed:  
✔ **Node.js & npm** – [Download Here](https://nodejs.org/en/download/)  
✔ **Git** – [Download Here](https://git-scm.com/downloads)  
✔ **Code Editor (VS Code Recommended)** – [Download Here](https://code.visualstudio.com/download)  
✔ **NewsAPI Key** – Get API Key

📌 **Check installations:**

node -v # Check Node.js version

npm -v # Check npm version

git --version # Check Git version

**🔹 Step 2: Clone the Repository**

Open your terminal or command prompt and **navigate to your desired directory**, then run:

git clone <REPOSITORY\_URL>

cd sjc-news-app

**🔹 Step 3: Install Dependencies**

Once inside the project folder, install required **npm packages**:

npm install

**🔹 Step 4: Configure Environment Variables**

SJC requires a **NewsAPI key** to fetch news articles. Follow these steps:

1️.Create a .env file in the project root directory:

touch .env

2️. Open .env and add your **NewsAPI key**:

REACT\_APP\_NEWS\_API\_KEY=your\_api\_key\_here

3️.**Restart the server** after adding the API key for changes to take effect.

**🔹 Step 5: Start the Development Server**

Run the following command to start the app:

npm start

**🔹 Step 6: Verify Installation**

✅ **Homepage loads successfully** with trending news.  
✅ **Navbar and search functionality** work as expected.  
✅ **Clicking on news categories** fetches relevant articles.

**5) Folder Structure**

Client-Side Organization of SJC ⚛️🗂️

The **SJC** React application is structured using a **modular approach**, making it scalable, maintainable, and efficient. Below is a breakdown of the **folder structure and its purpose**.

🔹 Project Folder Structure

📂 sjc-news-app

┣ 📂 public # Static assets (favicons, images, etc.)

┃ ┗ 📜 index.html # Main HTML file (React mounts here)

┣ 📂 src # Main source code directory

┃ ┣ 📂 components # Reusable UI components

┃ ┣ 📂 pages # Full-page components (mapped to routes)

┃ ┣ 📂 context # Global state management (Context API)

┃ ┣ 📂 styles # CSS files for styling

┃ ┣ 📂 assets # Images, icons, and other media

┃ ┣ 📂 utils # Helper functions (e.g., API requests)

┃ ┣ 📜 App.js # Root React component (Routes setup)

┃ ┣ 📜 index.js # Renders App component to the DOM

┃ ┣ 📜 NewsContext.js # Context API for state management

┗ 📜 package.json # Project dependencies & scripts

**🔹 Folder & File Breakdown**

**📁 1. public/ (Static Assets)**

* Contains **static files** that do not change during runtime.
* The main **index.html** file is stored here, where React mounts.

📌 **Key Files**:

📜 index.html # React root mounting file

📜 favicon.ico # App icon

📜 manifest.json # Web app metadata

**📁 2. src/ (Main Source Code)**

This is where all the React components, pages, styles, and logic reside.

**📂 3. components/ (Reusable UI Components)**

* Houses **small, reusable UI elements** used across multiple pages.
* Examples: Navbar, Footer, NewsCard, SearchBar, etc.

📌 **Key Components**:

📜 Navbar.js # Navigation bar (contains links & search)

📜 Footer.js # Footer with app info & social links

📜 NewsCard.js # Displays a single news article preview

📜 SearchBar.js # Search functionality for finding news

📜 CategoryList.js # List of news categories for filtering

Example **NewsCard.js**:

function NewsCard({ title, description, image, url }) {

return (

<div className="news-card">

<img src={image} alt="News" />

<h3>{title}</h3>

<p>{description}</p>

<a href={url} target="\_blank" rel="noopener noreferrer">Read More</a>

</div>

);

}

export default NewsCard;

**📂 4. pages/ (Full-Page Components)**

* Contains **full-page components** that match different routes.

📌 **Key Pages**:

bash

📜 Home.js # Homepage (Trending news & categories)

📜 Category.js # Displays articles based on selected category

📜 SearchResults.js # Shows articles based on user search

📜 Article.js # Displays full article details

**📂 5. context/ (Global State Management)**

* Manages **state using Context API** to share data across components.

📌 **Key File**:

bash

📜 NewsContext.js # Manages news state, categories, and search data

Example **NewsContext.js**:

import { createContext, useState, useEffect } from "react";

import axios from "axios";

const NewsContext = createContext();

export const NewsProvider = ({ children }) => {

const [news, setNews] = useState([]);

const [category, setCategory] = useState("general");

useEffect(() => {

const fetchNews = async () => {

const response = await axios.get(`https://newsapi.org/v2/top-headlines?category=${category}&apiKey=YOUR\_API\_KEY`);

setNews(response.data.articles);

};

fetchNews();

}, [category]);

return <NewsContext.Provider value={{ news, setCategory }}>{children}</NewsContext.Provider>;

};

export default NewsContext;

* + **Utilities in SJC 🛠️⚙️**

The **SJC** project uses various **helper functions, utility classes, and custom hooks** to keep the code modular, reusable, and efficient. These utilities are placed in the **utils/** and **hooks/** directories to separate logic from UI components.

**🔹 1. Helper Functions (Stored in utils/ Folder)**

**📌 api.js – Handles API Requests**

To avoid **repeating API calls** across components, we centralize them in utils/api.js.

import axios from "axios";

const API\_KEY = process.env.REACT\_APP\_NEWS\_API\_KEY;

const BASE\_URL = "https://newsapi.org/v2";

// Fetch top headlines based on category

export const fetchNewsByCategory = async (category) => {

try {

const response = await axios.get(

`${BASE\_URL}/top-headlines?category=${category}&apiKey=${API\_KEY}`

);

return response.data.articles;

} catch (error) {

console.error("Error fetching news:", error);

return [];

}

};

// Fetch news based on search query

export const fetchNewsBySearch = async (query) => {

try {

const response = await axios.get(

`${BASE\_URL}/everything?q=${query}&apiKey=${API\_KEY}`

);

return response.data.articles;

} catch (error) {

console.error("Error fetching search results:", error);

return [];

}

};

**🔹 2. Custom Hooks (Stored in hooks/ Folder)**

To simplify state management and API calls, we use **custom hooks**.

