Running the application

1. Start the Backend Server

```
cd backend
npm start
```

The server will start on port 5000.

2. Start the Frontend Development Server

```
cd frontend
npm run dev
```

The frontend development server will start and the application will be available at `http://localhost:5173`

Backend Development

- The backend runs on Express.js
- Main server file: `server.js`
- API endpoint: `/api/news`
- Supports category filtering via query parameters

Frontend Development

- Built with React and Vite
- Main component: `src/App.js`
- Styles: `src/App.css`
- Supports responsive design

Backend (Server.js)

Importing Required Libraries and Modules:

```
const express = require('express');
```

- Imports the Express library, which helps build a web server.

```
const cors = require('cors');
```

- Imports CORS (Cross-Origin Resource Sharing) middleware to allow requests from other origins (e.g., frontend apps on different domains).

```
const axios = require('axios');
```

- Imports Axios, a library for making HTTP requests to external APIs or servers.

```
const { SummarizerManager } = require('node-summarizer');
```

- Imports a summarizer tool from the `node-summarizer` package to generate text summaries.

```
require('dotenv').config();
```

- Loads environment variables from a `.env` file into `process.env`. These variables are typically used for sensitive data like API keys.

Setting Up the Express App:

```
const app = express();
```

- Creates an Express application instance.

```
app.use(cors());
```

- Enables CORS middleware on the app, allowing cross-origin requests.

Setting Up Variables and Helpers:

```
const API_KEY = '2ebc5b505303468e85cd26cef759fb02';
```

- Stores the API key used to access the news API.

```
const summarizer = new SummarizerManager();
```

- Initializes a summarizer instance for generating summaries.

```
const getSummary = async (text, sentences = 2) => {
  if (!text) return '';
```

- Defines a helper function `getSummary` to generate a summary of the given `text` with the specified number of sentences (default is 2).
- Returns an empty string if no `text` is provided.

```
try {
  const result = await summarizer.summarize(text, sentences);
  return result.summary;
} catch (error) {
  return text.split(' ').slice(0, 50).join(' ') + '...';
}
```

- Tries to generate a summary using the `summarizer` library.
- If it fails (e.g., due to an error), it creates a basic fallback summary by taking the first 50 words of the `text`.

Defining the `/api/news` Endpoint:

```
app.get('/api/news', async (req, res) => {
```

- Defines a route for handling HTTP GET requests at `/api/news`.

```
const category = req.query.category || 'general';
console.log('Fetching news for category:', category);
```

- Retrieves the `category` query parameter from the request. If not provided, defaults to `'general'`.
- Logs the category being fetched (for debugging purposes).

```
try {
  const response = await axios.get(
    `https://newsapi.org/v2/top-headlines`, {
      params: {
        country: 'us',
        category: category,
        apiKey: API_KEY
      }
    }
  );
```

- Makes a request to the News API to fetch top headlines for the given category and country (`'us'`).
- The API key is included in the request to authenticate.

```
const processedArticles = await Promise.all(
  response.data.articles.map(async (article) => {
    const fullText = article.content || article.description || '';
    const summary = await getSummary(fullText);
```

```
    return {
      ...article,
      summary
    };
  })
);
```

- Processes the list of news articles returned by the API:
    1. Extracts `content` or `description` as the full text.
    2. Generates a summary using `getSummary`.
    3. Adds the summary to the article object.
    4. Returns the updated article.
- `Promise.all` ensures all articles are processed concurrently before continuing.

```
res.json({ ...response.data, articles: processedArticles });
```

- Responds with the updated news data, replacing the original articles with the summarized versions.

```
} catch (error) {
  console.error('Error fetching news:', error);
  res.status(500).json({ error: error.message });
}
```

- If an error occurs while fetching or processing news, logs it and sends a 500 status code with the error message.

Starting the Server:

```
app.listen(5000, () => {
  console.log('Server running on port 5000');
});
```

- Starts the Express server on port 5000.
- Logs a confirmation message to indicate the server is running.

Frontend (App.jsx)

Import Statements

```
import { useState, useEffect, useCallback } from 'react';
import './App.css';
import snapNewsLogo from './assets/snapnews-logo.png'; // Add your logo
file
```

- Importing `useState`, `useEffect`, and `useCallback` hooks from React for state management and side effects.
- Importing the CSS file `App.css` for styling.
- Importing the logo `snapNewsLogo` to use in the loading screen.

