INTRODUCTION

Project Title: [InsightStream:News App]

Team ID : SWTID1741163688150247

Team Members: VAISHNAVI D[backend]

YASMIN V R[backend]

GOPIKAA SHREE Y[frontend]

RAKSHA VAISHNAV A[frontend]

Team Membersemail: reshuvaishu1@gmail.com

yasminvankayalapati@gmail.com

gopikaashree04@gmail.com

rakshavaishnav05@gmail.com

2.PROJECT OVERVIEW

Purpose

The News Website project is designed to provide users with an interactive platform to explore and read news articles. The goal is to create a simple, fast, and user-friendly website using React.js for the frontend and Node.js for the backend, without using a database. Instead of fetching news from a database, the backend serves static JSON data, making the system lightweight and easy to manage.

Goals of the Project

1. Deliver a clean and interactive UI – Ensure the website is easy to navigate and visually appealing.

2. Fetch news dynamically – The frontend retrieves news data from the backend via API requests.

3. Category-based browsing – Users can filter news articles by different categories (e.g., Technology, Sports).

4. Search functionality– Users can find specific articles using keywords.

5. Responsive design – Ensure compatibility with various screen sizes (mobile, tablet, and desktop).

6. Efficient performance– Using JSON files instead of a database ensures fast data retrieval and simple setup.

Features of the Project

Frontend Features (React.js)

- Homepage with Latest News – Displays a list of recent news articles with images and headlines.

- Category-Based Navigation– Allows users to filter news by categories such as Technology, Sports, and Business.

- Individual News Page– Clicking on an article opens a detailed news page with full content.

- React Router for Navigation – Provides seamless navigation between pages.

- Responsive UI– Optimized for desktop and mobile devices using CSS.

Backend Features (Node.js)

-REST API with Static JSON Data – The backend serves news articles stored in a JSON file.

API Endpoints:

- GET /news – Fetch all news articles.
- GET /news/:id – Fetch a specific news article by ID.
- GET /category/:categoryName – Fetch articles by category.
- Lightweight and Fast– Since data is stored in a JSON file, no database is needed.

Use Case Scenario

1. A user visits the homepage and sees a list of the latest news articles.
2. The user clicks on a *category (e.g., Sports)* to filter articles related to sports.
3. The user clicks on an article to read the full news content.
4. The user can return to the homepage or browse other categories for more news.

3.Architecture

The News Website follows a *client-server architecture, where the frontend (React.js)interacts with a Node.js backend that serves news data from a static JSON file. This ensures a lightweight, fast, and easy-to-maintain system without requiring a database.

1. **Component Structure** (React.js Frontend)

The frontend is structured using React components making it modular and reusable.

Major Components and their Interactions

Component Description

App.js The root component that sets up routing and global state.

Header.js Displays navigation links for different news categories.

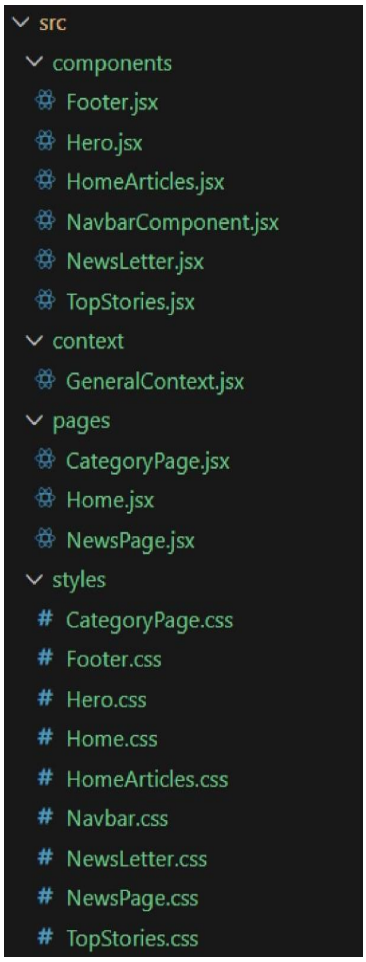
NewsList.js Fetches and displays a list of news articles from the backend.