**📌 useFetchNews.js – Custom Hook for Fetching News**

Instead of handling API requests inside each component, we create a **reusable hook**.

import { useState, useEffect } from "react";

import { fetchNewsByCategory } from "../utils/api";

const useFetchNews = (category) => {

const [news, setNews] = useState([]);

const [loading, setLoading] = useState(true);

useEffect(() => {

const getNews = async () => {

setLoading(true);

const articles = await fetchNewsByCategory(category);

setNews(articles);

setLoading(false);

};

getNews();

}, [category]);

return { news, loading };

};

export default useFetchNews;

**🔹 3. Utility Classes for Styling (Stored in styles/utils.css)**

To maintain **consistent design**, we use utility CSS classes.

**📌 utils.css – Common Styling Classes**

.text-center {

text-align: center;

}

.card {

border-radius: 8px;

box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.1);

padding: 15px;

}

.btn {

background-color: #007bff;

color: white;

padding: 10px 15px;

border: none;

cursor: pointer;

border-radius: 5px;

}

.btn:hover {

background-color: #0056b3;

}

**6) Running the Application :**

**Commands to Start the Frontend Server Locally 🚀**

**To run the SJC React application on your local machine, follow these steps:**

**🔹 Step 1: Navigate to the Project Directory**

**Make sure you are in the frontend (client) directory where the React project is located.**

cd sjc-news-app

**🔹 Step 2: Install Dependencies**

If you haven't installed dependencies yet, run:

npm install

**🔹 Step 3: Start the Development Server**

Run the following command to start the frontend server:

npm start

✅ This will launch the React app in your **default browser** at:  
🔗 http://localhost:3000

**🔹 Step 4: Troubleshooting (If Needed)**

🔴 **If you encounter port issues (Port 3000 is already in use)**:

PORT=4000 npm start

This will run the server on **http://localhost:4000** instead.

🔴 **If npm start doesn't work, try running:**

npm run dev

**7) Component Documentation**

**Key Components in SJC ⚛️📰**

The **SJC** news application is built using **React components**, each serving a specific purpose to ensure a smooth and modular experience. Below is a breakdown of the **major components**, their roles, and the props they receive.

**🔹 1. Navbar.js – Navigation Bar**

**📌 Purpose:**  
✔ Provides navigation links to different pages (Home, Categories, Search).  
✔ Includes a search bar to filter news.

**📥 Props:**

* **None (Uses internal state for search input)**

**🔹 Code Snippet:**

import { Link, useNavigate } from "react-router-dom";

import { useState } from "react";

function Navbar() {

const [searchQuery, setSearchQuery] = useState("");

const navigate = useNavigate();

const handleSearch = (e) => {

e.preventDefault();

navigate(`/search/${searchQuery}`);

};

return (

<nav>

<Link to="/">Home</Link>

<Link to="/category/technology">Technology</Link>

<Link to="/category/sports">Sports</Link>

<form onSubmit={handleSearch}>

<input type="text" placeholder="Search news..." onChange={(e) => setSearchQuery(e.target.value)} />

<button type="submit">Search</button>

</form>

</nav>

);

}

export default Navbar;

**🔹 2. NewsCard.js – News Article Preview Card**

**📌 Purpose:**  
✔ Displays a single news article preview.  
✔ Includes an image, title, and summary.  
✔ Clicking it redirects to the full article.

🔹 Code Snippet:

function NewsCard({ title, description, image, url }) {

return (

<div className="news-card">

<img src={image} alt="News" />

<h3>{title}</h3>

<p>{description}</p>

<a href={url} target="\_blank" rel="noopener noreferrer">Read More</a>

</div>

);

}

export default NewsCard;

**🔹 3. NewsList.js – List of News Articles**

**📌 Purpose:**  
✔ Fetches and displays a list of **NewsCard** components.  
✔ Used in **Home.js, Category.js, and SearchResults.js** pages.

import NewsCard from "./NewsCard";

function NewsList({ articles }) {

return (

<div className="news-list">

{articles.map((article, index) => (

<NewsCard key={index} {...article} />

))}

</div>

);

}

export default NewsList;

**🔹 4. Category.js – Displays News by Category**

**📌 Purpose:**  
✔ Fetches and displays news based on the selected **category**.  
✔ Uses useParams() to get the category from the URL.

**📥 Props:**

* **None (Gets category from URL using useParams)**

**🔹 Code Snippet:**

import { useParams } from "react-router-dom";

import { useEffect, useState } from "react";

import { fetchNewsByCategory } from "../utils/api";

import NewsList from "../components/NewsList";

function Category() {

const { categoryName } = useParams();

const [articles, setArticles] = useState([]);

useEffect(() => {

const getNews = async () => {

const data = await fetchNewsByCategory(categoryName);

setArticles(data);

};

getNews();

}, [categoryName]);

return (

<div>

<h2>{categoryName} News</h2>

<NewsList articles={articles} />

</div>

);

}

export default Category;

**🔹 5. SearchResults.js – Displays News Based on Search**

**📌 Purpose:**  
✔ Fetches and displays **news articles matching a search query**.  
✔ Uses useParams() to get the search query from the URL.

