Project Description

In this project, you will use your React skills to create a news feed app. The app will have the following capabilities:

1. Use News API to get news articles
2. Render list of articles based on news API results
3. Provide a search bar to search articles by keywords
4. Provide Pagination (Next & Previous buttons) to navigate through more results
5. Provide Category dropdown to filter results based on category.
6. Improve the Error handling experience; Show error messages when loading data fails.
7. Improve Loading experience; use splash screen while data is loading.
8. Style using Material UI components (or any other major styling library of your choice like Bootstrap)

Project Repo

For your reference, the project implementation can be found in https://github.com/almdrasa/react-newsfeed_app/tree/main. You can download the project for your reference.

Your actual implementation doesn't need to match the styling or the component structure of the solution 100%. The main goal is to satisfy the requirements described above. Styling is "Nice-to-have", but not required.

Project Setup

The project is set up using Vite with JavaScript. You will find the base project setup attached and you can use it directly.

If you want to create your own project from scratch, here are the steps to do so

```
$ npm create vite@latest
```

Follow the steps to create a new app directory. Navigate to that directory then install the needed dependencies

```
$ npm install @emotion/react @emotion/styled @mui/icons-material
@mui/material @mui/styled-engine-sc @fontsource/roboto lodash
styled-components
```

Implementation Steps

After the initial setup, you need to define the main components that will be used in the App. Please try to make your code clean and modular. Think of what are the main units that build your app. They don't need to be too small, but please avoid putting all your code in <App /> component.

News Article (Post)

Each news Article (single post) must contain the following information

1. Title
2. Description
3. Author
4. Publication date
5. Image (if any)

Please make sure your code doesn't break if any of these pieces is missing. For example: If the article has no image, the app shouldn't crash. Instead, the image will not be displayed for that article.

Article styling

You can use Material UI to style your articles. Material UI is a library of React components that come pre-styled and give you easy access to styling through component props and / or through Styled component (Explained in Styling section in React Basics course).

Here is a sample Article component with Material UI styling. The details are just static data. Your implementation of such component is expected to be more dynamic and data-driven (renders the data coming through props).

```
import Card from "@mui/material/Card";
import CardActionArea from "@mui/material/CardActionArea";
import CardContent from "@mui/material/CardContent";
import CardMedia from "@mui/material/CardMedia";
import Typography from "@mui/material/Typography";
import Box from "@mui/material/Box";
import { styled } from "@mui/material/styles";

const StyledCard = styled(Card)(({ theme }) => ({
  margin: theme.spacing(2, 0),
}));

function NewsArticle() {
  return (
    <StyledCard>
      <CardActionArea>
        <CardMedia component="img" height="200" image="https://placeholder.co/150"
alt="Sample article" />
        <CardContent>
          <Typography gutterBottom variant="h6" component="div">
            Sample Article (Title)
          </Typography>
          <Typography variant="body2" color="textSecondary">
            This is a sample article (Description)
          </Typography>
        </CardContent>
      </CardActionArea>
      <Box p={2}>
        <Typography variant="caption" color="textSecondary" display="block">
          John Doe (Author)
        </Typography>
        <Typography variant="caption" color="textSecondary">
          Tuesday October 3rd, 2023
        </Typography>
      </Box>
    </StyledCard>
  );
}
```

You can read more about Material UI styling options. Feel free to change that styling as you see fit. This is just an example for demonstration purposes only.

Sample Data

To test your app and styling, you can define a static array of articles like that

```
const sampleArticles = [
  {
    title: "Test News 1",
    description: "This is a test news description 1.",
    image: "https://placehold.co/150",
    author: "John Doe",
    publishedAt: "2023-03-01T12:00:00Z",
  },
  {
    title: "Test News 2",
    description: "This is a test news description 2.",
    image: "https://placehold.co/150",
    author: "Jane Smith",
    publishedAt: "2023-04-01T12:00:00Z",
  },
  {
    title: "Test News 3",
    description: "This is a test news description 3.",
    image: "https://placehold.co/150",
    author: "John Doe",
    publishedAt: "2023-05-01T12:00:00Z",
  },
  {
    title: "Test News 4",
    description: "This is a test news description 4.",
    image: "https://placehold.co/150",
    author: "Jane Smith",
    publishedAt: "2023-06-01T12:00:00Z",
  },
];
```

News API

We will be using https://newsapi.org/ API to fetch news articles and their details. In order to use the API, you need to obtain an API key [PLEASE DO NOT SHARE YOUR API KEYS IN ANY PUBLIC GITHUB REPO].