Categories

```
const CATEGORIES = [
  { id: 'general', label: 'General' },
  { id: 'business', label: 'Business' },
  { id: 'technology', label: 'Technology' },
  { id: 'entertainment', label: 'Entertainment' },
  { id: 'sports', label: 'Sports' },
  { id: 'science', label: 'Science' },
  { id: 'health', label: 'Health' }
];
```

- Defines a list of categories with an `id` for the API request and a user-friendly `label` for display in the UI.

Loading Screen

```
const LoadingScreen = () => (
  <div className="loading">
    <img
      src={snapNewsLogo}
      alt="SnapNews Logo"
      className="loading-logo"
    />
  </div>
);
```

- A functional component that displays the loading screen with the SnapNews logo.

App Component State

```
function App() {
```

```
  const [news, setNews] = useState([]);
  const [currentIndex, setCurrentIndex] = useState(0);
  const [loading, setLoading] = useState(true);
  const [category, setCategory] = useState('general');
```

- **`news`**: Stores the list of news articles fetched from the API.
- **`currentIndex`**: Tracks the index of the currently displayed article.
- **`loading`**: Indicates whether the app is in a loading state.
- **`category`**: Stores the currently selected news category (default is `'general'`).

Fetch News Function

```
const fetchNews = useCallback(async () => {
  try {
    setLoading(true);
    console.log('Fetching news for category:', category);
    const response = await
fetch(`http://localhost:5000/api/news?category=${category}`);
```

- `fetchNews`: An asynchronous function that fetches news articles based on the selected `category`.
- `useCallback`: Ensures the function isn't re-created unnecessarily when dependencies (`category`) change.
- Sends a GET request to the backend API with the selected category.

```
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    const data = await response.json();
    if (data.articles && Array.isArray(data.articles)) {
      setNews(data.articles);
      setCurrentIndex(0);
    } else {
      console.error('Invalid data format:', data);
    }
```

- Validates the API response:
  - If successful, updates the `news` state with the fetched articles and resets the `currentIndex` to `0`.
  - Logs an error if the response format is invalid.

```
  } catch (error) {
    console.error('Error fetching news:', error);
  } finally {
    setLoading(false);
  }
}, [category]);
```

- Catches and logs any errors that occur during the API request.
- Always sets `loading` to `false` after the request completes.

Effect Hook

```
useEffect(() => {
  fetchNews();
}, [fetchNews]);
```

- Calls `fetchNews` whenever the component mounts or the `fetchNews` function changes.

Scroll Event Handler

```
const handleScroll = (e) => {
  if (e.deltaY > 0 && currentIndex < news.length - 1) {
    setCurrentIndex(prev => prev + 1);
  } else if (e.deltaY < 0 && currentIndex > 0) {
    setCurrentIndex(prev => prev - 1);
  }
};
```

- Handles scrolling:
  - Scroll down (`deltaY > 0`) moves to the next article.
  - Scroll up (`deltaY < 0`) moves to the previous article.

Category Change Handler

```
const handleCategoryChange = (newCategory) => {
  console.log('Changing category to:', newCategory);
  setCategory(newCategory);
};
```

- Updates the `category` state when a user selects a different news category.

Source Click Handler

```
const handleSourceClick = (url) => {
  if (url) {
    window.open(url, '_blank', 'noopener noreferrer');
  }
};
```

- Opens the news source in a new browser tab.

Loading State

```
if (loading) {
  return <LoadingScreen />;
}
```

- Displays the loading screen if the app is in the `loading` state.

Rendering the App

```
<div className="app" onWheel={handleScroll}>
  <div className="categories-container">
    {CATEGORIES.map(cat => (
      <button
        key={cat.id}
        className={`category-button ${category === cat.id ? 'active' :
''}`}
        onClick={() => handleCategoryChange(cat.id)}
      >
        {cat.label}
      </button>
    ))}
  </div>
```

- Renders:
  - A scrollable container with event handling for article navigation.
  - A list of category buttons that highlight the active category.

```
  {news.length > 0 && (
    <div className="story-container">
      <div className="story-card">
        <h2 className="title">{news[currentIndex].title}</h2>
```

- Displays the current article's title and content based on `currentIndex`.