**📥 Props:**

* **None (Gets search query from URL using useParams)**

**🔹 Code Snippet:**

import { useParams } from "react-router-dom";

import { useEffect, useState } from "react";

import { fetchNewsBySearch } from "../utils/api";

import NewsList from "../components/NewsList";

function SearchResults() {

const { query } = useParams();

const [articles, setArticles] = useState([]);

useEffect(() => {

const getNews = async () => {

const data = await fetchNewsBySearch(query);

setArticles(data);

};

getNews();

}, [query]);

return (

<div>

<h2>Search Results for: "{query}"</h2>

<NewsList articles={articles} />

</div>

);

}

export default SearchResults;

**🔹 6. Article.js – Displays Full News Article**

**📌 Purpose:**  
✔ Fetches and displays **full details of a news article**.  
✔ Uses useParams() to get the article **ID** from the URL.

**📥 Props:**

* **None (Gets article ID from URL using useParams)**

**🔹 Code Snippet:**

import { useParams } from "react-router-dom";

import { useEffect, useState } from "react";

import { fetchArticleById } from "../utils/api";

function Article() {

const { id } = useParams();

const [article, setArticle] = useState(null);

useEffect(() => {

const getArticle = async () => {

const data = await fetchArticleById(id);

setArticle(data);

};

getArticle();

}, [id]);

return (

<div>

{article ? (

<>

<h2>{article.title}</h2>

<img src={article.image} alt="Article" />

<p>{article.content}</p>

<a href={article.url} target="\_blank" rel="noopener noreferrer">Read Full Article</a>

</>

) : (

<p>Loading...</p>

)}

</div>

);

}

export default Article;

**🔹 7. Footer.js – Footer Component**

**📌 Purpose:**  
✔ Displays **social media links and copyright info**.

**📥 Props:**

* **None**

**🔹 Code Snippet:**

function Footer() {

return (

<footer>

<p>© 2025 SJC News. All rights reserved.</p>

<div>

<a href="#">Facebook</a> | <a href="#">Twitter</a> | <a href="#">LinkedIn</a>

</div>

</footer>

);

}

export default Footer;

**Reusable Components in SJC ⚛️🛠️**

To ensure modularity and maintainability, **SJC** follows a **component-based architecture** with several **reusable components**. These components are designed to be flexible and configurable using **props**, making them adaptable across multiple pages.

**🔹 1. NewsCard.js – Reusable News Preview Component**

**📌 Purpose:**  
✔ Displays a **single news article preview** (image, title, and short description).  
✔ Used in **Home, Category, and SearchResults** pages.  
✔ Clicking redirects users to the **full article**.

**📥 Props & Configurations:**

🔹 Code Snippet:

function NewsCard({ title, description, image, url }) {

return (

<div className="news-card">

<img src={image} alt="News" />

<h3>{title}</h3>

<p>{description}</p>

<a href={url} target="\_blank" rel="noopener noreferrer">Read More</a>

</div>

);

}

export default NewsCard;

✅ Usage Example in NewsList.js:

import NewsCard from "./NewsCard";

function NewsList({ articles }) {

return (

<div className="news-list">

{articles.map((article, index) => (

<NewsCard key={index} {...article} />

))}

</div>

);

}

export default NewsList;

**🔹 2. Button.js – Reusable Button Component**

**📌 Purpose:**  
✔ A **customizable button** with **different styles and actions**.  
✔ Accepts **text, click event, and styling as props**.  
✔ Used for **"Read More," "Subscribe," "Load More"** buttons.

**📥 Props & Configurations:**

🔹 Code Snippet:

function Button({ text, onClick, className = "" }) {

return (

<button className={`btn ${className}`} onClick={onClick}>

{text}

</button>

);

}

export default Button;

✅ Usage Example in Article.js:

<Button text="Read Full Article" onClick={() => window.open(article.url, "\_blank")} className="primary" />

**🔹 3. SearchBar.js – Reusable Search Component**

**📌 Purpose:**  
✔ Provides a **search input field** for finding news articles.  
✔ Used in **Navbar and Home pages**.  
✔ Redirects users to the **Search Results page**.

**📥 Props & Configurations:**

🔹 Code Snippet:

import { useState } from "react";

function SearchBar({ onSearch }) {

const [query, setQuery] = useState("");

const handleSubmit = (e) => {

e.preventDefault();

if (query) onSearch(query);

};

return (

<form onSubmit={handleSubmit}>

<input

type="text"

placeholder="Search news..."

value={query}

onChange={(e) => setQuery(e.target.value)}

/>

<button type="submit">Search</button>

</form>

);

}

export default SearchBar;

**🔹 4. CategoryList.js – Reusable Category Selector**

**📌 Purpose:**  
✔ Displays a **list of news categories** for filtering articles.  
✔ Used in **Home and Navbar** to navigate between news categories.

**📥 Props & Configurations:**

🔹 Code Snippet:

function CategoryList({ categories, onSelectCategory }) {

return (

<div className="category-list">

{categories.map((category) => (

<button key={category} onClick={() => onSelectCategory(category)}>

{category}

</button>

))}

</div>

);

}

export default CategoryList;

**🔹 5. LoadingSpinner.js – Reusable Loading Indicator**

**📌 Purpose:**  
✔ Shows a **loading animation** while fetching data.  
✔ Used in **Category, SearchResults, and Article pages**.

**📥 Props & Configurations:**

* **None (Simply displays a loading animation)**

**🔹 Code Snippet:**

function LoadingSpinner() {

return <div className="spinner">Loading...</div>;

}

export default LoadingSpinner;

**8) State Management**

**Global State Management in SJC 🌐⚛️**

SJC uses **Context API** for global state management, allowing data to be shared across components without **prop drilling**. This approach ensures that components can **access and update state efficiently**, improving performance and maintainability.

**🔹 Why Use Context API?**

✅ **Lightweight & Built-in** – No need for external libraries like Redux.  
✅ **Global Accessibility** – Easily share data across multiple components.  
✅ **Performance Optimization** – Updates only the necessary parts of the UI.

**🔹 How State Flows Across SJC?**

📌 **Context API is used to manage:**

1. **News Articles** – Fetching and storing news data.
2. **Selected Category** – Tracking the currently viewed category.
3. **Search Query** – Managing user searches dynamically.