Follow the Get API button and fill out the information as needed

Once you are done, you will be directed to a page with your API Key. Take a copy of that Key and store it in a local file on your machine.

Why API Key is important to keep secret

API Key is the way the API owners can identify their API users. It is also a common way to charge the API users based on their billing scheme. If you share your API Key publicly, you allow other developers to use your key and their API quota and usage will be counted as your own usage.

How to use the API Key in an app without making it shared on GitHub repo?

To keep your API Key safe and still use it in your app codebase, please follow the following steps

1. Create a new file at the codebase root directory named ".env". This is a special file known to Vite to store any environment variables that you need to access in your project.
2. Add the following line to your .env file.
    a. Please note the **`VITE_`** prefix for the environment variable is required here. Otherwise, the environment variable will be ignored by Vite. You can read more about Vite environment variables here https://vitejs.dev/guide/env-and-mode.html
    b. Replace **`<your_api_key>`** with the actual key you got from NewsApi
3. Whenever you need to use the API Key in your codebase, replace it with this code instead: **`import.meta.env.VITE_NEWS_API_KEY`**
    a. Example: if you add this statement to your `App` component `console.log(import.meta.env.VITE_NEWS_API_KEY)`, it will print the API key in the console log
    b. This reads the API key value from the ".env" file. The UPPER_CASE name must match what you define in the ".env" file.
4. Add .env to your .gitignore file, this prevents git from tracking and committing this file.

Fetch news from News API

We will be using the fetch API (Javascript built-in function), as explained in the "React Deep-dive" course. Based on NewsAPI documentation, we will use the top-headlines API described here https://newsapi.org/docs/endpoints/top-headlines

Based on the documentation, the URL to call the API will look like this (using Javascript string template)

```
`https://newsapi.org/v2/top-headlines?country=eg&apiKey=${import.meta.env.VITE_NEWS_API_KEY}`
```

The response of that API looks like this

```
{
  "status": "ok",
  "totalResults": 38,
  "articles": [
    {
      "source": {
        "id": null,
        "name": "Yahoo Entertainment"
      },
      "author": "Yahoo Sports Staff",
      "title": "London game tracker: Jaguars Bills England - Yahoo Sports",
      "description": "The Jaguars are looking for London.",
      "url": "https://sports.yahoo.com/travel-to-england.html",
      "urlToImage": "https://s.yimg.com/ny/2023-10/4e2a6500-3e06eaf",
      "publishedAt": "2023-10-08T14:45:46Z",
      "content": "Stefon Diggs is ready to put on a show in London in Week 5. (Photo by Timothy T Ludwig/Getty Images) (Timothy T Ludwig via Getty Images)\r\nFor at least one week, the Jacksonville Jaguars are the toast... [+631 chars]"
    },
    { … Another article }
    … Other articles
  ]
}
```

In order to use fetch API correctly, please check the following steps

1. Make sure you put the URL correctly (using your API Key)
2. In this URL example, I changed the country from “us” (United States) to “eg” (Egypt). You can use any country you need.
   a. **BONUS**: Add a country selector to your News Feed component and fetch new data when the country changes.
   b. List of supported countries is in the API documentation
3. To get the data from the fetch API response, you need to call **`response.json()`**. This is described in the “React Deep-dive” course.
4. The articles array is not the entire “data” of the response. After you get data from response.json(), you can get the articles by calling data.articles. Example code looks like this

```
const response = await fetch(
`https://newsapi.org/v2/top-headlines?country=eg&apiKey=${import.meta.env.VITE_NEWS_API_KEY}`);
const data = await response.json();
return data.articles
```

Please make sure that you map each article from the articles array correctly to the expected properties you use to render each article component. Can you spot the difference between the properties of the Article in the API response compared to Sample Articles? (Hint: One of the properties has a different name)

State Management

In order to load the data successfully, you may need to consider using the useEffect react hook to avoid fetching data from the API on every component rerender. (Do you remember how to set up the hook dependency array so that it only executes once?)

