Systemutveckling
Ramverk

25 HP

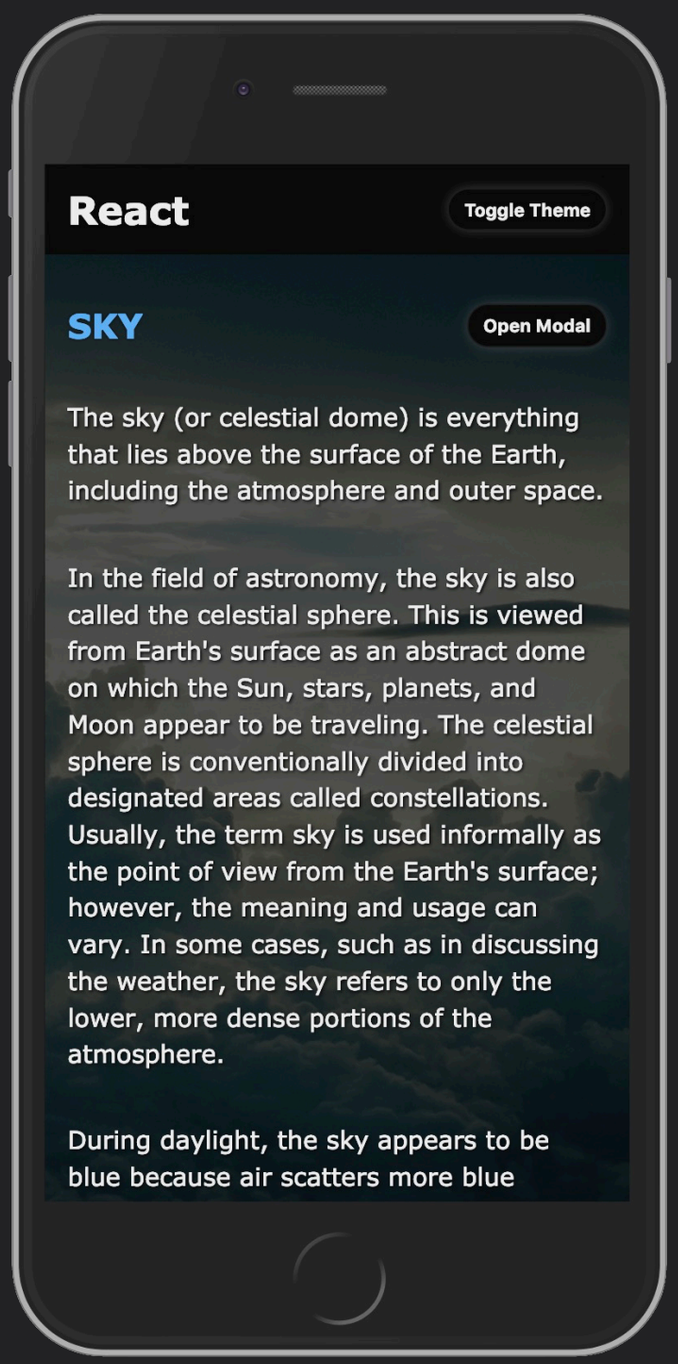
Kursupplägget

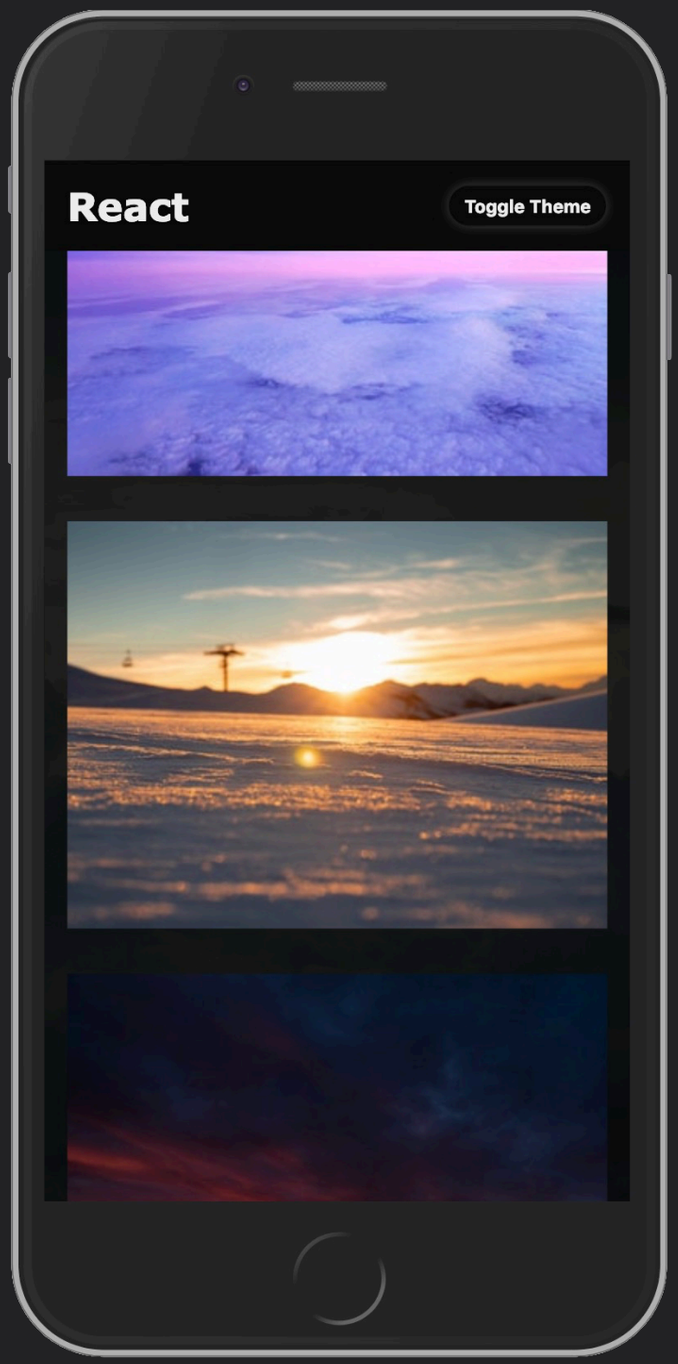
Föreläsning			Innehåll (FM)	Övningar (EM)	Inlämning
1	mån 25 mars	D	Introduktion Typescript (Basic Types, Type Inferrence, Variable Declarations, Iterators and Generators)	TypeScripts Hemsida TypeScript in 5 minutes + övning	
2	ons 27 mars	V	Typescript forts. (Functions, Classes, Interfaces, Modules)		
3	fre 29 mars	V	Introduktion React (SPA, Virtuell DOM, Kap. 1-6)	Reacts Hemsida Main Concepts inklusive övningar i CodePen Kodövning (RP)	
4	mån 1 apr.	D	React Playground (1-initial-setup + 2-layout)		Inlämning 1 ges ut
5	ons 3 apr.	V	React forts. (Kap. 7-12)		
6	fre 5 apr.	V	Create React App TS Intro	Övning ”Todo App”	Handledning
7	tis 9 apr.	D	React Playground (3-navigation-with-state) React Playground (4-navigation-with-routes)	Kodövning (RP)	
8	ons 10 apr.	V	React Playground (5-code-splitting) React Playground (6-error-boundary)	Kodövning (RP)	Inlämning 1 lämnas in
9	fre 12 apr.	V	React Playground (7-portals)	Kodövning (RP)	Inlämning 2 ges ut
10	mån 15 apr.	D	React Playground (8-code-split-app-start)	Kodövning (RP)	
11	tis 16 apr.	D	React Playground (9-api-lib-axios)	Kodövning (RP)	Handledning
12	tis 23 apr.	D	React Playground (10-context)	Kodövning (RP)	
13	tors 25 apr.	V	Tentaplugg		Inlämning 2 lämnas in
14	fre 26 apr.	V	TENTAMEN		