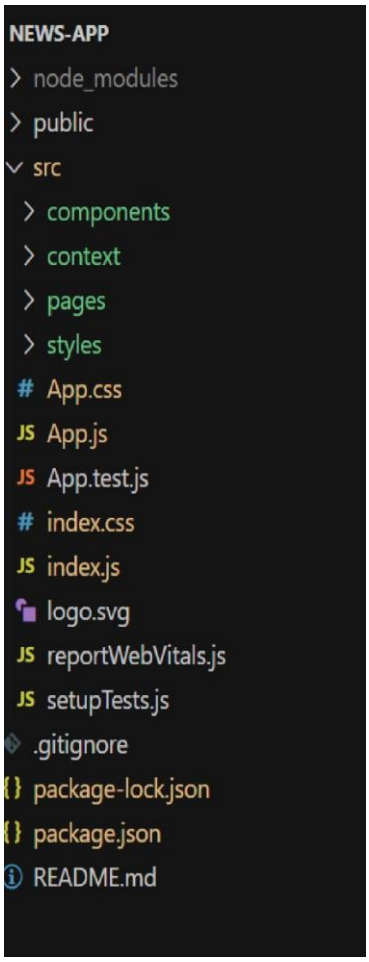
NewsCard.js Represents an individual news article with a title, image, and description.

SearchBar.js Allows users to search for news articles using keywords.

NewsDetails.js Displays the full content of an article when clicked.

Footer.js Contains website footer information.





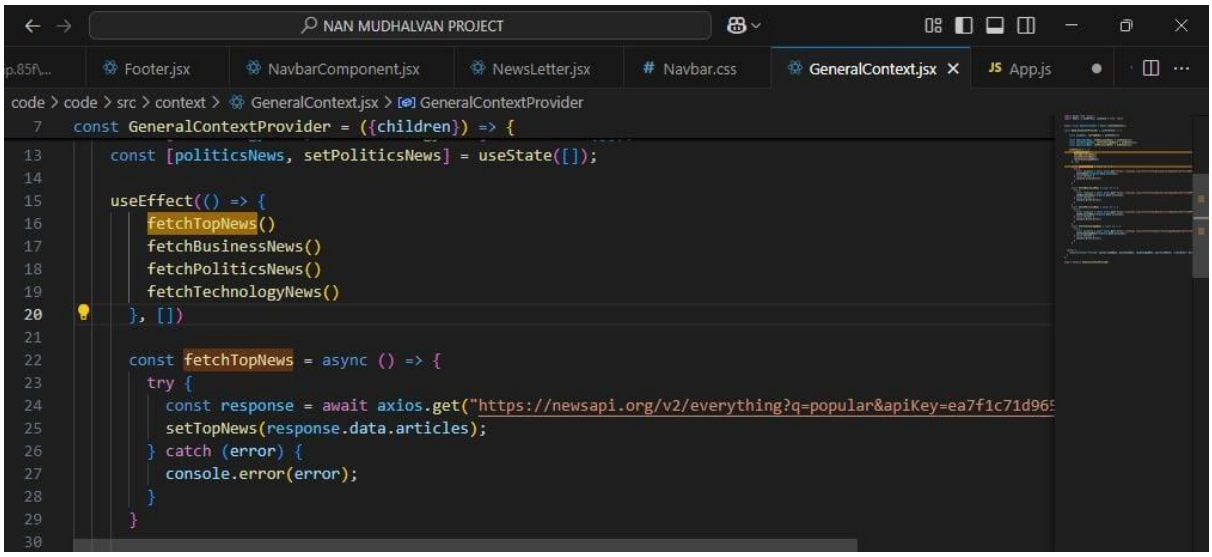
State Management

State is managed using React's built-in state hooks (useState and useEffect) and the Context API for global state management.

Local State (useState, useEffect)

useState is used to store fetched news articles, user search input, and loading states.

useEffect is used to fetch data when a component mounts or updates.



Routing (React Router)

React Router is used for navigation between different pages, allowing users to browse news categories, view full articles, and search for news.

