JUNE 14, 2021 / #REACT

How to Build a Custom Pagination Component in React

Shubham Khatri

We often work with web applications that need to fetch large amounts of data from a server through APIs and render it on the screen.

For example, in a **Social media application** we fetch and render users' posts and comments. In an **HR dashboard** we display information about candidates who applied for a job. And in an **Email Client** we show the a user's emails.

your webpage to slow down considerably because of the large number of DOM elements present in the webpage.

If we want to optimize on performance we can adopt various techniques to render data in a more efficient manner. Some of these methods include **infinite scroll with virtualization** and **pagination**.

Pagination works well when you know the size of the data in advance, and you don't make frequent additions or deletions to the data-set.

For instance in a social media website where new posts are published every few milliseconds, pagination wouldn't be an ideal solution. But it would work well for an HR dashboard where candidate applications are displayed and need to be filtered or sorted as well.

In this post, we will focus on pagination and we'll build a custom controlled component that handles page

range of numbers to be rendered by the pagination component. We can use this hook independently as well when we want to render a pagination component with different styles or in a different design.

Below is a demo of what we will be building in this tutorial:

ID	FIRST NAME	LAST NAME	EMAIL	PHONE
1	Jessamyn	Espinazo	jespinazo0@chicagotribune.com	162-166-0977
2	Isac	Tooher	itooher1@psu.edu	655-567-3619
3	Tabbatha	Proschke	tproschke2@weibo.com	327-612-4850
4	Ninetta	Mabb	nmabb3@canalblog.com	971-296-0911
5	Danni	Wallentin	dwallentin4@comcast.net	983-297-0506
6	Neely	Purkins	npurkins5@mediafire.com	379-119-4237
7	Jessika	Kinkaid	jkinkaid6@eventbrite.com	771-888-6284
8	Julianna	Swindall	jswindall7@aol.com	252-614-0486
9	Corrinne	Geeve	cgeeve8@wisc.edu	450-872-8646
10	Trumann	Flux	tflux9@census.gov	249-892-1585

‹ 1 2 3 4 5 … 100 ›

How to Set Up the Project

If you are familiar with setting up a React project, you can skip this section.

In order to set up our React Project, we will use the `create-react-app` command line package. You can install the package globally using `npm install -g create-react-app` or `yarn add global create-react-app`.

Run `create-react-app` from the command line to create a new project like this:

```
npx create-react-app react-pagination
```

Next we need to install our dependencies. We will just be using a simple additional dependency called `classnames` which provides flexibility when handling multiple classNames conditionally.

Now, we can run our project using the below command:

```
yarn start
```

How to Define the Interface

Now that we have our project running, we'll dive straight into our `Pagination` component.

Let's first look at what values we need as props to our `Pagination` component:

- **totalCount**: represents the total count of data available from the source.
- **currentPage**: represents the current active page. We'll use a **1-based index** instead of a

- **pageSize**: represents the maximum data that is visible in a single page.
- **onPageChange**: callback function invoked with the updated page value when the page is changed.
- **siblingCount** (optional): represents the min number of page buttons to be shown on each side of the current page button. Defaults to 1.

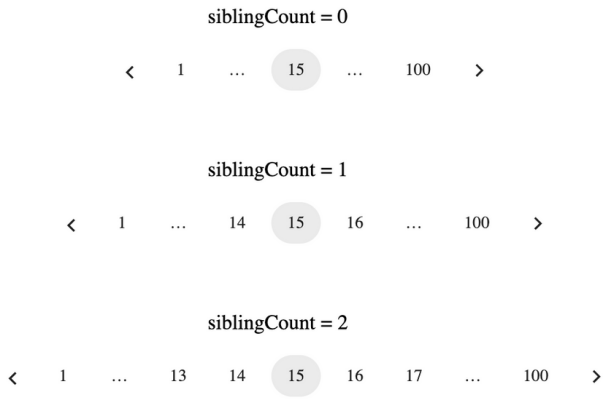


Illustration of different values siblingCount

From the pagination component we'll invoke the `usePagination` hook which will take in the following parameters to compute the page ranges: `totalCount`, `currentPage`, `pageSize`, `siblingCount`.

How to Implement the usePagination Hook

Below are the few things we need to keep in mind while implementing the `usePagination` hook:

- Our pagination hook must return the range of numbers to be displayed in our pagination component as an array.
- The computation logic needs to re-run when either `currentPage`, `pageSize`, `siblingCount`, or `totalCount` changes.
- The total number of items returned by the hook should remain constant. This will avoid

user is interacting with the component.

