Paginated Fetch

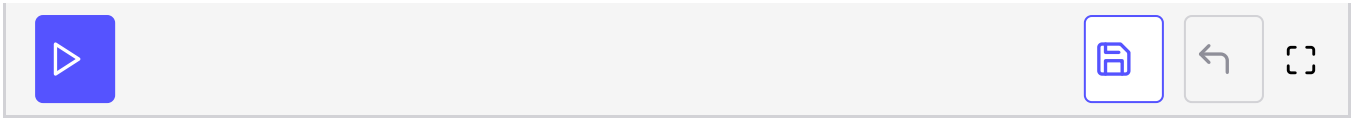
Let's extend the composable API constants so it can deal with paginated data!

Take a closer look at the data structure and observe how the Hacker News API returns more than a list of hits. Precisely, it returns a paginated list. The page property, which is `0` in the first response, can be used to fetch more paginated sublists as results. You only need to pass the next page with the same search term to the API.

We'll start with:

```
const DEFAULT_QUERY = 'redux';

const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
const PARAM_PAGE = 'page=';
```



Now you can use the new constant to add the page parameter to your API request:

```
const DEFAULT_QUERY = 'redux';

const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
const PARAM_PAGE = 'page=';

const url = `${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${DEFAULT_QUERY}&${PARAM_PAGE}`;

console.log(url);
// output: https://hn.algolia.com/api/v1/search?query=redux&page=
```

The `fetchSearchTopStories()` method will take the page as second argument. If you don't provide the second argument, it will fallback to the `0` page for the initial request. Thus the `componentDidMount()` and `onSearchSubmit()` methods

fetch the first page on the first request. Every additional fetch should fetch the next page by providing the second argument.

```
class App extends Component {

  ...

  fetchSearchTopStories(searchTerm, page = 0) {
    fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${page}`)
      .then(response => response.json())
      .then(result => this.setSearchTopStories(result))
      .catch(error => error);
  }

  ...

}
```

The page argument uses the JavaScript ES6 default parameter to introduce the fallback to page `0` in case no defined page argument is provided for the function.

Now you can use the current page from the API response in `fetchSearchTopStories()`. You can use this method in a button to fetch more stories with an `onClick` button handler. Let's use the Button to fetch more paginated data from the Hacker News API. For this, we'll define the `onClick()` handler, which takes the current search term and the next page (current page + 1).

```
class App extends Component {

  ...

  render() {
    const { searchTerm, result } = this.state;
    const page = (result && result.page) || 0;
    return (
      <div className="page">
        <div className="interactions">
        ...
        { result &&
          <Table
            list={result.hits}
            onDismiss={this.onDismiss}
          />
        }
        <div className="interactions">
          <Button onClick={() => this.fetchSearchTopStories(searchTerm, page + 1)}>
```

```
            More
          </Button>
        </div>

      </div>
    );
  }
}
```

In your `render()` method, make sure to default to page 0 when there is no result. Remember, the `render()` method is called before the data is fetched asynchronously in the `componentDidMount()` lifecycle method.

There is still one step missing, because fetching the next page of data will override your previous page of data. We want to concatenate the old and new list of hits from the local state and new result object, so we'll adjust its functionality to add new data rather than override it.

```
setSearchTopStories(result) {
  const { hits, page } = result;

  const oldHits = page !== 0
    ? this.state.result.hits
    : [];

  const updatedHits = [
    ...oldHits,
    ...hits
  ];

  this.setState({
    result: { hits: updatedHits, page }
  });
}
```

A couple things happen in the `setSearchTopStories()` method now. First, you get the hits and page from the result.

Second, you have to check if there are already old hits. When the page is 0, it is a new search request from `componentDidMount()` or `onSearchSubmit()`. The hits are empty. But when you click the "More" button to fetch paginated data the page isn't 0. The old hits are already stored in your state and thus can be used.

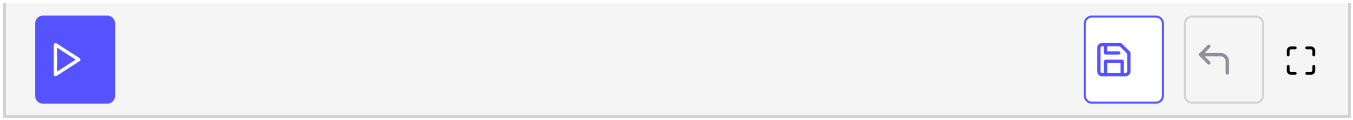
Third, you don't want to override the old hits. You can merge old and new hits from the recent API request, which can be done with a JavaScript ES6 array spread operator.