Routing Flow*

Route

Homepage with latest news

/category/:categoryName News articles filtered by category

/news/:id Full article detail

Summary of the Architecture

1. Frontend (React.js)

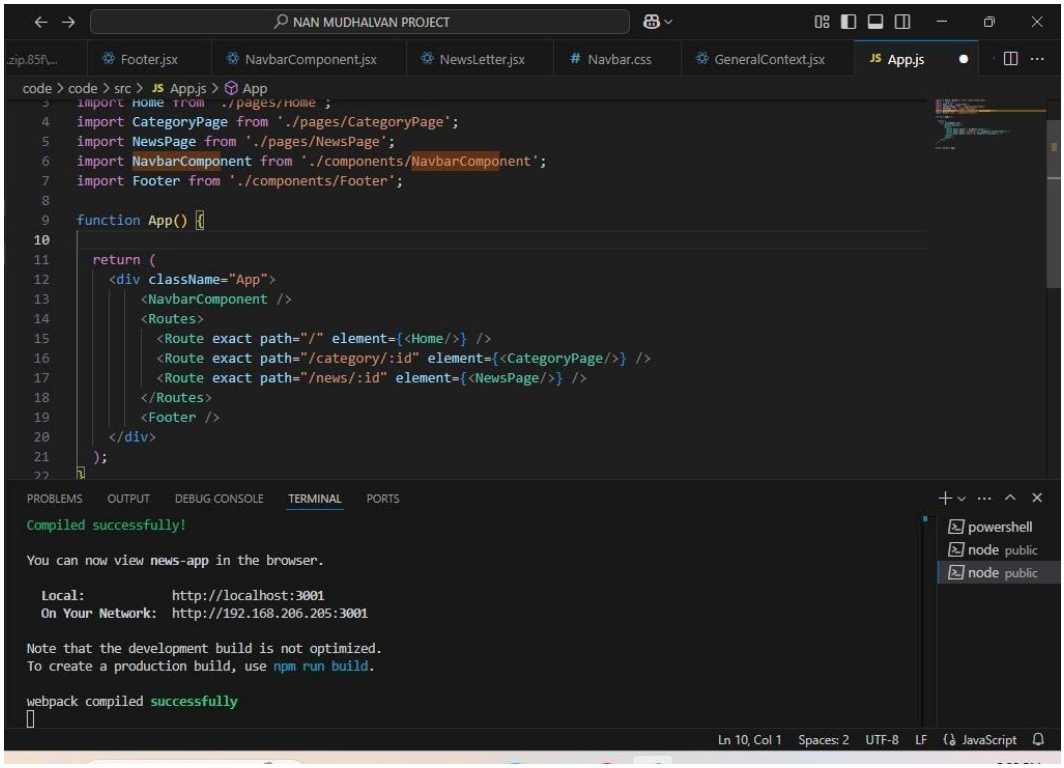
- Component-based structure
- State management using useState, useEffect, and Context API
-Routing handled by React Router

2. *Backend (Node.js)

- Serves news data from a static JSON file
- Provides REST API for fetching news

3. Data Flow

- The frontend requests data → Express.js reads JSON file → Sends response as JSON → React updates UI



4.PRE-REQUISITES:

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:

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:

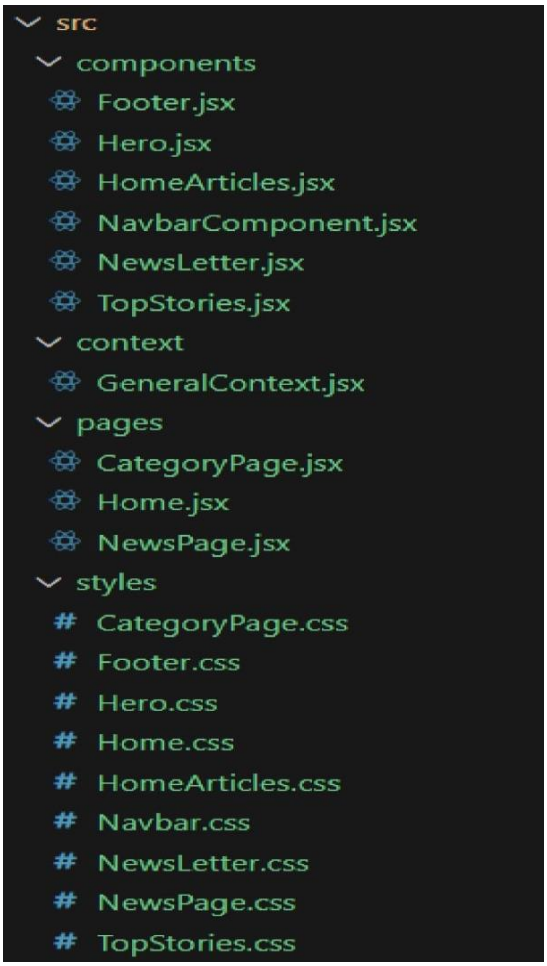
• Open your web browser and navigate to http://localhost:3000.