📌 **Key Global State Files:**

* **NewsContext.js** – Defines the global state.
* **NewsProvider** – Wraps the entire app to provide state access.
* **Components (Navbar, Category, SearchResults)** consume the context.

**🔹 1. Setting Up the Global State (NewsContext.js)**

* This file defines the **NewsContext** and manages the global state for news articles, categories, and search results.

Node.js

import { createContext, useState, useEffect } from "react";

import axios from "axios";

const NewsContext = createContext();

export const NewsProvider = ({ children }) => {

const [news, setNews] = useState([]);

const [category, setCategory] = useState("general");

const [searchQuery, setSearchQuery] = useState("");

// Fetch news based on category

useEffect(() => {

const fetchNews = async () => {

try {

const response = await axios.get(

`https://newsapi.org/v2/top-headlines?category=${category}&apiKey=${process.env.REACT\_APP\_NEWS\_API\_KEY}`

);

setNews(response.data.articles);

} catch (error) {

console.error("Error fetching news:", error);

}

};

fetchNews();

}, [category]); // Refetch when category changes

return (

<NewsContext.Provider value={{ news, setCategory, setSearchQuery }}>

{children}

</NewsContext.Provider>

);

};

export default NewsContext;

import { createContext, useState, useEffect } from "react";

import axios from "axios";

const NewsContext = createContext();

export const NewsProvider = ({ children }) => {

const [news, setNews] = useState([]);

const [category, setCategory] = useState("general");

const [searchQuery, setSearchQuery] = useState("");

// Fetch news based on category

useEffect(() => {

const fetchNews = async () => {

try {

const response = await axios.get(

`https://newsapi.org/v2/top-headlines?category=${category}&apiKey=${process.env.REACT\_APP\_NEWS\_API\_KEY}`

);

setNews(response.data.articles);

} catch (error) {

console.error("Error fetching news:", error);

}

};

fetchNews();

}, [category]); // Refetch when category changes

return (

<NewsContext.Provider value={{ news, setCategory, setSearchQuery }}>

{children}

</NewsContext.Provider>

);};

export default NewsContext;

**🔹 2. Providing State to the Entire App (App.js)**

The NewsProvider wraps the **entire application**, allowing all components to access the global state.

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import { NewsProvider } from "./context/NewsContext";

import Home from "./pages/Home";

import Category from "./pages/Category";

import SearchResults from "./pages/SearchResults";

import Article from "./pages/Article";

import Navbar from "./components/Navbar";

import Footer from "./components/Footer";

function App() {

return (

<NewsProvider>

<Router>

<Navbar />

<Routes>

<Route path="/" element={<Home />} />

<Route path="/category/:categoryName" element={<Category />} />

<Route path="/search/:query" element={<SearchResults />} />

<Route path="/article/:id" element={<Article />} />

</Routes>

<Footer />

</Router>

</NewsProvider>

);

}

export default App;

**🔹 3. Consuming the Global State**

Components can access and update the global state using useContext().

**🔹 Using Global State in Category.js**

import { useParams } from "react-router-dom";

import { useContext } from "react";

import NewsContext from "../context/NewsContext";

import NewsList from "../components/NewsList";

function Category() {

const { categoryName } = useParams();

const { news, setCategory } = useContext(NewsContext);

useEffect(() => {

setCategory(categoryName);

}, [categoryName]);

return (

<div>

<h2>{categoryName} News</h2>

<NewsList articles={news} />

</div>

);

}

export default Category;

🔹 Using Global State in Navbar.js

import { useContext, useState } from "react";

import { useNavigate } from "react-router-dom";

import NewsContext from "../context/NewsContext";

function Navbar() {

const { setSearchQuery } = useContext(NewsContext);

const [query, setQuery] = useState("");

const navigate = useNavigate();

const handleSearch = (e) => {

e.preventDefault();

setSearchQuery(query);

navigate(`/search/${query}`);

};

return (

<nav>

<form onSubmit={handleSearch}>

<input type="text" placeholder="Search..." onChange={(e) => setQuery(e.target.value)} />

<button type="submit">Search</button>

</form>

</nav>

);

}

export default Navbar;

**🔹 4. Using Global State in SearchResults.js**

import { useParams } from "react-router-dom";

import { useEffect, useState, useContext } from "react";

import NewsContext from "../context/NewsContext";

import NewsList from "../components/NewsList";

import axios from "axios";

function SearchResults() {

const { query } = useParams();

const { searchQuery } = useContext(NewsContext);

const [articles, setArticles] = useState([]);

useEffect(() => {

const fetchSearchResults = async () => {

const response = await axios.get(

`https://newsapi.org/v2/everything?q=${searchQuery}&apiKey=${process.env.REACT\_APP\_NEWS\_API\_KEY}`

);

setArticles(response.data.articles);

};

fetchSearchResults();

}, [searchQuery]);

return (

<div>

<h2>Search Results for "{searchQuery}"</h2>

<NewsList articles={articles} />

</div>

);

}

export default SearchResults;

**Local State Management in SJC 🎛️**

Local state is used for **temporary, component-specific data**, such as **form inputs, loading states, and UI interactions**.

