React Paginated User List Documentation

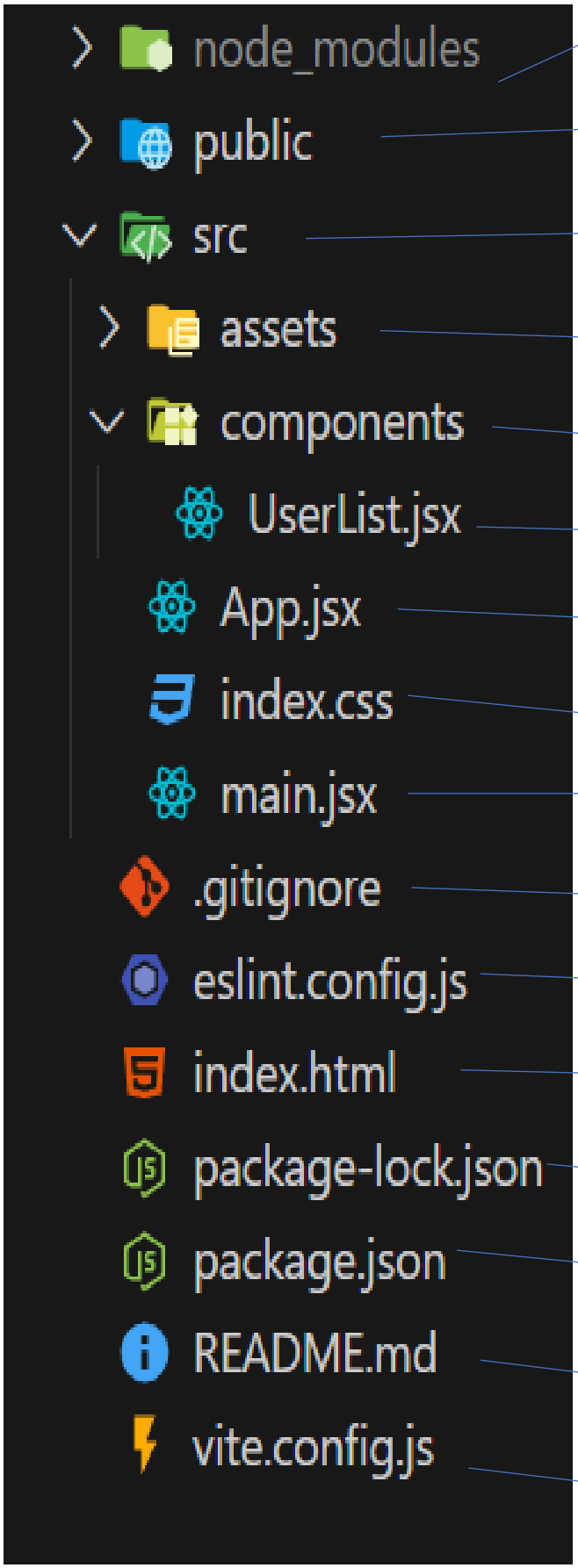
Project Overview

This React project fetches user data from an API and displays a paginated list of users. The user can navigate through different pages using "Previous" and "Next" buttons.

Key Features

- Fetches user data from **https://dummyjson.com/users**.
- Displays **3 users per page**.
- Provides **pagination controls** to navigate through pages.
- Ensures:
    - "Previous" button is **disabled** on the first page.
    - "Next" button is **disabled** on the last page.
- Shows user details:
    - Name
    - Email
    - Phone Number
    - Profile Picture

Project Foder Structure



Contains all the npm packages and dependencies installed for the project.

Stores static assets like images, fonts, and HTML files that are publicly accessible.

The source code directory containing the main application code.

Typically contains static assets like images, styles, and other resources used within the application.

Contains reusable React components.

A React component file that displays a list of users.

The main React component that serves as the entry point for the application.

A CSS file for styling the main application.

The main JavaScript file that renders the React application into the DOM.

Specifies files and directories to be ignored by Git.

Configuration file for ESLint, a tool for identifying and fixing problems in JavaScript code.

The main HTML file that serves as the entry point for the web application.

Automatically generated file that locks the versions of dependencies.

Contains metadata about the project and its dependencies.

A markdown file providing an overview of the project, including setup instructions and usage.

Configuration file for Vite, a build tool for modern web projects.

Project Implementation

1. Fetching User Data

- The useEffect hook is used to fetch user data when the component loads.
- Data is stored in the users state using useState.