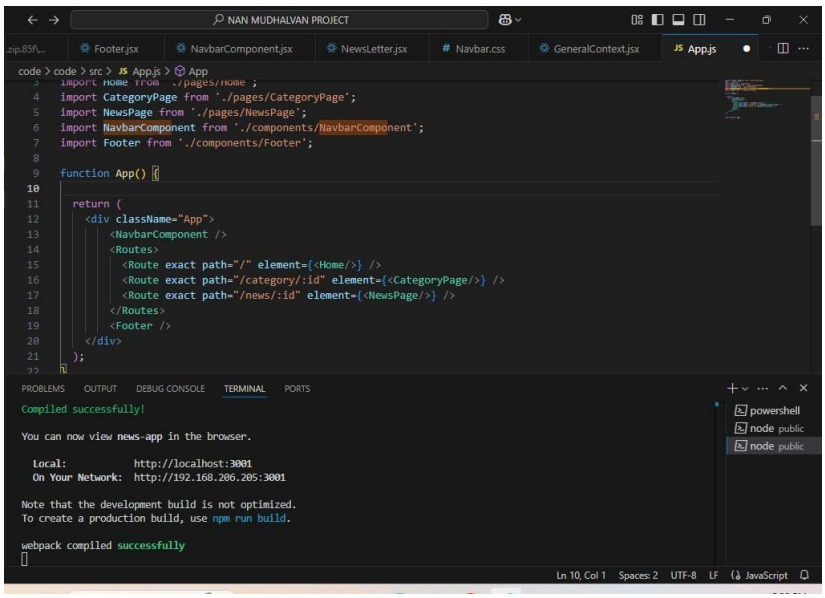
• You should see the applications homepage, indicating that the installation and setup were successful.

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.

5.Folder Structure

The project follows a structured and organized folder hierarchy to separate concerns and maintain modularity. The React frontend (client) and the Node.js backend (server) are kept in separate directories.





Summary

Client (React.js)

- components/ – Reusable UI components (Header, Footer, NewsCard, etc.).
- pages/ – Page-level components (Home, NewsDetails).
- context/ – Context API for managing global state.
- hooks/ – Custom React hooks for reusable logic.
- utils/ – Helper functions (e.g., date formatting).
- styles/ – CSS styles for the application.

Server (Node.js)

- data/ – Static JSON files for storing news articles.
- routes/ – API routes to serve news data.
- index.js – Entry point for setting up the Express.js server.

This folder structure ensures a well-organized, modular, and scalable codebase, making the project easy to maintain and extend. Let me know if you need any modifications!

6.Running the Application

Frontend: npm start in the client directory.

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.

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.

7. Component Documentation

This section provides details about **key components** and **reusable components** used in the React frontend of the news.