🔹 1. Local State in Search (Navbar.js)

const [searchQuery, setSearchQuery] = useState("");

const navigate = useNavigate();

const handleSearch = (e) => { e.preventDefault(); if (searchQuery.trim()) navigate(`/search/${searchQuery}`); };

🔹 2. Loading State (Category.js)

const [loading, setLoading] = useState(true);

useEffect(() => { fetchNews().then(() => setLoading(false)); }, []);

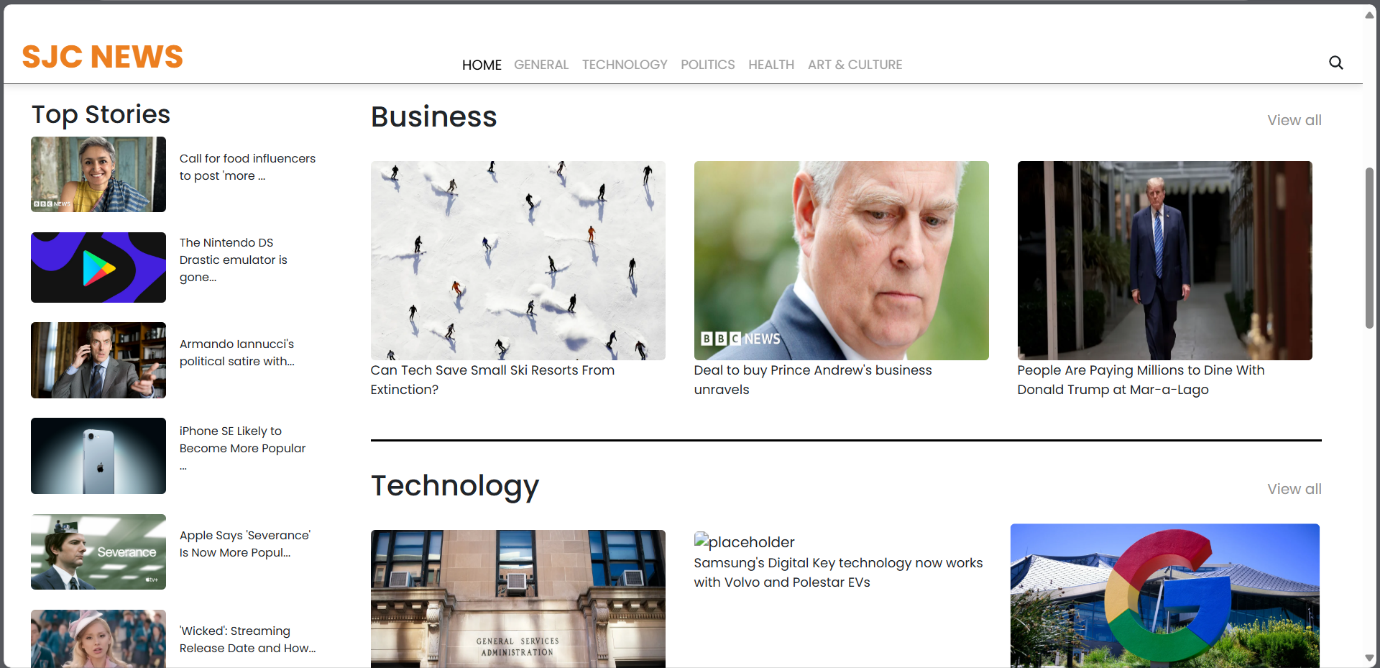
🔹 3. Modal Toggle (Article.js)

const [showModal, setShowModal] = useState(false);

<button onClick={() => setShowModal(true)}>Open</button>

**9) User Interface**

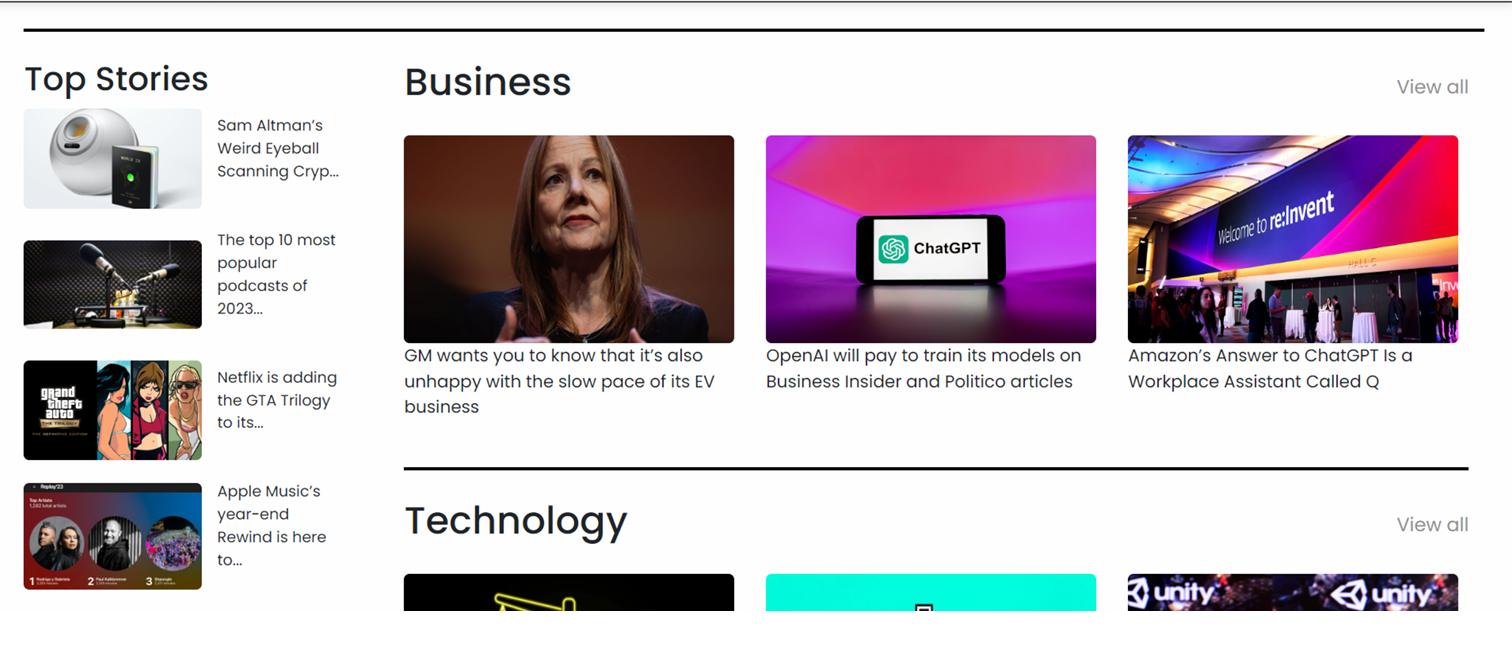
Here are some of the screenshots of the application. Hero components In the hero component, the trending news articles are displayed. It is to highlight them. Apart from that, the search bar is also available to search for various articles and categories.