You will want to use the useState react hook to store the results of the API call. You will also need to handle the initial case (while data is loading), where no results are there yet. Depending on how you define the initial state of articles, the News feed needs to make sure it can handle an undefined, null or empty list of articles.

To make the loading state more visually appealing, you can use Material UI Circular Progress to render instead of a news feed list of articles while the app is still waiting for the API response. This is an example of how Circular progress can be used

```
import CircularProgress from "@mui/material/CircularProgress";

… rest of your code

if (!articles?.length) {
  return (
    <Box
      display="flex"
      justifyContent="center"
      alignItems="center"
      height="50vh"
    >
      <CircularProgress />
    </Box>
  );
}
```

This is just an example. The actual implementation and conditional statement depends on your app logic and how it handles the initial state of articles in the app.

Search Functionality

To implement search correctly, you need to add a search bar to your header section. The search bar with proposed styling can use Material UI library and could look like this

```
import InputBase from "@mui/material/InputBase";
import SearchIcon from "@mui/icons-material/Search";
import { styled } from "@mui/material/styles";
```

```
const Search = styled("div")(({ theme }) => ({
  position: "relative",
  borderRadius: theme.shape.borderRadius,
  backgroundColor: theme.palette.common.white,
  "&:hover": {
    backgroundColor: theme.palette.common.white,
  },
  marginLeft: "auto",
  width: 200,
}));

const SearchIconWrapper = styled("div")(({ theme }) => ({
  padding: theme.spacing(0, 2),
  height: "100%",
  position: "absolute",
  pointerEvents: "none",
  display: "flex",
  alignItems: "center",
  justifyContent: "center",
}));

const StyledInputBase = styled(InputBase)(({ theme }) => ({
  color: theme.palette.text.primary,
  "& .MuiInputBase-input": {
    padding: theme.spacing(1, 1, 1, 5),
    transition: theme.transitions.create("width"),
    width: "100%",
  },
}));
```

And the JSX would look like this

```
<AppBar position="static">
  <Toolbar>
    <Typography variant="h6">NewsFeed App</Typography>
    <Search>
      <SearchIconWrapper>
        <SearchIcon color="action" />
      </SearchIconWrapper>
      <StyledInputBase
        placeholder="Search…"
        inputProps={{ "aria-label": "search" }}
        onChange={handleInputChange}
      />
    </Search>
  </Toolbar>
</AppBar>
```

Feel free to use this style or change it as you see fit. This is just a demonstration and gives you quick access to a styled search bar implementation. You need to handle user input and pass the search query correctly to the fetch API.

You also need to make sure you change your useEffect dependency so it triggers a new query when the user input changes. How can you handle the loading state differently to distinguish loading from “No results” data?

Problem with the number of API calls

This App has a big problem with the number of API calls it makes. If the user types 5 letters in the search bar, the app makes 5 consecutive API calls and only the last API call is the important one. The others are wasted and unused.

As a front-end engineer, it’s important that you try to reduce the number of API calls you make for two reasons

1. Avoid overloading the backend with many unused API calls.
2. Avoid making unnecessary network calls to save client bandwidth and data

You can improve the number of the requests made to the server in two ways

1. There is a known front-end technique named “Debounce” where instead of making a new API request every time the user presses a new key, the Debounce waits for the user to finish typing (times out if the user stops typing for some duration), then after that timeout, it sends an API request with what the user typed so far.
    a. So if the user types 5 letters in the search bar, instead of sending 5 requests. Debounce waits till the user stops typing and only 1 request is sent after the 5 letters have been typed.
2. **[BONUS]** Cancel (abort) current API calls if a new request is triggered. Javascript fetch API allows you to cancel current requests if they didn’t finish. This protects the client from rendering outdated information and can possibly save the server from sending a response that is no longer needed.
    a. Search for fetch API Abort controller to understand how abort works https://developer.mozilla.org/en-US/docs/Web/API/AbortController
    b. Can you change the useEffect logic to handle this? (nit: useEffect cleanup function will help here)

c. What react hook can you use to store the AbortController so you don't lose it when the component rerenders?