1. Key Components

1.1 App.js (Main Component)Purpose

```jsx
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Header from "./components/Header";
import Footer from "./components/Footer";
import NewsList from "./pages/NewsList";
import NewsDetails from "./pages/NewsDetails";

function App() {
  return (
    <Router>
      <Header />
      <Routes>
        <Route path="/" element={<NewsList />} />
        <Route path="/news/:id" element={<NewsDetails />} />
      </Routes>
      <Footer />
    </Router>
  );
}

export default App;
```

Header.js (Navigation Bar)

Purpose:

- Displays the website logo and navigation links for different news categories.
- Provides a **search bar** for users to search news articles.

```jsx
import { Link } from "react-router-dom";

const Header = () => {
  return (
    <header>
      <h1>News Website</h1>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/category/technology">Technology</Link>
        <Link to="/category/sports">Sports</Link>
      </nav>
    </header>
  );
};

export default Header;
```

NewsList.js (Displays List of News Articles)

Purpose:

- Fetches and displays a list of news articles from the backend.
- Renders multiple NewsCard components.

```jsx
import { useEffect, useState } from "react";
import NewsCard from "../components/NewsCard";

const NewsList = () => {
  const [news, setNews] = useState([]);

  useEffect(() => {
    fetch("http://localhost:5000/news")
      .then((response) => response.json())
      .then((data) => setNews(data))
      .catch((error) => console.error("Error fetching news:", error));
  }, []);

  return (
    <div>
      {news.map((article) => (
        <NewsCard key={article.id} {...article} />
      ))}
    </div>
  );
};

export default NewsList;
```

NewsDetails.js (Displays Full Article Details)

Purpose:

- Fetches and displays the full details of a news article when a user clicks on it.

Props:

- Retrieves **id** from the URL using useParams().

```jsx
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";

const NewsDetails = () => {
  const { id } = useParams();
  const [article, setArticle] = useState(null);

  useEffect(() => {
    fetch(`http://localhost:5000/news/${id}`)
      .then((response) => response.json())
      .then((data) => setArticle(data))
      .catch((error) => console.error("Error fetching news:", error));
  }, [id]);

  if (!article) return <p>Loading...</p>;

  return (
    <div>
      <h2>{article.title}</h2>
      <img src={article.imageUrl} alt={article.title} />
      <p>{article.content}</p>
    </div>
  );
};

export default NewsDetails;
```

Reusable Components

NewsCard.js (Displays Individual News Article Preview)

Purpose:

- Displays a **news article preview** with an image, title, and a short description.
- Clicking on it navigates to the full article page.

```jsx
import { Link } from "react-router-dom";

const NewsCard = ({ id, title, imageUrl, content }) => {
  return (
    <div className="news-card">
      <img src={imageUrl} alt={title} />
      <h3>{title}</h3>
      <p>{content.substring(0, 100)}...</p>
      <Link to={`/news/${id}`}>Read More</Link>
    </div>
  );
};

export default NewsCard;
```

Reusable Components:

Component	purpose
NewsCard.js	Displays individual news article previews.
SearchBar.js	Handles user search functionality.
Footer.js	Displays website footer.

8.State Management

State management is crucial in ensuring **smooth data flow** and **efficient updates** across the React application. In this project, we use a combination of **Global State** (managed using React Context API) and **Local State** (managed within individual components using useState).

1. Global State Management

Approach: React Context API

- The **React Context API** is used to **manage state globally** across multiple components.
- This allows components to **share data** (like the search query or selected category) without **prop drilling** (passing props deeply).

How State Flows in the Application:

1. **Context Provider (NewsContext.js)**:
    - Stores **global state** like searchQuery.
    - Provides **setter functions** to update state.
2. **Components (SearchBar.js, NewsList.js)**:
    - Use useContext(NewsContext) to **access and modify** the global state.
3. **When the user searches**:
    - SearchBar updates searchQuery in **global state**.
    - NewsList reads searchQuery and filters articles.

2. Local State Management

Approach: useState Hook

- **Local state** is managed **within individual components**.
- It is **used for UI interactions** and **component-specific data** that doesn't need to be shared globally.

