React Clientside Pagination

Complete Documentation & Usage Guide

🚀 Production Ready ♿ Accessible 🎨 Fully Customizable

⚡ Zero Dependencies 📱 Responsive 🔧 TypeScript

📖 Table of Contents

🎯 Introduction

What is React Clientside Pagination?

A **flexible, robust, and production-ready** React pagination component that handles all your pagination needs with zero hassle. Whether you're building an e-commerce site, blog, dashboard, or any data-heavy application, this package makes pagination simple and powerful.

✅ Zero Dependencies

Lightweight and fast with only React as peer dependency

✅ Full Customization

Make it look exactly how you want with any design system

✅ Accessible by Default

WCAG compliant out of the box with ARIA labels

✅ TypeScript Ready

Full type definitions included for better development

✅ Two Usage Modes

Simple component or headless hook for maximum flexibility

✅ Responsive Design

Works perfectly on all screen sizes with smart ellipsis

📦 Installation

Note: Ensure you have React 16.8+ installed (peer dependency).

Step 1: Install the Package

```
# Using npm
npm install react-clientside-pagination
```

```
# Using yarn
yarn add react-clientside-pagination

# Using pnpm
pnpm add react-clientside-pagination
```

Step 2: That's It!

No additional dependencies needed. Just import and start using!

🚀 Quick Start

Your First Pagination in 2 Minutes

```
import React from 'react';
import { Pagination } from 'react-clientside-pagination';

// Your data (could be from API, JSON, anywhere)
const products = [
  { id: 1, name: 'Product 1', price: 29.99 },
  { id: 2, name: 'Product 2', price: 39.99 },
  // ... more products (25+ for pagination to show)
];

function ProductList() {
  return (
    <div>
      <Pagination
        data={products}
        itemsPerPage={5}
        renderItem={(product) => (
          <div key={product.id} className="product-card">
            <h3>{product.name}</h3>
            <p>${product.price}</p>
          </div>
        )}
        labels={{ prev: '← Prev', next: 'Next →' }}
      />
    </div>
  );
}
```

✅ **What just happened?**
• Your products are automatically paginated
• Navigation buttons are generated
• You control how each product looks
• It's fully accessible
• Responsive design works out of the box

🧠 Core Concepts

1. The Two Modes of Pagination

🎮 Controlled Mode (You Manage State)

```
const [currentPage, setCurrentPage] = useState(1);

<Pagination
  data={data}
  currentPage={currentPage}
  onPageChange={setCurrentPage}
  renderItem={/* ... */}
/>
```

Use when: You need to know/sync the current page elsewhere in your app.

Uncontrolled Mode (Component Manages State)

```
<Pagination
  data={data}
  initialPage={1}
  renderItem={/* ... */}
/>
```

Use when: You want simple pagination without extra state management.

2. The Magic of renderItem

Think of `renderItem` as your **"item blueprint"** - you tell the component how to display each item, and it handles the rest:

```
renderItem={(item) => <YourCustomComponent data={item} />}
```

> **Simple analogy:** It's like giving a factory instructions: "Here's how to build each car, you handle the assembly line."

3. Smart Pagination Display

The component automatically handles:
maxButtons (default: 5) - Controls how many page buttons show
Ellipsis logic - Shows "..." when too many pages for clean display
Responsive behavior - Adapts to different screen sizes

Basic Usage

Example 1: Simple Blog Posts

```
import React from 'react';
import { Pagination } from 'react-clientside-pagination';

const blogPosts = [
  { id: 1, title: 'Getting Started with React', excerpt: 'Learn React
basics...' },
  { id: 2, title: 'Advanced React Patterns', excerpt: 'Master complex
patterns...' },
  // ... more posts
];

function Blog() {
  return (
    <div className="blog-container">
      <h1>My Blog</h1>
      <Pagination
        data={blogPosts}
        itemsPerPage={3}
        renderItem={(post) => (
          <article key={post.id} className="blog-post">
            <h2>{post.title}</h2>
            <p>{post.excerpt}</p>
            <button>Read More</button>
          </article>
        )}
      />
    </div>
  );
}
```

⚡ Advanced Features

1. Using the Headless Hook

For maximum flexibility, use the `usePagination` hook:

```
import React from 'react';
import { usePagination } from 'react-clientside-pagination';

function CustomPaginationUI() {
  const pagination = usePagination({
    data: products,
    itemsPerPage: 6
  });

  return (
    <div>
      {/* Your custom layout */}
      <div className="products-grid">
        {pagination.data.map(product => (
          <ProductCard key={product.id} product={product} />
        ))}
      </div>

      {/* Your custom pagination controls */}
      <div className="custom-pagination">
        <button
          onClick={pagination.goToPrev}
          disabled={!pagination.hasPrevPage}
        >
          ← Previous
        </button>

        <span>Page {pagination.currentPage} of {pagination.totalPages}
</span>

        <button
          onClick={pagination.goToNext}
          disabled={!pagination.hasNextPage}
        >
          Next →
        </button>
      </div>
    </div>
  );
}
```

2. Handling Empty States

Show a custom message when no data is available:

```
<Pagination
  data={[]} // Empty array
  itemsPerPage={10}
  emptyPlaceholder={
    <div className="empty-state">
      <h3>No products found</h3>
      <p>Try adjusting your search filters</p>
    </div>
  }
  renderItem={/* ... */}
/>
```