Keeping the above things in mind let's create a file called `usePagination.js` in our project `src` folder and start with the implementation.

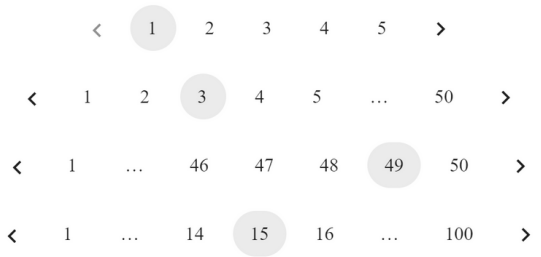
Our code skeleton will be as follows:

```
export const usePagination = ({
  totalCount,
  pageSize,
  siblingCount = 1,
  currentPage
}) => {
  const paginationRange = useMemo(() => {
     // Our implementation logic will go here

  }, [totalCount, pageSize, siblingCount, currentPa

  return paginationRange;
};
```

If we look at the above code, we are using the

Also we are setting the `defaultValue` of our `siblingCount` prop to be `1` as it is an optional prop.

Before we go ahead and implement the code logic, let's understand the different behaviors of the `Pagination` component. The below image contains the possible states of a pagination component:



Different states of a Pagination Component

Note that there are four possible states for a pagination component. We'll go over them one by one.

range from 1 to totalPageCount.

- Total page count is greater than the page pills but only the right DOTS are visible.
- Total page count is greater than the page pills but only the left DOTS are visible.
- Total page count is greater than the page pills and both the left and the right DOTS are visible.

As a first step, we shall go about calculating the total pages from `totalCount` and `pageSize` as follows:

```
const totalPageCount = Math.ceil(totalCount / pageS
```

Notice that we are using `Math.ceil` to round of the number to the next higher integer value. This ensures that we are reserving an extra page for the remaining

function which takes in a `start` and `end` value and returns an array with elements from start to end:

```
const range = (start, end) => {
  let length = end - start + 1;
  /*
    Create an array of certain length and set the e
    start value to end value.
  */
  return Array.from({ length }, (_, idx) => idx + s
};
```

Finally, we'll implement the core logic by keeping the above cases in mind.

```
export const usePagination = ({
  totalCount,

  pageSize,
  siblingCount = 1,
  currentPage
}) => {
```

```
const totalPageNumbers = siblingCount + 5;

/*
  Case 1:
  If the number of pages is less than the page
  paginationComponent, we return the range [1..
*/
if (totalPageNumbers >= totalPageCount) {
  return range(1, totalPageCount);
}

/*
    Calculate left and right sibling index and
*/
const leftSiblingIndex = Math.max(currentPage -
const rightSiblingIndex = Math.min(
  currentPage + siblingCount,
  totalPageCount
);

/*
  We do not show dots just when there is just o
*/
const shouldShowLeftDots = leftSiblingIndex > 2
const shouldShowRightDots = rightSiblingIndex <

const firstPageIndex = 1;
const lastPageIndex = totalPageCount;

/*
```

```
    let leftRange = range(1, leftItemCount);

    return [...leftRange, DOTS, totalPageCount];
  }

  /*
      Case 3: No right dots to show, but left dot
  */
  if (shouldShowLeftDots && !shouldShowRightDots)

    let rightItemCount = 3 + 2 * siblingCount;
    let rightRange = range(
      totalPageCount - rightItemCount + 1,
      totalPageCount
    );
    return [firstPageIndex, DOTS, ...rightRange];
  }

  /*
      Case 4: Both left and right dots to be show
  */
  if (shouldShowLeftDots && shouldShowRightDots)
    let middleRange = range(leftSiblingIndex, rig
    return [firstPageIndex, DOTS, ...middleRange,
  }
}, [totalCount, pageSize, siblingCount, currentPa

  return paginationRange;
};
```

The idea of the implementation is that we identify the range of numbers we want to show in our pagination component and then join them together with the separators or DOTS when we return the final range.

For the first scenario where our `totalPageCount` is less than the total number of pills we calculated based on the other params, we just return a range of numbers `1..totalPageCount`.