Debounce Implementation

There is a popular JS library named lodash that has a debounce function that you can directly use. Firsty you need to install the library through

```
npm install lodash
```

Then you need to import and use debounce. Follow lodash documentation found here: https://lodash.com/docs/#debounce

IMPORTANT: debounce returns a function. Since useEffect will rerender the component every time the user query changes, the debounce function needs to be stored using useCallback so that the same function is returned with every rerender of the component in order for the debounce to work correctly.

The code will look something like this

```
import { debounce } from "lodash";

const fetchArticles = useCallback(debounce((query) => {
  … Previous content of the useEffect
}, 500), []);

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

If you look carefully at the useCallback dependency array, you will notice it is empty. This means it is created once when the component is mounted, then every other rerender of the component uses the same function. This ensures the debounce effect works as useEffect calls the same function every time.

Without useCallback, every rerender creates a new debounced function and calls it, so the previously debounced functions (from previous renders) will time out eventually and they will all execute in the sequence they were created and called.

Please note that debounce needs to take all the variables as inputs to its callback function. Since we're using useCallback, we need to pass all the variables as inputs. If you read state information directly from the debounced function implementation, useCallback will take a snapshot of such state values (query for example) and will ignore any changes that happen to this value in the future.

Implement Pagination

News API allows for requesting more information through pagination. Since the API results are huge, the API doesn't return all the results it has at once. This is standard API design for multiple reasons:

1. Reduces network bandwidth
2. Increases app performance; Loading huge amount of data can slow down your app
3. Reduces server load; Loading large dataset in one operation can overlord the server.
4. Faster response; less data means faster server execution and faster response. Instead of getting stuck at loading state for too long.

For these reasons, APIs that are expected to return a lot of results split the results into chunks (pages) of data. Each page has a page number and a page size (how many results you need to return per page). The API usually limits the page size to a certain maximum to avoid the problems mentioned above. News API documentation states that the maximum page size allowed is 100. This means it cannot return more than 100 articles in a single request.

We need to add a UI for navigating through pages of data. User needs to be able to go to the next page or previous page. Each next page click, loads the following page from the API. If you are at page 1 and click, the next API call needs to request page 2 and so on.

You need to handle the following corner cases

1. Disable the next button if the page result is less than 20 (Default page size) as it means there is no more data available for the next pages.
2. Disable the previous button if the current page is page 1 (First page)
3. Disable both buttons when results are loading

The API URL would look like this

```
`https://newsapi.org/v2/top-headlines?country=eg&q=${query}&page=${page}&apiKey=${import.meta.env.VITE_NEWS_API_KEY}`
```

`page` is a numeric value that starts from **1** (First page)

Handling API Errors

Errors are an expected part of any app. Especially when the app attempts to make API calls. Errors can happen for a variety of reasons like

1. Invalid user credentials (username or password)
2. User doesn't have permission to access the API
3. Invalid endpoint: API URL is invalid
4. Network error: Server is down or unreachable
5. Rate limiting errors: Server is no longer accepting requests from this user
6. Invalid request parameters

7. Invalid http method: API endpoint expects a GET method and client sent a POST request instead
8. Not found error: API is looking for a record that couldn't be found
9. Server error: Server encountered a problem while trying to execute the request.
10. Timeout error: Execution has taken more time than the client could wait

For these and other reasons, a well designed app is expected to handle errors gracefully. It shouldn't crash or present unexpected behavior. It's even better if the app could show the user an indication that it has ended up with an error and show friendly error details to the user.

Now, we need to handle News API errors gracefully. We will do so in a few steps.

1. We need a JSX element to render error messages. We can use Material UI Typography component to render styled text (or you can use your own native <span> or <p> elements with custom CSS styling.
2. We need to capture the errors from API calls. If you're using an Async function to load the data, you need to throw the error with a descriptive error message from that function and handle it with a Catch block when you call the loadData function.
3. API Errors come in two ways:
   a. API throws an error, you need to wrap fetch API with try / catch block. It should be caught by promise.catch where you call loadData function.
   b. The API call doesn't throw an error but returns error code in the response. News API does that per their documentation https://newsapi.org/docs/errors. In this case, we need to explicitly check response status code and throw an error to catch all errors in a consistent way like so