****

* **Popular categories**

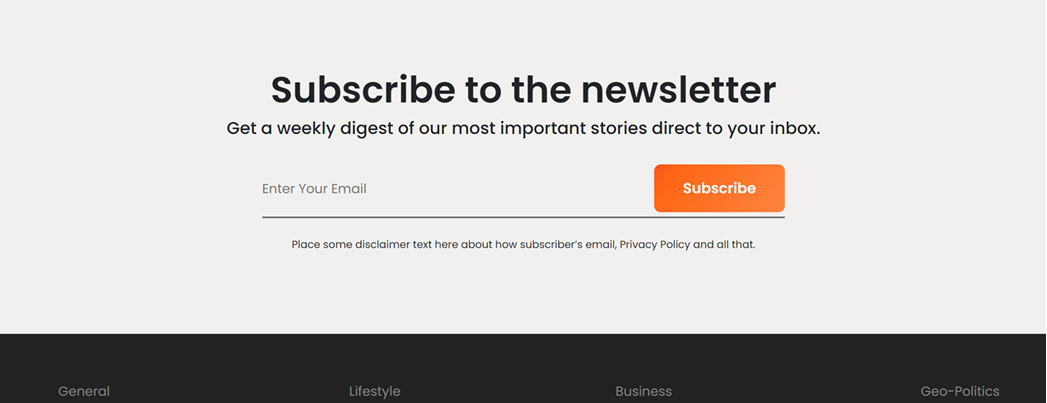
In the hero component, the trending news articles are displayed. It is to highlight them. Apart from that, the search bar is also available to search

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.

10) **Styling**

**CSS Frameworks & Libraries in SJC 🎨🖌️**

SJC uses **modern styling solutions** to ensure a **responsive, visually appealing, and maintainable UI**.

**🔹 1. Tailwind CSS (Primary Framework)**

📌 **Why?**  
✔ Utility-first approach → Faster styling.  
✔ Built-in responsiveness → No extra media queries needed.  
✔ Lightweight → No unused CSS bloat.

📌 **Installation:**

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p

**🔹 2. Bootstrap (Alternative Option)**

📌 **Why?**  
✔ Pre-built components → Faster development.  
✔ Grid system → Easy layout control.

📌 **Installation:**

npm install react-bootstrap bootstrap

**🔹 3. Custom CSS (Global & Component-Level)**

📌 **Why?**  
✔ Fine-tuned styling for custom needs.

📌 **Usage:**  
**📂 styles/global.css**

body { font-family: Arial, sans-serif; background: #f4f4f4; }

Component-Level CSS (e.g., NewsCard.css)

.news-card { border-radius: 8px; box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.1); }

**🔹 4. Styled Components (For Dynamic Styling, If Needed)**

📌 **Why?**  
✔ CSS-in-JS → Scoped styles within components.

📌 **Installation:**

npm install styled-components

📌 **Usage Example:**

import styled from "styled-components";

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

**Theming in SJC 🎨🌗**

SJC supports **theming** to enhance **user experience, branding, and accessibility**. It uses **CSS variables, Tailwind CSS, and optional dark mode** to achieve a **customizable design system**.

**🔹 1. CSS Variables for Custom Themes**

📌 **Why?**  
✔ Centralized theme colors → Easy modifications.  
✔ Supports **light/dark mode** and custom branding.

📌 **Example (global.css):**

:root {

--primary-color: #007bff;

--secondary-color: #6c757d;

--background-color: #f8f9fa;

}

body {

background-color: var(--background-color);

color: var(--primary-color);

}

✅ **Modify one value → Update the entire UI!**

**🔹 2. Tailwind CSS for Dynamic Theming**

📌 **Why?**  
✔ **Utility classes** make styling efficient.  
✔ **Dark mode support** with dark: prefix.

📌 **Example (tailwind.config.js):**

module.exports = {

darkMode: "class",

theme: {

extend: {

colors: { primary: "#007bff", secondary: "#6c757d" },

},

},

};

📌 **Usage in Components:**

<button className="bg-primary text-white px-4 py-2">Click Me</button>

**🔹 3. Dark Mode Toggle (Optional Feature)**

📌 **Why?**  
✔ Improves readability in low-light conditions.  
✔ Uses **local storage** to remember user preference.

📌 **Implementation (ThemeToggle.js):**

import { useEffect, useState } from "react";

function ThemeToggle() {

const [darkMode, setDarkMode] = useState(localStorage.getItem("theme") === "dark");

useEffect(() => {

document.documentElement.classList.toggle("dark", darkMode);

localStorage.setItem("theme", darkMode ? "dark" : "light");

}, [darkMode]);

return <button onClick={() => setDarkMode(!darkMode)}>Toggle Theme</button>;

}

export default ThemeToggle;

📌 **Usage in Navbar.js:**

<ThemeToggle />

**11) Testing**

**Testing Strategy for SJC**

SJC follows a **comprehensive testing approach** to ensure **stability, reliability, and maintainability**. The strategy includes **unit tests, integration tests, and end-to-end (E2E) tests** using **Jest, React Testing Library, and Cypress**.

**🔹 1. Unit Testing (Jest & React Testing Library) 🧩**

📌 Purpose:  
✔ Tests individual components in isolation (e.g., NewsCard.js).  
✔ Ensures props are rendered correctly and UI logic works.