For the other scenarios, we go about identifying whether we need DOTS on the left or right side of the `currentPage` by calculating the left and right indices after including the sibling pills to the `currentPage` and then make our decisions.

Once we know where we want to show the DOTS, the rest of the calculations are quite straightforward.

How to Implement the

`usePagination` hook in our pagination component and we'll map over the returned range to render them.

We create a `Pagination.js` file in our `src` folder and implement the code logic as follows:

```
import React from 'react';
import classnames from 'classnames';
import { usePagination, DOTS } from './usePaginatio
import './pagination.scss';
const Pagination = props => {
  const {
    onPageChange,
    totalCount,
    siblingCount = 1,
    currentPage,
    pageSize,
    className
  } = props;

  const paginationRange = usePagination({

    currentPage,
    totalCount,
    siblingCount,
    pageSize
  });
```

```
  return null;
}

const onNext = () => {
  onPageChange(currentPage + 1);
};

const onPrevious = () => {
  onPageChange(currentPage - 1);
};

let lastPage = paginationRange[paginationRange.le
return (
  <ul
    className={classnames('pagination-container',
  >
     {/* Left navigation arrow */}
    <li
      className={classnames('pagination-item', {
        disabled: currentPage === 1
      })}
      onClick={onPrevious}
    >
      <div className="arrow left" />
    </li>
    {paginationRange.map(pageNumber => {

      // If the pageItem is a DOT, render the DOT
      if (pageNumber === DOTS) {
        return <li className="pagination-item dot
```

```
            <li
              className={classnames('pagination-item'
                selected: pageNumber === currentPage
              })}
              onClick={() => onPageChange(pageNumber)
            >
              {pageNumber}
            </li>
          );
        })}
        {/*  Right Navigation arrow */}
        <li
          className={classnames('pagination-item', {
            disabled: currentPage === lastPage
          })}
          onClick={onNext}
        >
          <div className="arrow right" />
        </li>
      </ul>
    );
};

export default Pagination;
```

Pagination Implementation

We render the `Pagination` component as a list with left and right arrows which handle the previous and next actions the user makes. In between the arrows, we map over the `paginationRange` and render the page numbers as `pagination-items`. If the page item is a DOT we render a unicode character for it.

As a special handling we add a `disabled` class to the left/right arrow if the `currentPage` is the first or the last page, respectively. We disable the `pointer-events` and update the styles of the arrow icons through CSS if the icon needs to be disabled.

We also add click event handlers to the page pills which will invoke the `onPageChanged` callback function with the updated value of `currentPage`.

Our CSS file will contain the following styles:

```
.pagination-item {
  padding: 0 12px;
  height: 32px;
  text-align: center;
  margin: auto 4px;
  color: rgba(0, 0, 0, 0.87);
  display: flex;
  box-sizing: border-box;
  align-items: center;
  letter-spacing: 0.01071em;
  border-radius: 16px;
  line-height: 1.43;
  font-size: 13px;
  min-width: 32px;

  &.dots:hover {
    background-color: transparent;
    cursor: default;
  }
  &:hover {
    background-color: rgba(0, 0, 0, 0.04);
    cursor: pointer;
  }

  &.selected {
    background-color: rgba(0, 0, 0, 0.08);
  }

  .arrow {
    &::before {
```

```
      display: inline-block;
      width: 0.4em;
      height: 0.4em;
      border-right: 0.12em solid rgba(0, 0, 0, 0.
      border-top: 0.12em solid rgba(0, 0, 0, 0.87
    }

    &.left {
      transform: rotate(-135deg) translate(-50%);
    }

    &.right {
      transform: rotate(45deg);
    }
  }

  &.disabled {
    pointer-events: none;

    .arrow::before {
      border-right: 0.12em solid rgba(0, 0, 0, 0.
      border-top: 0.12em solid rgba(0, 0, 0, 0.43
    }

    &:hover {
      background-color: transparent;
      cursor: default;
    }
  }
}
```

And that's it!

Our generic pagination implementation is ready and we can use it anywhere in our codebase.

How to Use the Custom Pagination Component

As a last step, let's incorporate this component in a small example.