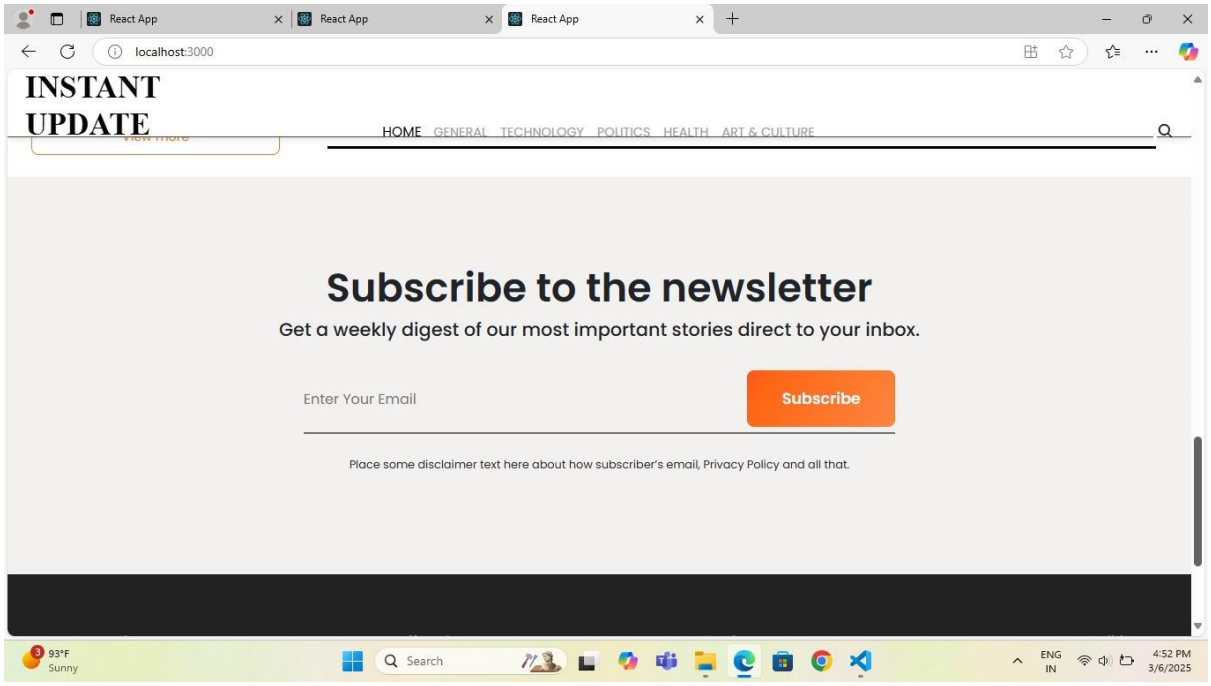
Handling User Input in a Search Bar

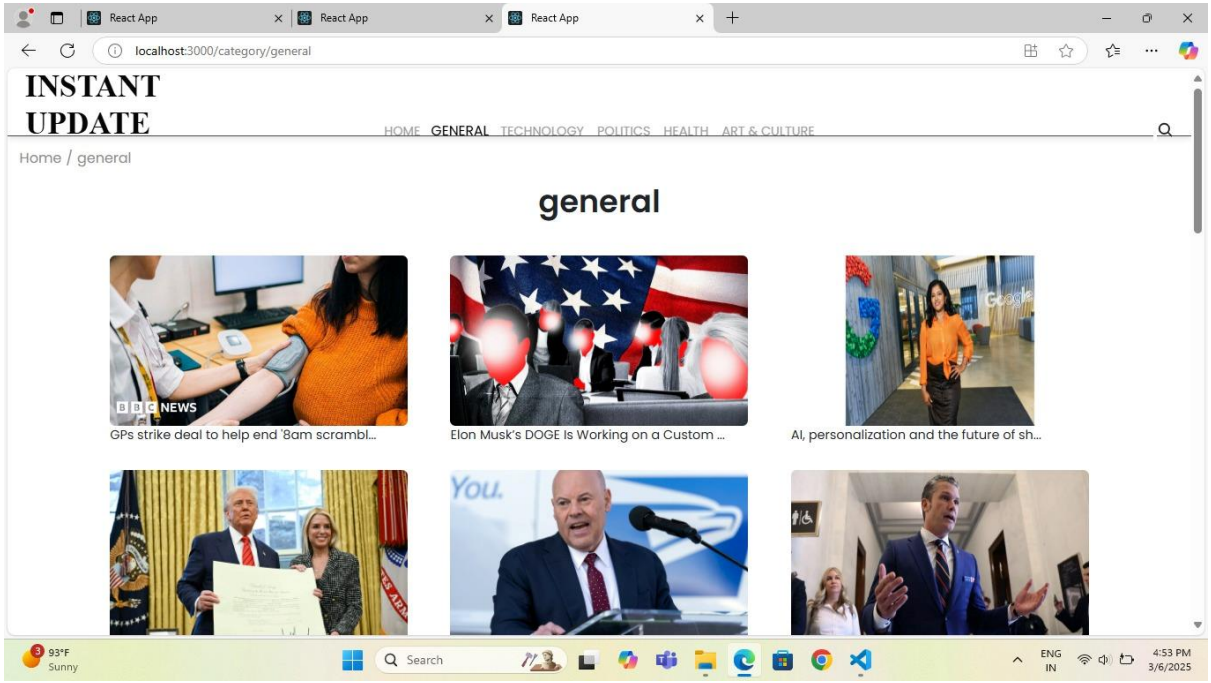
- Local state stores **input value** before sending it to the global state.