🔧 API Reference

Pagination Component Props

Prop	Type	Default	Description
`data`	`array`	`[]`	**Required:** Your data array
`itemsPerPage`	`number`	`10`	Items to show per page

Prop	Type	Default	Description
`currentPage`	`number`	`-`	Controlled mode current page
`onPageChange`	`function`	`-`	Called when page changes
`initialPage`	`number`	`1`	Initial page for uncontrolled mode
`renderItem`	`function`	`-`	**Required:** Renders each item
`keyExtractor`	`function`	`(item, index) => item.id || index`	Gets unique keys for React
`maxButtons`	`number`	`5`	Max page buttons to show
`showNumbers`	`boolean`	`true`	Show page number buttons
`showPrevNext`	`boolean`	`true`	Show previous/next buttons
`showFirstLast`	`boolean`	`false`	Show first/last page buttons
`disabled`	`boolean`	`false`	Disable all interactions

Prop	Type	Default	Description
`labels`	`object`	`{prev: 'Previous', next: 'Next'}`	Button labels for i18n
`emptyPlaceholder`	`ReactNode`	`null`	Shown when data is empty
`ariaLabel`	`string`	`'Pagination'`	Accessibility label for screen readers

Styling Props

Prop	Type	Description
`className`	`string`	Pagination container class
`containerClassName`	`string`	Outer container class
`buttonClassName`	`string`	All button classes
`activeClassName`	`string`	Active page button class
`disabledClassName`	`string`	Disabled button class

usePagination Hook

```
const pagination = usePagination({
  data: array,
  itemsPerPage: number,
  initialPage: number,
  currentPage: number,
  onPageChange: function
});
```

Returns object with:

`data` - Current page items

`currentPage` - Current page number

`totalPages` - Total number of pages

`totalItems` - Total items count

`hasNextPage` - If next page exists

`hasPrevPage` - If previous page exists

`goToPage(page)` - Go to specific page

`goToNext()` - Go to next page

`goToPrev()` - Go to previous page

`goToFirst()` - Go to first page

`goToLast()` - Go to last page

`setItemsPerPage(number)` - Change items per page

Recipes & Examples

Recipe 1: E-commerce Product Grid

```
function ProductGrid() {
  const [products] = useState([
    { id: 1, name: 'Laptop', price: 999, image: '/laptop.jpg' },
    // ... more products
  ]);

  return (
    <Pagination
      data={products}
```

```
        itemsPerPage={12}
        className="product-pagination"
        renderItem={(product) => (
          <div key={product.id} className="product-card">
            <img src={product.image} alt={product.name} />
            <h3>{product.name}</h3>
            <p className="price">${product.price}</p>
            <button className="add-to-cart">Add to Cart</button>
          </div>
        )}
        emptyPlaceholder={
          <div className="empty-state">
            <h3>No products found</h3>
            <p>Try adjusting your filters</p>
          </div>
        }
      />
    );
  }
```

🛠 Troubleshooting

Common Issues & Solutions

Problem: Warning about both initialPage and currentPage

```
// ❌ Don't do this:
<Pagination data={data} initialPage={1} currentPage={page} />

// ✅ Do this instead:
<Pagination data={data} currentPage={page} onPageChange={setPage} />
```

```
// OR
<Pagination data={data} initialPage={1} />
```

Problem: Pagination buttons not showing

```
// ❌ Not enough items for pagination
<Pagination data={[item1, item2]} itemsPerPage={10} />

// ✅ Ensure total items > itemsPerPage
<Pagination data={[item1, item2, ...item15]} itemsPerPage={5} />
```

Problem: Custom styles not applying

```
// ✅ Use the correct class props:
<Pagination
  data={data}
  className="pagination-container" // Pagination controls
  containerClassName="outer-wrapper" // Outer container
  buttonClassName="page-button" // All buttons
  activeClassName="active-page" // Current page button
/>
```

❓ Frequently Asked Questions

❓ Can I use this with TypeScript?

A: Yes! Full TypeScript support included. All types are automatically available and provide full IntelliSense in your editor.

❓ How do I change the number of visible page buttons?

A: Use the `maxButtons` prop:

```
<Pagination data={data} maxButtons={7} /> // Shows 7 page buttons
```

❓ Can I hide page numbers and only show prev/next?

A: Yes!

```
<Pagination
  data={data}
  showNumbers={false}
  showPrevNext={true}
/>
```

❓ How do I style using CSS modules or styled-components?

A: Use the className props or components prop:

```
// CSS Modules:
<Pagination
  data={data}
  className={styles.pagination}
  buttonClassName={styles.button}
/>

// styled-components:
<Pagination
  data={data}
```

```
    components={{ Button: StyledButton }}
  />
```

❓ Is it accessible?

A: Absolutely! Includes ARIA labels, keyboard navigation, screen reader support, proper focus management, and follows WCAG guidelines out of the box.

🎉 Congratulations!

You've now mastered React Clientside Pagination! Whether you're building a simple blog or a complex enterprise application, you have all the tools you need for perfect pagination.

Next Steps:

- Try the examples in your project
- Experiment with customization using the styling props
- Check the TypeScript types for autocomplete in your editor
- Join the community - share your creations and get help

Happy coding! 🚀