Image Handling

```
<div className="image-container">
  {news[currentIndex].urlToImage ? (
    <img
      src={news[currentIndex].urlToImage}
      alt={news[currentIndex].title}
      className="news-image"
      onError={(e) => {
        e.target.onerror = null;
        e.target.parentElement.innerHTML = '<div class="placeholder-
image">No image available</div>';
      }}
    />
  ) : (
    <div className="placeholder-image">No image available</div>
```

```
  )}
</div>
```

- Displays the article's image:
    - If unavailable or fails to load, shows a placeholder.

Metadata and Source

```
<p className="source">
  Source:{' '}
  <a
    href={news[currentIndex].url}
    target="_blank"
    rel="noopener noreferrer"
    className="source-link"
  >
    {news[currentIndex].source?.name || 'Unknown'}
  </a>
</p>
```

- Displays metadata (source and publication date) with links to the original article.

Default Export

```
export default App;
```

- Exports the `App` component for rendering in `index.js`.

Frontend (App.css)

Global Styles

```
.app {
  width: 100vw;
  height: 100vh;
  background-color: #f0f2f5;
  display: flex;
  flex-direction: column;
  align-items: center;
  overflow: hidden;
  padding: 20px;
}
```

- `.app`: Styles the main container of the application.
  - Takes up the full viewport width and height (`100vw`, `100vh`).
  - Sets a light gray background color (`#f0f2f5`).
  - Uses Flexbox to organize child elements in a column and centers them horizontally (`align-items: center`).
  - Prevents scrolling (`overflow: hidden`) and adds padding for spacing.

Loading Screen

```
.loading {
  width: 100vw;
  height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #f0f2f5;
}
```

- `.loading`: Styles the loading screen to fill the viewport and center content.
  - Light gray background matches the app's overall theme.
  - Centers the loading logo using Flexbox.

```
.loading-logo {
  width: 200px;
  height: auto;
  animation: pulse 1.5s ease-in-out infinite;
}
```

- `.loading-logo`: Sets the logo's size and applies a pulsating animation.
  - Width is fixed at `200px`, height adjusts automatically to maintain aspect ratio.
  - Applies the `pulse` animation for a glowing effect.

```
@keyframes pulse {
  0% { opacity: 0.6; transform: scale(0.98); }
  50% { opacity: 1; transform: scale(1); }
```

```
  100% { opacity: 0.6; transform: scale(0.98); }
}
```

- `@keyframes pulse`: Defines the pulsating animation:
  - Alternates between dimmed (`opacity: 0.6`) and brightened (`opacity: 1`) states.
  - Slightly scales the logo up and down for emphasis.

Categories Section

```
.categories-container {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
  margin-bottom: 20px;
  justify-content: center;
  max-width: 100%;
  padding: 0 20px;
}
```

- `.categories-container`: Arranges category buttons.
  - Flexbox layout allows wrapping to new lines if space is limited (`flex-wrap: wrap`).
  - Adds gaps between buttons (`gap: 10px`).
  - Centers the container and limits its width for responsiveness.

```
.category-button {
  padding: 8px 16px;
  border: none;
  border-radius: 20px;
  background-color: #e4e6eb;
  color: #1a1a1a;
  cursor: pointer;
  transition: all 0.2s ease;
  font-size: 0.9rem;
}
```

- `.category-button`: Styles individual buttons.
  - Rounded edges (`border-radius: 20px`).
  - Light gray background with dark text for readability.
  - Smooth hover and focus transitions (`transition: all 0.2s ease`).

```
.category-button:hover {
  background-color: #dce0e6;
}

.category-button.active {
  background-color: #1a1a1a;
  color: white;
}
```

- `.category-button:hover`: Changes background on hover for interactivity.

- `.category-button.active`: Highlights the selected button with a dark background and white text.

Story Container

```
.story-container {
  width: 100%;
  max-width: 400px;
  height: calc(100vh - 120px);
  background-color: white;
  border-radius: 15px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  overflow: hidden;
}
```

- `.story-container`: Defines a card for displaying the current story.
  - Fixed height, minus space for navigation and paddings.
  - Rounded corners and subtle shadow for a modern look.

```
.story-card {
  width: 100%;
  height: 100%;
  display: flex;
  flex-direction: column;
  padding: 20px;
  box-sizing: border-box;
}
```

- `.story-card`: Styles the inner card layout.
  - Uses Flexbox for vertical stacking of content (title, image, summary, metadata).