```
function loadData() {
  // ... loadData previous code
  const response = await fetch(
    query ? `${baseEndpoint}&q=${query}` : baseEndpoint,
      {signal: abortController.current.signal}
  );
  const data = await response.json();
  if (data.status !== "ok") {
    throw new Error(`Error: ${data.code}, Details: ${data.message}`);
  }
  // ... rest of the code
});
```

4. Then you need to handle setting that error when the function hits the catch block like so

```
loadData(query, page)
  .then((articles) => {
    setLoading(false);
    setArticles(articles);
  })
```

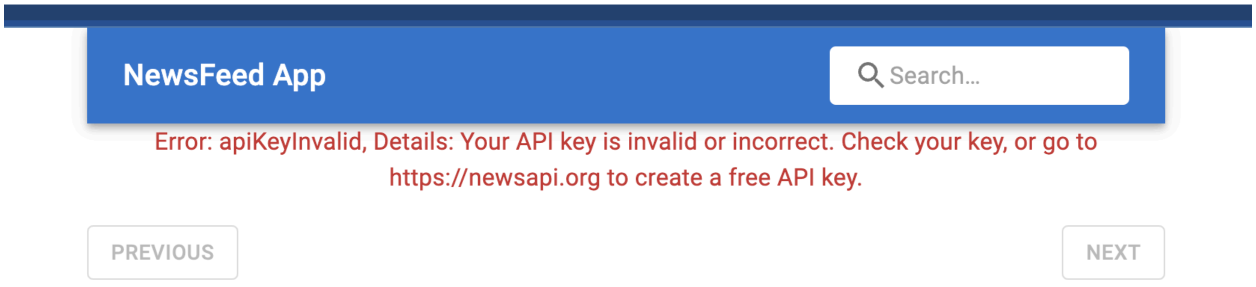
```
.catch((error) => {
  setLoading(false);
  setError(error.message);
});
```

5. Finally we need React to do its magic to render the error message in the Component JSX. For that, we need to define error state using useState similar to how we did with loading and page states.
    a. Make sure you display the error Typography conditionally only when the error message is set in the state.

Testing Error Handling

As we're using a 3rd party API, it's hard to introduce arbitrary errors like permission or server errors, however, we can look in the API documentation for ways to force the API to return an error state. A closer look at the errors documentation for that API tells you what kinds of errors the API can run into. Most of them are controlled through making changes to the API URL in your code.
A straightforward way to test the error is by removing your API key from the URL or making random changes to its value so it becomes invalid. This immediately triggers either `apiKeyMissing` or `apiKeyInvalid` error codes.

I tested my solution with changing the API Key value to get apiKeyInvalid error and with my custom throw error message, the UI now looks like this



Error: apiKeyInvalid, Details: Your API key is invalid or incorrect. Check your key, or go to https://newsapi.org to create a free API key.

[Bonus] Improve Loading State with Skeleton

A common UX improvement to quality web application is to implement a skeleton effect to loading state (example from Material UI https://mui.com/material-ui/react-skeleton/). Skeleton gives users a better sense of app responsiveness, it gives them some UI while data is loading. Skeleton loads much faster than the real data because it's pure UI element with no data requests. This is similar to having a spinning wheel to indicate that data is loading, but is better as it takes a shape close to the shape of the actual results that would render. Major applications like Youtube, Facebook News Feed or LinkedIn news feed adopt this behavior. In this section, you will build your own version of loading skeleton using Material UI Skeleton.

Loading state is handled in NewsFeed component. In this component we can import Skeleton from Material UI and add Skeleton JSX elements to look similar to the original results cards.

For better code organization, we can create a separate Component similar to News Article and call it LoadingArticle

`LoadingArticle` has a similar structure to `NewsArticle` JSX except for not including a placeholder for image. This is because images are optional and a Skeleton doesn't need to look very big if the actual data won't include images. This is an example of how it could look like

