Here are five common use cases for using React with Redux, along with code examples demonstrating the advantages:

1. Centralized State Management
Redux provides a single source of truth for managing the state of your application. This makes it easier to manage and debug the state, especially in large applications.

Example:

```javascript
// Define actions
const setUser = (user) => {
  return { type: 'SET_USER', payload: user };
};

// Define reducer
const userReducer = (state = null, action) => {
  switch(action.type) {
    case 'SET_USER':
      return action.payload;
    default:
      return state;
  }
};

// Create Redux store
import { createStore } from 'redux';
const store = createStore(userReducer);

// React component
import { connect } from 'react-redux';

const UserProfile = ({ user }) => {
  return (
    <div>
      <p>Name: {user.name}</p>
      <p>Email: {user.email}</p>
    </div>
  );
};

// Map state to props
const mapStateToProps = (state) => {
  return {
```

```
    user: state.user
  };
};

// Connect component to Redux store
const ConnectedUserProfile = connect(mapStateToProps)(UserProfile);

export default ConnectedUserProfile;
```

2. Improved Performance with Memoization
React-Redux provides the `connect()` function which optimizes the performance of components by preventing unnecessary re-renders through memoization techniques.

Example:

```javascript
import React, { memo } from 'react';
import { connect } from 'react-redux';

const TodoList = memo(({ todos }) => {
  return (
    <ul>
      {todos.map(todo => (
        <li key={todo.id}>{todo.text}</li>
      ))}
    </ul>
  );
});

const mapStateToProps = (state) => {
  return {
    todos: state.todos
  };
};

export default connect(mapStateToProps)(TodoList);
```

3. Time-Travel Debugging
Redux DevTools Extension allows you to visualize and inspect the state changes in your application over time, enabling powerful debugging capabilities, including time-travel debugging.

Example:

```javascript
import { createStore } from 'redux';
import rootReducer from './reducers';

const store = createStore(
  rootReducer,
  window.__REDUX_DEVTOOLS_EXTENSION__ &&
window.__REDUX_DEVTOOLS_EXTENSION__()
);
```

4. Middleware Integration
Redux middleware allows you to extend Redux with custom functionality, such as logging, asynchronous actions, and more, without modifying the core Redux library.

Example:
```javascript
import { applyMiddleware, createStore } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const store = createStore(
  rootReducer,
  applyMiddleware(thunk)
);
```

5. Server-side Rendering (SSR)
Redux can be easily integrated into server-side rendering setups, allowing you to pre-render your React components on the server with the correct initial state before sending them to the client.

Example:
```javascript
// Server-side code
import { renderToString } from 'react-dom/server';
import { Provider } from 'react-redux';
import store from './store';
import App from './App';

const html = renderToString(
  <Provider store={store}>
    <App />
  </Provider>
```

```
);

// Client-side code
import { hydrate } from 'react-dom';
import { Provider } from 'react-redux';
import store from './store';
import App from './App';

hydrate(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

These examples illustrate various advantages of using React with Redux, including centralized state management, performance optimization, debugging capabilities, middleware integration, and support for server-side rendering.

Certainly! Here's a table summarizing the advantages of using React with Redux:

Advantage	Description
Centralized State Management	Redux provides a single source of truth for managing the state of your application, making it easier to manage and debug the state.
Improved Performance	React-Redux optimizes performance by preventing unnecessary re-renders through memoization techniques.
Time-Travel Debugging	Redux DevTools Extension allows for visualization and inspection of state changes, enabling powerful time-travel debugging.
Middleware Integration	Redux middleware allows for extending Redux with custom functionality, such as logging and asynchronous actions.
Server-side Rendering (SSR) Support	Redux can be easily integrated into server-side rendering setups, allowing for pre-rendering of React components with initial state.

These advantages collectively contribute to better state management, performance optimization, debugging capabilities, middleware integration, and support for server-side rendering in React applications utilizing Redux.