Fourth, you set the merged hits and page in the local component state.

Now we'll make one last adjustment. When you try the "More" button it only fetches a few list items. The API URL can be extended to fetch more list items with each request, so we add even more composable path constants:

```
const DEFAULT_QUERY = 'redux';
const DEFAULT_HPP = '100';

const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
const PARAM_PAGE = 'page=';
const PARAM_HPP = 'hitsPerPage=';
```



Now you can use the constants to extend the API URL.

```
fetchSearchTopStories(searchTerm, page = 0) {
  fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${page}&${PARAM
    .then(response => response.json())
    .then(result => this.setSearchTopStories(result))
    .catch(error => error);
}
```

The request to the Hacker News API fetches more list items in one request than before. As you can see, a powerful API such as the Hacker News API gives plenty of ways to experiment with real world data. You should make use of it to make your endeavours when learning something new more exciting. That's how I learned about the empowerment that APIs provide when learning a new programming language or library.

Here's the code so far:

```
import React, { Component } from 'react';
require('./App.css');

const DEFAULT_QUERY = 'redux';
const DEFAULT_HPP = '100';

const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
const PARAM_PAGE = 'page=';
```

```
class App extends Component {

  constructor(props) {
    super(props);

    this.state = {
      result: null,
      searchTerm: DEFAULT_QUERY,
    };

    this.setSearchTopstories = this.setSearchTopstories.bind(this);
    this.fetchSearchTopstories = this.fetchSearchTopstories.bind(this);
    this.onSearchChange = this.onSearchChange.bind(this);
    this.onSearchSubmit = this.onSearchSubmit.bind(this);
    this.onDismiss = this.onDismiss.bind(this);
  }

  setSearchTopstories(result) {
    const { hits, page } = result;

    const oldHits = page !== 0
      ? this.state.result.hits
      : [];

    const updatedHits = [
      ...oldHits,
      ...hits
    ];

    this.setState({
      result: { hits: updatedHits, page }
    });
  }

  fetchSearchTopstories(searchTerm, page = 0) {
    fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${page}`)
      .then(response => response.json())
      .then(result => this.setSearchTopstories(result))
        .catch(e => e);
  }

  componentDidMount() {
    const { searchTerm } = this.state;
    this.fetchSearchTopstories(searchTerm);
  }

  onSearchChange(event) {
    this.setState({ searchTerm: event.target.value });
  }

  onSearchSubmit(event) {
    const { searchTerm } = this.state;
    this.fetchSearchTopstories(searchTerm);
    event.preventDefault();
  }

  onDismiss(id) {
    const isNotId = item => item.objectID !== id;
    const updatedHits = this.state.result.hits.filter(isNotId);
    this.setState({
      result: { ...this.state.result, hits: updatedHits }
    });
```

```
  }

  render() {
    const { searchTerm, result } = this.state;
    const page = (result && result.page) || 0;
    return (
      <div className="page">
        <div className="interactions">
          <Search
            value={searchTerm}
            onChange={this.onSearchChange}
            onSubmit={this.onSearchSubmit}
          >
            Search
          </Search>
        </div>
        { result &&
          <Table
            list={result.hits}
            onDismiss={this.onDismiss}
          />
        }
        <div className="interactions">
          <Button onClick={() => this.fetchSearchTopstories(searchTerm, page + 1)}>
            More
          </Button>
        </div>
      </div>
    );
  }
}


const Search = ({
  value,
  onChange,
  onSubmit,
  children
}) =>
  <form onSubmit={onSubmit}>
    <input
      type="text"
      value={value}
      onChange={onChange}
    />
    <button type="submit">
      {children}
    </button>
  </form>

const Table = ({ list, onDismiss }) =>
  <div className="table">
    { list.map(item =>
      <div key={item.objectID} className="table-row">
        <span style={{ width: '40%' }}>
          <a href={item.url}>{item.title}</a>
        </span>
        <span style={{ width: '30%' }}>
          {item.author}
        </span>
        <span style={{ width: '10%' }}>
          {item.num_comments}
```

```
        </span>
        <span style={{ width: '10%' }}>
          {item.points}
        </span>
        <span style={{ width: '10%' }}>
          <Button
            onClick={() => onDismiss(item.objectID)}
            className="button-inline"
          >
            Dismiss
          </Button>
        </span>
      </div>
    )}
  </div>

const Button = ({ onClick, className = '', children }) =>
  <button
    onClick={onClick}
    className={className}
    type="button"
  >
    {children}
  </button>

export default App;
```

Exercise:

- Experiment with the Hacker News API parameters

Further Reading:

- Read about ES6 default parameters