```
return (
<StyledCard>
  <CardActionArea>
    <CardContent>
      <Skeleton variant="text" sx={{ fontSize: "5rem" }} />
      <Skeleton variant="text" sx={{ fontSize: "1.5rem" }} />
    </CardContent>
  </CardActionArea>
  <Box p={2}>
    <Skeleton variant="text" width={200} sx={{ fontSize: "1.5rem" }} />
    <Skeleton variant="text" width={200} sx={{ fontSize: "1.5rem" }} />
  </Box>
</StyledCard>
);
```

LoadingAritcle uses the same StyledCard we created in NewsArticle component. To avoid copying the code between both components, we can refactor StyledCard into a separate component with Children. Try to refactor StyledCard into a reusable component in both places.

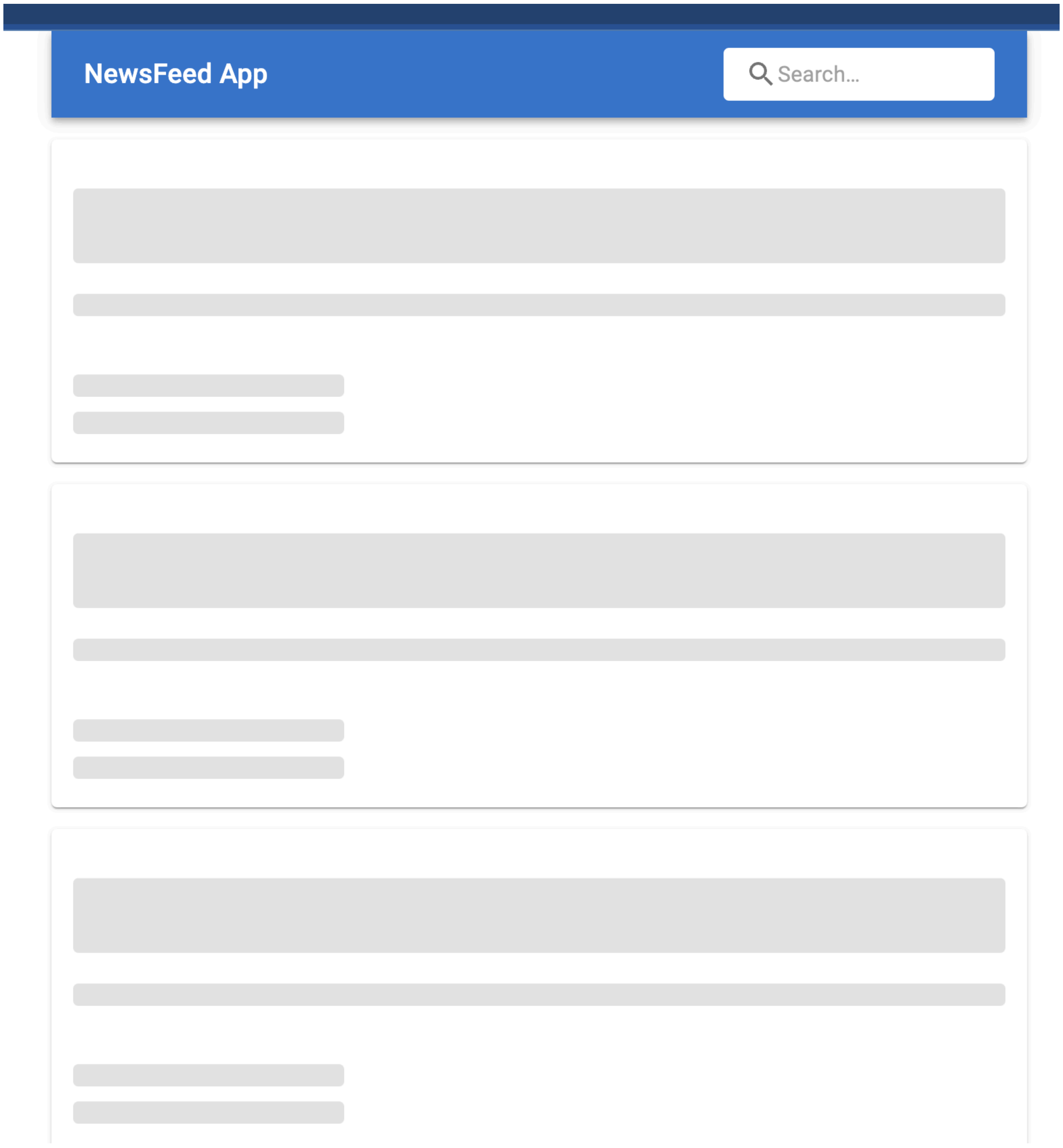
Next, we need to conditionally load 5 LoadingArticle components inside NewsFeed component when the loading state is true. Try to change the logic inside this component to make it work.

You can hardcode the number 5 to create an Array of 5 elements when loading prop is true, or you can take that number from App.jsx and pass it down to NewsFeed component as App component has the expected result page size.

The array of LoadingArticles would eventually look like this