SearchBar

```jsx
import { useState } from "react";

const SearchBar = ({ onSearch }) => {
  const [inputValue, setInputValue] = useState(""); // Local state

  const handleChange = (event) => {
    setInputValue(event.target.value); // Update local state
    onSearch(event.target.value); // Pass value to parent component
  };

  return (
    <input
      type="text"
      placeholder="Search news..."
      value={inputValue}
      onChange={handleChange}
    />
  );
};

export default SearchBar;
```

9.User Interface


React App
localhost:3000
INSTANT UPDATE
HOME GENERAL TECHNOLOGY POLITICS HEALTH ART & CULTURE
Subscribe to the newsletter
Get a weekly digest of our most important stories direct to your inbox.
Enter Your Email
Subscribe
Place some disclaimer text here about how subscriber's email, Privacy Policy and all that.
93°F Sunny
Search
ENG IN
4:52 PM 3/6/2025


React App
localhost:3000/category/general
INSTANT UPDATE
HOME GENERAL TECHNOLOGY POLITICS HEALTH ART & CULTURE
Home / general
general
GPs strike deal to help end '8am scrambl...
Elon Musk's DOGE Is Working on a Custom ...
AI, personalization and the future of sh...
93°F Sunny
Search
ENG IN
4:53 PM 3/6/2025

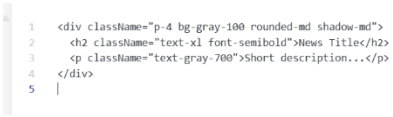
10.Styling

1. CSS Frameworks & Libraries

The project uses **modern styling techniques** to enhance the UI and user experience. Here are the key CSS frameworks/libraries used:

Tailwind CSS (if applicable)

- Utility-first framework for **faster styling without writing extra CSS.**



```
<div className="p-4 bg-gray-100 rounded-md shadow-md">
  <h2 className="text-xl font-semibold">News Title</h2>
  <p className="text-gray-700">Short description...</p>
</div>
```

Bootstrap (if applicable)

- Predefined styles for **buttons, grids, modals,**
- <button className="btn btn-primary">Read More</button>

11.Testing Strategy

To ensure the reliability and quality of the application, different testing strategies are implemented. The testing process includes **unit testing, integration testing, and end-to-end (E2E) testing** using tools like **Jest, React Testing Library, and Cypress**.

1. Testing Approach

- **Unit Testing (Component Testing)**
  - Ensures **individual components** work correctly in isolation.
  - Uses **Jest** and **React Testing Library**.
  - **Tests:**
    - Rendering without errors
    - Props and state changes
    - Event handlers (e.g., button clicks, input changes)

Integration Testing (Component Interaction Testing)

- Ensures **multiple components work together correctly**.
- Focuses on **user interactions and API calls**.
- Uses **React Testing Library** and **Mock Service Worker (MSW)** for API testing.

End-to-End (E2E) Testing

- Simulates **real user behavior** in the browser.
- Uses **Cypress** for UI testing.
- **Tests:**
  - Page navigation
  - Form submissions
  - API integration

Code Coverage

Code coverage is used to measure how much of the codebase is tested by automated tests. The goal is to ensure that **critical parts of the application are well-tested**, reducing the chances of bugs in production.

