Recap

This chapter focused on implementing advanced React components. You will learnt how to implement higher-order components, and we dove into more advanced topics in React.

You have learned advanced component techniques in React! Let's recap the chapter:

- **React**
  - The `ref` attribute to reference DOM elements
  - Higher-order components are a common way to build advanced components
  - Implementation of advanced interactions in React
  - Conditional classNames with a neat helper library
- **ES6**
  - Rest destructuring to split up objects and arrays

You can always find the source code in the official repository.

```
import React, { Component } from 'react';
import { sortBy } from 'lodash';
import classNames from 'classnames';
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=';

const SORTS = {
  NONE: list => list,
  TITLE: list => sortBy(list, 'title'),
  AUTHOR: list => sortBy(list, 'author'),
  COMMENTS: list => sortBy(list, 'num_comments').reverse(),
  POINTS: list => sortBy(list, 'points').reverse(),
};

class App extends Component {
```

```
constructor(props) {
  super(props);


  this.state = {
    results: null,
    searchKey: '',
    searchTerm: DEFAULT_QUERY,
    error: null,
    isLoading: false,
    sortKey: 'NONE',
    isSortReverse: false,
  };

  this.needsToSearchTopstories = this.needsToSearchTopstories.bind(this);
  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);
  this.onSort = this.onSort.bind(this);
}

onSort(sortKey) {
  const isSortReverse = this.state.sortKey === sortKey && !this.state.isSortReverse;
  this.setState({ sortKey, isSortReverse });
}

needsToSearchTopstories(searchTerm) {
  return !this.state.results[searchTerm];
}

setSearchTopstories(result) {
  const { hits, page } = result;
  const { searchKey, results } = this.state;

  const oldHits = results && results[searchKey]
    ? results[searchKey].hits
    : [];

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

  this.setState({
    results: {
      ...results,
      [searchKey]: { hits: updatedHits, page }
    },
    isLoading: false
  });
}

fetchSearchTopstories(searchTerm, page = 0) {
  this.setState({ isLoading: true });

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

```
componentDidMount() {
  const { searchTerm } = this.state;
  this.setState({ searchKey: searchTerm });
  this.fetchSearchTopstories(searchTerm);
}

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

onSearchSubmit(event) {
  const { searchTerm } = this.state;
  this.setState({ searchKey: searchTerm });

  if (this.needsToSearchTopstories(searchTerm)) {
    this.fetchSearchTopstories(searchTerm);
  }

  event.preventDefault();
}

onDismiss(id) {
  const { searchKey, results } = this.state;
  const { hits, page } = results[searchKey];

  const isNotId = item => item.objectID !== id;
  const updatedHits = hits.filter(isNotId);

  this.setState({
    results: {
      ...results,
      [searchKey]: { hits: updatedHits, page }
    }
  });
}

render() {
  const {
    searchTerm,
    results,
    searchKey,
    error,
    isLoading,
    sortKey,
    isSortReverse
  } = this.state;

  const page = (
    results &&
    results[searchKey] &&
    results[searchKey].page
  ) || 0;

  const list = (
    results &&
    results[searchKey] &&
    results[searchKey].hits
  ) || [];

  return (
    <div className="page">
      <div className="interactions">
```

```
          <Search
            value={searchTerm}
            onChange={this.onSearchChange}
            onSubmit={this.onSearchSubmit}
          >
            Search
          </Search>
        </div>
        { error
          ? <div classNam="interactions">
            <p>Something went wrong.</p>
          </div>
          : <Table
                 list={list}
              sortKey={sortKey}
                isSortReverse={isSortReverse}
              onSort={this.onSort}
            onDismiss={this.onDismiss}
          />
        }
        <div classNam="interactions">
          <ButtonWithLoading
            isLoading={isLoading}
            onClick={() => this.fetchSearchTopstories(searchKey, page + 1)}>
            More
          </ButtonWithLoading>
        </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,
  sortKey,
  isSortReverse,
  onSort,
  onDismiss
}) => {
  const sortedList = SORTS[sortKey](list);
  const reverseSortedList = isSortReverse
    ? sortedList.reverse()
    : sortedList;

  return(
```

```
    <div className="table">
      <div className="table-header">
        <span style={{ width: '40%' }}>
          <Sort
            sortKey={'TITLE'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Title
          </Sort>
        </span>
        <span style={{ width: '30%' }}>
          <Sort
            sortKey={'AUTHOR'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Author
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          <Sort
            sortKey={'COMMENTS'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Comments
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          <Sort
            sortKey={'POINTS'}
            onSort={onSort}
            activeSortKey={sortKey}
          >
            Points
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          Archive
        </span>
      </div>
      { reverseSortedList.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>

const Loading = () =>
  <div>Loading ...</div>

const withLoading = (Component) => ({ isLoading, ...rest }) =>
  isLoading ? <Loading /> : <Component { ...rest } />

const ButtonWithLoading = withLoading(Button);

const Sort = ({
  sortKey,
  activeSortKey,
  onSort,
  children
}) => {
  const sortClass = classNames(
    'button-inline',
    { 'button-active': sortKey === activeSortKey }
  );

  return (
    <Button
      onClick={() => onSort(sortKey)}
      className={sortClass}
    >
      {children}
    </Button>
  );
}

export default App;

export {
  Button,
  Search,
  Table,
};
```