📌 Setup (if not installed):

npm install --save-dev jest @testing-library/react @testing-library/jest-dom

📌 **Example: NewsCard.test.js (Unit Test)**

import { render, screen } from "@testing-library/react";

import NewsCard from "../components/NewsCard";

test("renders NewsCard with title and description", () => {

render(<NewsCard title="Test News" description="This is a test." />);

expect(screen.getByText("Test News")).toBeInTheDocument();

expect(screen.getByText("This is a test.")).toBeInTheDocument();

});

✅ **Checks if the component renders correctly.**

**🔹 2. Integration Testing (React Testing Library) 🔗**

📌 **Purpose:**  
✔ Tests how **multiple components interact** (e.g., NewsList.js displaying NewsCard.js).

📌 **Example: NewsList.test.js (Integration Test)**

import { render, screen } from "@testing-library/react";

import NewsList from "../components/NewsList";

const mockArticles = [

{ title: "News 1", description: "Desc 1", url: "#" },

{ title: "News 2", description: "Desc 2", url: "#" },

];

test("renders multiple NewsCards", () => {

render(<NewsList articles={mockArticles} />);

expect(screen.getByText("News 1")).toBeInTheDocument();

expect(screen.getByText("News 2")).toBeInTheDocument();

});

✅ **Ensures the news list correctly renders multiple cards.**

**🔹 3. End-to-End (E2E) Testing (Cypress) 🌎**

📌 **Purpose:**  
✔ Simulates **real user interactions** (e.g., navigating, searching, clicking).  
✔ Ensures **full workflow functionality** (e.g., search → article page).

📌 **Installation:**

npm install --save-dev cypress

npx cypress open

📌 **Example: search.cy.js (E2E Test for Search Functionality)**

describe("Search Functionality", () => {

it("Searches and displays results", () => {

cy.visit("http://localhost:3000");

cy.get("input[type='text']").type("Technology");

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

cy.contains("Search Results for: Technology").should("exist");

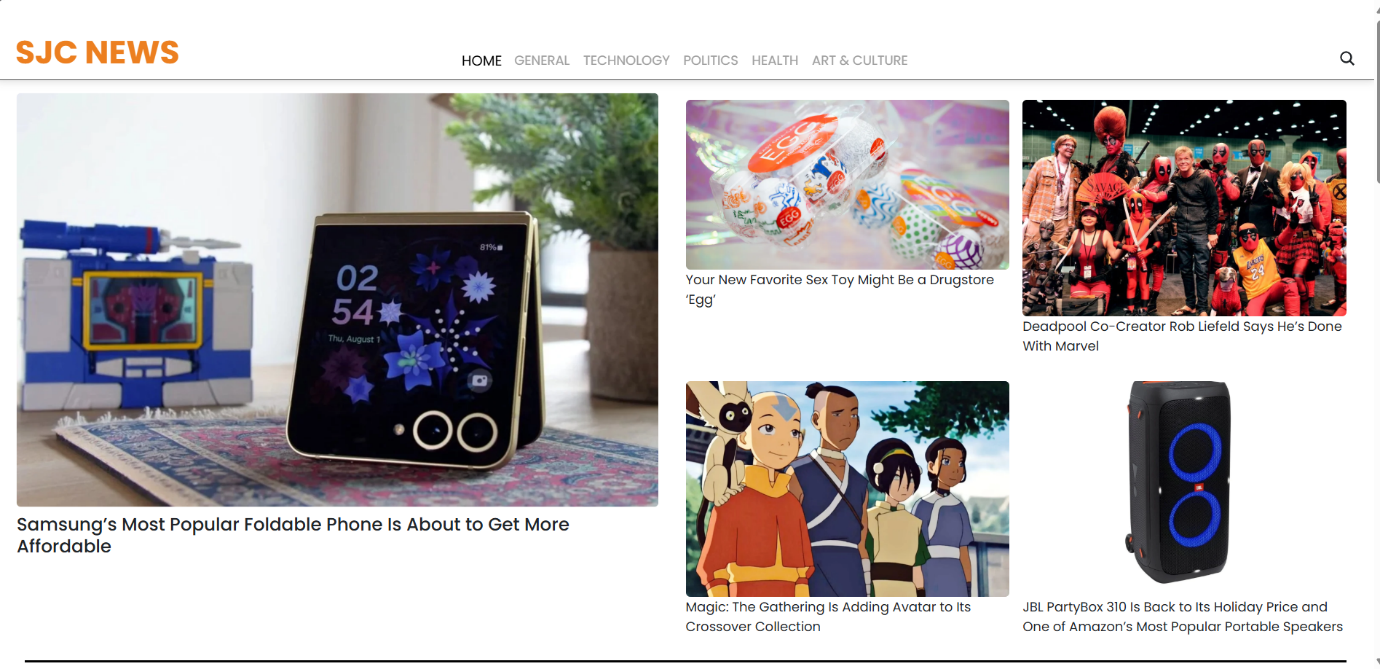
});

});