. Code Coverage Tools 🛠️

- **Jest Coverage (for Unit & Integration Tests)**

- Jest comes with built-in support for **code coverage reports**.
- To generate a coverage report, run:

  npm test -- --coverage
- This command outputs a summary showing the percentage of:
  - **Statements** covered
  - **Branches** (if-else conditions) tested
  - **Functions** tested
  - **Lines** of code executed

12.Screenshots or Demo

https://drive.google.com/drive/folders/10AcHrTeeqiKo815nefU3h8EceH1DykVY

13. Known Issues

This section documents any known bugs, limitations, or issues in the project that users or developers should be aware of.

- Current Known Issues

1. API Response Delay in Backend

- Issue: When fetching news data, there may be a delay in API responses, especially for large data requests.
- Cause: The backend processes requests sequentially without caching.
- Workaround: Implement response caching (e.g., using in-memory storage or middleware like compression for faster API responses).

2. UI Glitches in Mobile View

- Issue: Some UI components don't resize properly on smaller screens.
- Cause: The lack of proper responsive design adjustments.
- Workaround:
    - Use CSS media queries to fix layout shifting.
    - Test components on various screen sizes using Chrome DevTools.

3. Navigation Issues on Page Refresh

- Issue: On refreshing a page with a dynamic route (e.g., /news/:id), a 404 error occurs.
- Cause: React Router handles routes on the client-side, but refreshing makes the browser request a non-existent server route.
- Workaround:
    - Configure the backend to redirect all routes to index.html.
    - If using Express, add this in server.js:

```
app.get("*", (req, res) => {
  res.sendFile(path.join(__dirname, "client", "build", "index.html"));
});
```

4. State Persistence Issues

- Issue: Page reloads reset all local states, losing user-selected filters or settings.
- Cause: State is managed in React's local state instead of persistent storage.
- Workaround:
    - Store critical state in localStorage or sessionStorage.
    - Use React's Context API for global state management.

5. Console Warnings & Errors in Development Mode

- Issue: Some warnings appear in the console when running the app in development mode (e.g., missing key props in lists, deprecated functions).

14.Future Enhancements

This section outlines potential improvements and features that can be added to enhance the functionality, performance, and user experience of the project.

🔹 Planned Feature Enhancements

1. Enhanced UI & Animations

- Improve UI responsiveness for better **mobile compatibility**.
- Add **smooth animations** using Framer Motion or CSS transitions.
- Implement **interactive hover effects** on news cards.

2. Improved API Handling & Caching

- Implement **client-side caching** to reduce redundant API calls.
- Add **loading skeletons** to improve perceived performance.
- Optimize **backend API responses** with middleware like compression.

3. Search & Filtering Upgrades

- Implement **fuzzy search** to match keywords more effectively.
- Add **filtering by category, date, and popularity** for better user experience.
- Improve **pagination and infinite scrolling** for smoother navigation.

4. Dark Mode Implementation

- Provide a **toggle switch** for users to switch between light and dark mode.
- Store user preference using **localStorage**.

- Apply **dynamic theming** using CSS variables.

5. Accessibility (A11Y) Improvements

- Improve keyboard navigation and screen reader support.
- Use **semantic HTML elements** to enhance accessibility.
- Add **ARIA attributes** for better UI interaction.

6. Performance Optimization

- Implement **lazy loading** for images and components.
- Optimize assets (e.g., **minify CSS & JS, compress images**).
- Use **code splitting** to reduce initial load time.

7. Integration of AI-Based Features

- Add **AI-generated news summaries** for quick article previews.
- Use **NLP-based sentiment analysis** to categorize news content.
- Implement **voice-based search** for hands-free interaction.

8. Offline Mode & PWA Support

- Convert the project into a **Progressive Web App (PWA)**.
- Allow users to **read news offline** by caching articles.

Conclusion

These enhancements will improve **user experience, performance, and accessibility** while making the application more modern and feature-rich.