For the scope of this article, we shall render static data in the form of a table. So let's go ahead and do that first:

```
import React from 'react';
import data from './data/mock-data.json';

export default function App() {
```

```
            <tneaa>
              <tr>
                <th>ID</th>
                <th>FIRST NAME</th>
                <th>LAST NAME</th>
                <th>EMAIL</th>
                <th>PHONE</th>
              </tr>
            </thead>
            <tbody>
              {data.map(item => {
                return (
                  <tr>
                    <td>{item.id}</td>
                    <td>{item.first_name}</td>
                    <td>{item.last_name}</td>
                    <td>{item.email}</td>
                    <td>{item.phone}</td>
                  </tr>
                );
              })}
            </tbody>
          </table>
        </>
      );
    }
```

At this point our UI looks as follows:

1	Jessamyn	Espinazo	jespinazo0@chicagotribune.com	162-166-0977
2	Isac	Tooher	itooher1@psu.edu	655-567-3619
3	Tabbatha	Proschke	tproschke2@weibo.com	327-612-4850
4	Ninetta	Mabb	nmabb3@canalblog.com	971-296-0911
5	Danni	Wallentin	dwallentin4@comcast.net	983-297-0506
6	Neely	Purkins	npurkins5@mediafire.com	379-119-4237
7	Jessika	Kinkaid	jkinkaid6@eventbrite.com	771-888-6284
8	Julianna	Swindall	jswindall7@aol.com	252-614-0486
9	Corrinne	Geeve	cgeeve8@wisc.edu	450-872-8646
10	Trumann	Flux	tflux9@census.gov	249-892-1585
11	Annalise	Keinrat	akeinrata@i2i.jp	659-283-4601
12	Cal	Haverson	chaversonb@multiply.com	689-567-9516
13	Erik	McGillivrie	emcgillivriec@theglobeandmail.com	334-579-0995
14	Cherilyn	Tuddenham	ctuddenhamd@indiegogo.com	408-721-4575

Now to incorporate the `Pagination` component, we need two things.

- First, we maintain a `currentPage` state.

- Second, we calculate the data to be rendered for a given page and just map and render it.

For the purposes of this demo, we'll keep the

Once we have made changes, we can go ahead and render our `Pagination` component with the appropriate props.

With these changes in mind, our final code will be as follows:

```
import React, { useState, useMemo } from 'react';
import Pagination from '../Pagination';
import data from './data/mock-data.json';
import './style.scss';

let PageSize = 10;

export default function App() {
  const [currentPage, setCurrentPage] = useState(1)

  const currentTableData = useMemo(() => {
    const firstPageIndex = (currentPage - 1) * Page
    const lastPageIndex = firstPageIndex + PageSize
    return data.slice(firstPageIndex, lastPageIndex
  }, [currentPage]);
```

```
          <tr>
            <th>ID</th>
            <th>FIRST NAME</th>
            <th>LAST NAME</th>
            <th>EMAIL</th>
            <th>PHONE</th>
          </tr>
        </thead>
        <tbody>
          {currentTableData.map(item => {
            return (
              <tr>
                <td>{item.id}</td>
                <td>{item.first_name}</td>
                <td>{item.last_name}</td>
                <td>{item.email}</td>
                <td>{item.phone}</td>
              </tr>
            );
          })}
        </tbody>
      </table>
      <Pagination
        className="pagination-bar"
        currentPage={currentPage}
        totalCount={data.length}
        pageSize={PageSize}
        onPageChange={page => setCurrentPage(page)}
      />
    </>
```

Final Demo code

Here is a live demo of this tutorial:

In this article, we create a custom React hook `usePagination` and used it within our `Pagination` component. We also implemented a short demo which used this component.

You can check out the full source code for this tutorial in this GitHub repository.

If you have any questions or suggestions regarding this article, please feel free to reach out to me on Twitter.

Thank you for reading.

Shubham Khatri

Sr Frontend Developer at Flock | Passionate about Javascript, React, and Web Development | Active Stackoverflow contributor

---

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

interactive coding lessons - all freely available to the public.

We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can make a tax-deductible donation here.

Trending Guides

Python map()

Python .pop()

Python arrays

npm Uninstall

Insertion Sort

Python If-Else

All Caps in CSS

Open Task Manager on Mac

parseInt() in JavaScript

Print statement in Python

Remove Directory in Linux

Python str.lower() Example

Second Monitor Not Detected

How to Declare Strings in C

How to Use .len() in Python

Python Convert String to Int

How to create a free website

Donate

Learn to code — free 3,000-hour curriculum

Sponsors Academic Honesty Code of Conduct

Privacy Policy Terms of Service Copyright Policy