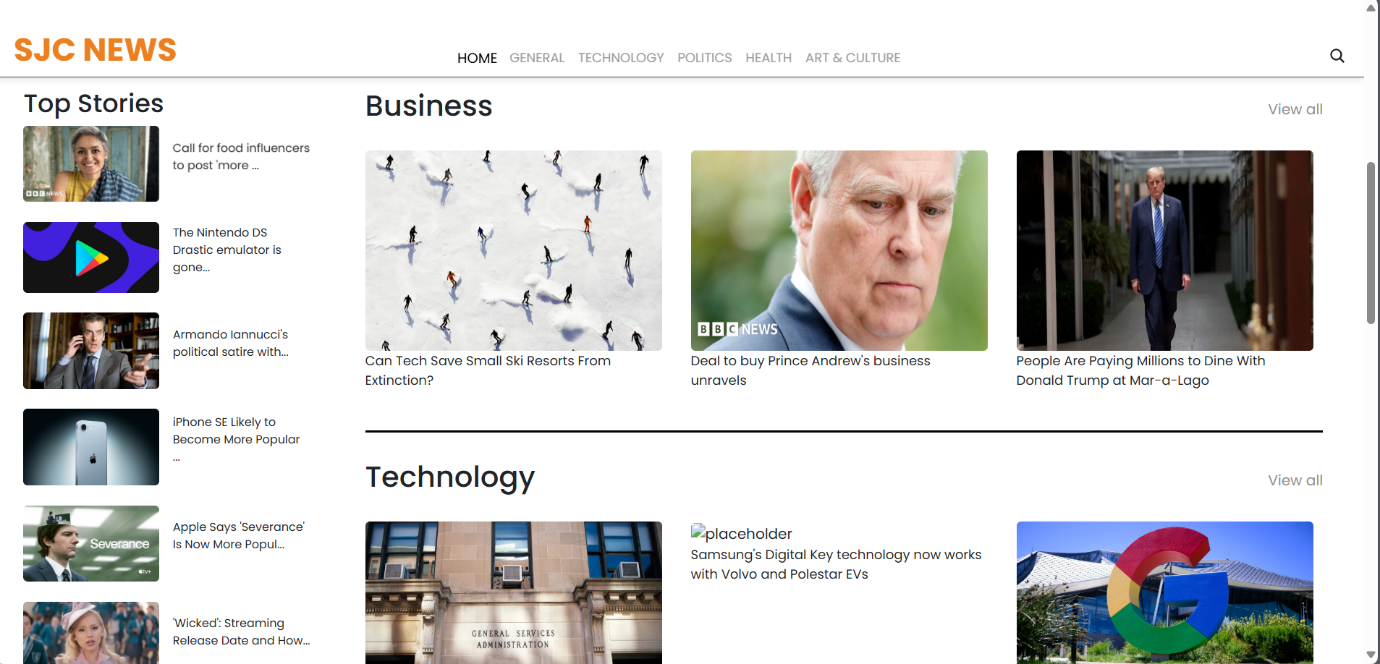
✅ **Simulates typing a search query and verifying results.**

**12) Screenshot**

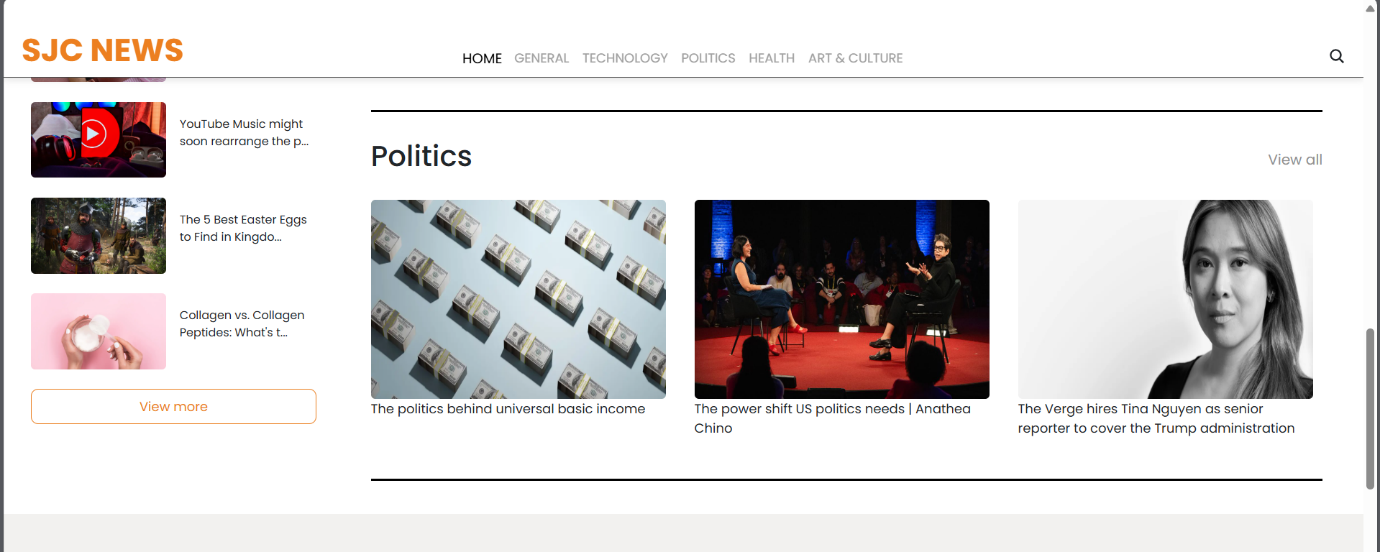
**Home page:**



**Categories :**

****

**Short Stories in Menu Bar :**

****

**13) Future Enhancements**

**🌟 Future Features & Enhancements**

**1️.New Functionalities**

* **🔍 Advanced Search** – Implement filters by category, date, or keyword.
* **🌐 Multi-language Support** – Add language selection for a global audience.
* **💾 Offline Mode** – Allow users to read saved articles without an internet connection.
* **📌 Bookmark & Read Later** – Enable users to save favorite articles.

**2️.UI/UX Improvements**

* **🎨 Enhanced Styling** – Use better typography, colors, and layouts for a modern look.
* **🌙 Dark Mode** – Provide a toggle between light and dark themes.
* **📱 Responsive Design** – Ensure a seamless experience on all screen sizes.
* **📢 Live News Ticker** – Show real-time breaking news updates.

**3️.Performance Upgrades**

* **🚀 Lazy Loading** – Load images and articles only when needed to improve speed.
* **⚡ Optimized API Calls** – Reduce API requests for better performance.
* **🗄️ Caching Mechanism** – Store previously fetched news for faster access.

**4️.Animations & Interactivity**

* **🔥 Smooth Page Transitions** – Add animated transitions between different pages.
* **🎥 Animated Loading Screens** – Improve user experience with skeleton loaders.
* **💬 Interactive Comments Section** – Let users engage with discussions.