Story Details

```
.title {
  font-size: 1.5rem;
  margin: 0 0 15px 0;
  color: #1a1a1a;
  line-height: 1.3;
}
```

- `.title`: Styles the news title with bold font and proper spacing.

```
.image-container {
  width: 100%;
  height: 200px;
  background-color: #f0f2f5;
  border-radius: 10px;
  overflow: hidden;
  margin-bottom: 15px;
}
```

```
.news-image {
  width: 100%;
  height: 100%;
  object-fit: cover;
}

.placeholder-image {
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #e4e6eb;
  color: #65676b;
}
```

- `.image-container`: Holds the news image or a placeholder if unavailable.
- `.news-image`: Ensures the image covers the container proportionally.
- `.placeholder-image`: Displays text when no image is available.

Story Explanation

```
.explanation {
  font-size: 1rem;
  line-height: 1.5;
  color: #333;
  margin: 15px 0;
  flex-grow: 1;
  overflow-y: auto;
  padding-right: 10px;
}
```

- `.explanation`: Styles the news summary with scrolling enabled.

```
.explanation::-webkit-scrollbar { width: 6px; }
.explanation::-webkit-scrollbar-track { background: #f1f1f1; }
.explanation::-webkit-scrollbar-thumb { background: #888; }
.explanation::-webkit-scrollbar-thumb:hover { background: #555; }
```

- Customizes the appearance of the scrollbar for a clean UI.

Metadata

```
.metadata {
  margin-top: auto;
  border-top: 1px solid #e4e6eb;
  padding-top: 15px;
}
.source, .date {
  font-size: 0.9rem;
  color: #65676b;
}
.source-link {
```

```
  color: #1a73e8;
  text-decoration: none;
  transition: color 0.2s ease;
}
.source-link:hover {
  color: #1557b0;
  text-decoration: underline;
}
```

- `.metadata`: Adds a section for source and publication date.
- `.source-link`: Styles clickable source links with hover effects.

Responsiveness

```
@media (max-width: 768px) { ... }
@media (max-width: 480px) { ... }
```

- Adjusts styles for tablets and smaller devices:
  - Reduces padding and font sizes.
  - Scales down images and logo sizes for smaller screens.

Frontend (main.jsx)

---

1. **`import React from 'react'`**

- Imports the core React library, which is necessary to use JSX (JavaScript XML) and create React components.

2. **`import ReactDOM from 'react-dom/client'`**

- Imports ReactDOM, a library used for rendering React components into the DOM.
- Specifically, this imports the `client` version, which is used for rendering React applications using the newer `createRoot` API introduced in React 18.

3. **`import App from './App'`**

- Imports the main application component (`App`) from the `App` module, located in the same directory (`./`).
- This is the root component of your application, where the structure and functionality of your app are defined.

4. **`import './index.css'`**

- Imports the global CSS styles from the `index.css` file.
- These styles apply to the entire application and are typically used for global resets, fonts, and base styling.

5. **`ReactDOM.createRoot(document.getElementById('root')).render(...)`**

- **`document.getElementById('root'):`**
  - Selects the DOM element with the `id` of `root`. This is the container in the `index.html` file where the React app will be mounted.
- **`ReactDOM.createRoot(...):`**
  - Creates a React root for the selected DOM element. This root enables features like concurrent rendering and is required for React 18 and above.
- **`.render(...):`**
  - Specifies what to render inside the root. In this case, it renders the `App` component wrapped in `<React.StrictMode>`.

6. **`<React.StrictMode>`**

- A wrapper provided by React that activates additional checks and warnings for detecting potential problems in an application (e.g., deprecated features, side effects, etc.).
- It does not affect the production build but helps developers catch issues during development.

7. **`<App />`**

- Represents the main component of your React application.
- This is where all child components and app logic will be included.

How It Works:

1. The `index.html` file contains an empty `<div id="root"></div>`.
2. This script selects the `root` div, creates a React root, and renders the `App` component inside it.
3. The `App` component becomes the entry point for the rest of the application.