2. Pagination Logic

- The total list of users is divided into pages using slice().
- The currentPage state keeps track of the active page.
- Users can move between pages using:
  - nextPage() → Moves to the next page.
  - prevPage() → Moves to the previous page.
- The "Next" button is **disabled** when reaching the last page.
- The "Previous" button is **disabled** on the first page.

3. Displaying Users

- The map() function is used to loop through the current page's users and display their details in cards.

Code Breakdown

1. State Variables

```
const [users, setUsers] = useState([]);

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

const usersPerPage = 3;
```

- users: Stores fetched user data.
- currentPage: Tracks the current page number.
- usersPerPage: Defines how many users appear per page.

2. Fetching Users (useEffect)

```
useEffect(() => {
  const fetchUsers = async () => {
    try {
      const response = await fetch('https://dummyjson.com/users');
      const data = await response.json();
      setUsers(data.users);
    } catch (error) {
      console.error('Error fetching users:', error);
    }
  };

  fetchUsers();
}, []);
```

- Fetches users **once** when the component mounts.
- Handles errors gracefully.

3. Pagination Logic

```
const indexOfLastUser = currentPage * usersPerPage;
const indexOfFirstUser = indexOfLastUser - usersPerPage;
const currentUsers = users.slice(indexOfFirstUser, indexOfLastUser);
```

- Determines which users should be displayed on the current page.

4. Page Navigation

```
const nextPage = () => {
  if (currentPage < Math.ceil(users.length / usersPerPage)) {
    setCurrentPage(currentPage + 1);
  }
};
```

```
const prevPage = () => {
  if (currentPage > 1) {
    setCurrentPage(currentPage - 1);
  }
};
```

- Controls pagination while preventing invalid page navigation.

5. UI Components

```
return (
  <div>
    <h1 className='user-list-heading'>User List</h1>
    <div className="user-list">
      {currentUsers.map((user) => (
        <div className="user-card" key={user.id}>
          <img src={user.image} alt={`${user.firstName} ${user.lastName}`} />
          <h2>{user.firstName} {user.lastName}</h2>
          <strong>Email:</strong> <p>{user.email}</p>
          <strong>Phone:</strong> <p>{user.phone}</p>
        </div>
      ))}
    </div>

    {/* Pagination Controls */}
    <div className="pagination">
      <button onClick={prevPage} disabled={currentPage ===
1}>Previous</button>
```

```
    <button onClick={nextPage} disabled={currentPage ===
Math.ceil(users.length / usersPerPage)}>Next</button>
   </div>

   {/* Page Indicator */}
   <div className="pagecount-container">
    <div className="pagecount">
     <span>Page {currentPage}</span>
    </div>
   </div>
  </div>
);
```

- Displays users dynamically.
- Provides pagination buttons.
- Shows current page number.

Styling (CSS)

1. Global Styles (Good for Overall Look)

```
body {
  font-family: Arial, sans-serif;
  background: linear-gradient(to right, #1abc9c, #3498db);
  padding: 20px;
}
```

2. User Card Styling (Core UI Component)

```
.user-card {
  background: #2c3e50;
```

```
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  padding: 20px;
  width: 300px;
  text-align: center;
  color: #ecf0f1;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.user-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 6px 12px rgba(0, 0, 0, 0.15);
}

.user-card img {
  width: 100px;
  height: 100px;
  border-radius: 50%;
  margin-bottom: 15px;
  border: 3px solid #ecf0f1;
}
```

3. Pagination Buttons (Navigation Controls)

```
.pagination {
  display: flex;
  justify-content: center;
  margin-top: 20px;
```

```
}

.pagination button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  background-color: #1abc9c;
  color: white;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

.pagination button:disabled {
  background-color: #7f8c8d;
  cursor: not-allowed;
}
.pagination button:hover:not(:disabled) {
  background-color: #16a085;
}
```

4. Page Counter Styling (Current Page Display)

```
.pagecount {
  padding: 10px 20px;
  background: #2c3e50;
  color: white;
  font-size: 18px;
  font-weight: bold;
```

```
  border-radius: 10px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  transition: transform 0.2s ease-in-out;
}

.pagecount:hover {
  transform: scale(1.1);
}
```

Conclusion

This React project efficiently implements **pagination, API fetching, and user display** in a simple yet effective way. The useState and useEffect hooks ensure smooth state management and data fetching.