```
{loading && [...Array(pageSize)].map(() => <LoadingArticle />)}
```

The UI will look this when articles are loading



IMPORTANT NOTE

Because we’re using debounce to load data even on initial page load. The initial load takes about 500 milliseconds before you can see any data. We need to change loading state initial value to true instead of false like this

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

Otherwise, loading will only be set after 500 milliseconds and it will show initially as if “No articles where found”.

Add Category Filter

NewsAPI allows adding a category name to the search URL. Possible category options are **business**, **entertainment**, **general**, **health**, **science**, **sports**, and **technology**. Your goal is to add a category dropdown menu in the News Feed header. If no category is selected, the search query ignores the category. If a category is selected, it should be used to reload data.

First, we need to define Category dropdown in the search header like so

```
<StyledSelect>
  <StyledMenuItem value="general">General</StyledMenuItem>
  <StyledMenuItem value="business">Business</StyledMenuItem>
  <StyledMenuItem value="entertainment">Entertainment</StyledMenuItem>
  <StyledMenuItem value="health">Health</StyledMenuItem>
  <StyledMenuItem value="science">Science</StyledMenuItem>
  <StyledMenuItem value="sports">Sports</StyledMenuItem>
  <StyledMenuItem value="technology">Technology</StyledMenuItem>
</StyledSelect>
```

StyledSelect and StyledMenuItem have similar styling to the search input to make them look similar. You can copy the following styling directly into your code as needed

```
const StyledSelect = styled(Select)(({ theme }) => ({
  color: theme.palette.action,
  backgroundColor: theme.palette.common.white,
  "&:before": {
    borderColor: theme.palette.action,
  },
  "&:after": {
    borderColor: theme.palette.action,
  },
  "& .MuiSelect-icon": {
    color: theme.palette.action,
  },
  margin: theme.spacing(2),
  width: 200,
  height: 40,
}));

const StyledMenuItem = styled(MenuItem)(({ theme }) => ({
  color: theme.palette.text.primary,
}));
```

Next, we need to add a category state in App component to be used by the fetch query URL. Adjust the state and query to make it accept the category. Default value for the category state is "**general**".

Next, we need to listen to category changes by the dropdown. We'll need to pass category and setCategory from App component to NewsHeader component similar to how we do for the search text.

IMPORTANT NOTE:

Because we need the default category to be "**general**" in the App state, we need to pass down the category to the **Select component** "**value**" prop to have the correct selection displayed. Don't forget to reset Page number to "1" when category changes. You also need to pass "**category**" to loadData function similar to how you do with "**query**".

Select Component should look like this

```
<StyledSelect value={category} onChange={onCategoryChange}>
  ...
</StyledSelect>
```

Category change handle would look like this

```
const handleCategoryChange = (event) => {
  setCategory(event.target.value);
  setPage(1);
};
```

And The call to NewsHeader would look like this

```
<NewsHeader
  category={category}
  onCategoryChange={handleCategoryChange}
  onSearchChange={handleSearchChange}
/>
```

Add URL to each card to open the news source in a new page

Main part of the news feed experience is to be open to click on news post to see the details of the corresponding news. NewsAPI returns a "url" as part of its response. Your task here is to read "url" from the response and pass it down to NewsArticle to render the card as a URL (Anchor link). Clicking anywhere in the card (or possibly the top section containing the Image, Title & description) should open the news URL in a new tab

Important Note

You can wrap the Card (or the top section) with <a> tag with target=”__blank” to open it in a new tab. You need to set the href value to the URL and change the tag style to keep the current style of the text in the card.