Example Context:

- If `App` is a simple component:

```
// App.jsx
import React from 'react';

function App() {
  return <h1>Hello, World!</h1>;
}

export default App;
```

- When this application runs, the browser will display "Hello, World!" inside the `<div id="root"></div>`.

Frontend (index.css)

---

1. Universal Selector (`*`)

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

- **`*`**: The universal selector applies styles to all elements on the page.
- **`margin: 0;`**: Removes the default margin applied by browsers to many HTML elements (e.g., `<body>`, `<h1>`).
- **`padding: 0;`**: Removes the default padding applied by browsers to many HTML elements (e.g., `<ul>`, `<ol>`).
- **`box-sizing: border-box;`**: Ensures that the width and height of elements include padding and borders, making layout calculations more predictable. Without this, padding and borders are added *outside* the defined width/height, potentially causing layout issues.

This rule ensures a consistent baseline for styling across all elements, eliminating browser-specific inconsistencies.

---

2. `body` Selector

```
body {
    font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
Oxygen,
      Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
}
```

- **`body`**: Targets the `<body>` element of the webpage, which contains all visible content.
- **`font-family`**:
  - Specifies a list of fonts to be used for the text in the document.
  - Fonts are listed in order of preference. If the first font isn't available, the browser tries the next one, and so on.
  - The list includes:
    - **`-apple-system`**: The default system font for Apple devices.
    - **`BlinkMacSystemFont`**: The default system font for Chrome on macOS.
    - **`'Segoe UI'`**: The default system font for Windows.
    - **`Roboto`**, **`Oxygen`**, **`Ubuntu`**, **`Cantarell`**, **`'Open Sans'`**, **`'Helvetica Neue'`**: Popular web-safe fonts that are widely supported across platforms.
    - **`sans-serif`**: A generic fallback for sans-serif fonts in case none of the specified fonts are available.

This rule ensures the text on the page looks modern, clean, and consistent across different devices and operating systems.

Purpose:

- **Universal Reset**: Eliminates default margins and paddings and sets a consistent box model.
- **Consistent Font Styling**: Applies a modern, readable font stack across the entire webpage, providing a better user experience.

Result:

- A consistent, clean slate for further styling. This approach is a common best practice in modern web development.

Frontend (index.html)

<!doctype html>

- Declares the document type as HTML5.
- Informs the browser that the document follows HTML5 standards, ensuring proper rendering.

<html lang="en">

- **<html>**: The root element of the HTML document.
- **lang="en"**: Specifies the primary language of the document as English, aiding accessibility tools like screen readers and search engines.

<head>

The <head> section contains metadata and resources for the document.

```
<meta charset="UTF-8" />
```

- Sets the character encoding to UTF-8, which supports most characters used in writing systems worldwide.
- Ensures proper rendering of special characters.

```
<link rel="icon" type="image/svg+xml" href="/logo.svg" />
```

- **rel="icon"**: Specifies that this link is for the browser's favicon (the small icon displayed in the browser tab).
- **type="image/svg+xml"**: Indicates that the favicon is an SVG image.
- **href="/logo.svg"**: Path to the favicon file.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

- Configures the viewport for responsive design.
    - **width=device-width**: Sets the width of the viewport to match the device's screen width.
    - **initial-scale=1.0**: Ensures the page's initial zoom level is 1 (default size).

```
<title>SnapNews</title>
```

- Sets the title of the document as "SnapNews," displayed in the browser tab.

<body>

The `<body>` section contains the visible content and scripts for the page.

```
<div id="root"></div>
```

- A container `<div>` with an ID of `root`.
- Acts as the mounting point for the React application. React renders its components inside this element.

```
<script type="module" src="/src/main.jsx"></script>
```

- **`<script>`**: Includes a JavaScript file.
- **`type="module"`**: Indicates the script is a JavaScript module, enabling the use of ES6+ features like `import` and `export`.
- **`src="/src/main.jsx"`**: Path to the main JavaScript file, typically where the React application is initialized and rendered into the `#root` div.

---

1. **Metadata Setup**: Ensures proper character encoding, responsive design, and branding via a favicon.
2. **Mount Point for React**: Provides an element (`#root`) where React can render the application.
3. **JavaScript Module**: Loads the main JavaScript file that initializes the React app.

This structure is standard for modern single-page applications (SPAs) built with React.