React Playground

Projektet vi kommer jobba med under kursen

React Playground

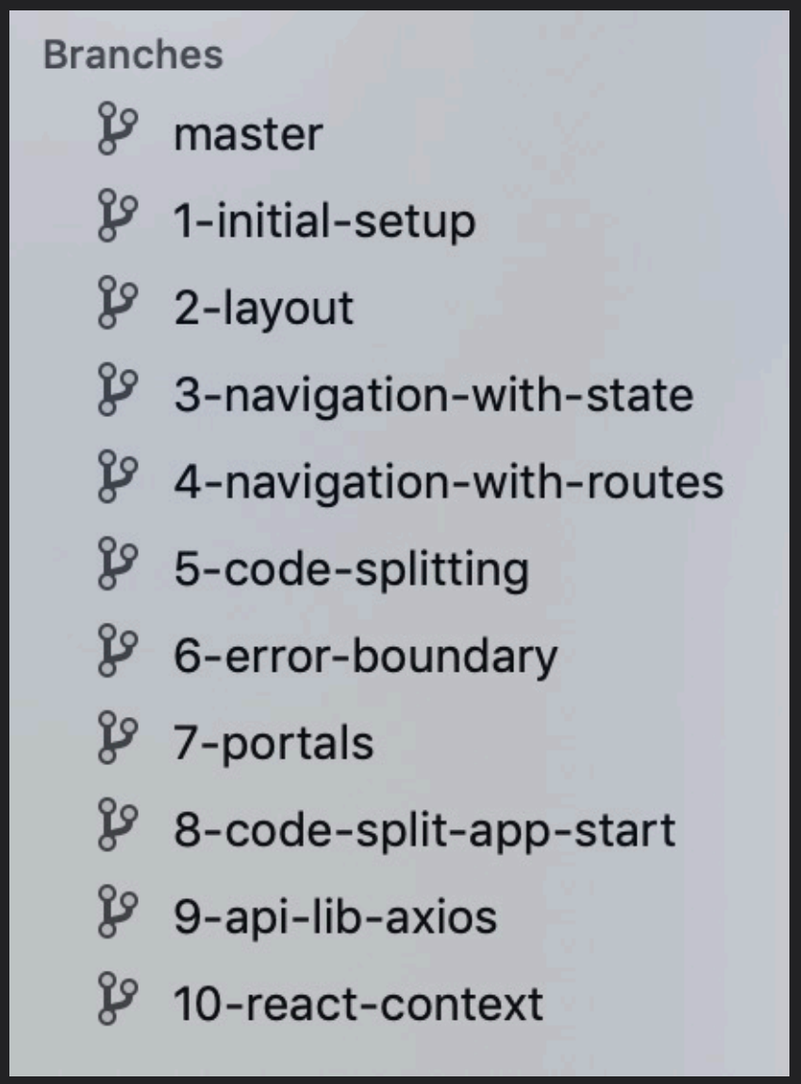




Branches

- master
- 1-initial-setup
- 2-layout
- 3-navigation-with-state
- 4-navigation-with-routes
- 5-code-splitting
- 6-error-boundary
- 7-portals
- 8-code-split-app-start
- 9-api-lib-axios
- 10-react-context

React Playground



Innan vi börjar

1. Starta det ni gjorde i förra veckan och se så att allt fungerar.

- npm run app

2. Hämta 6-error-boundary koden från #slack och utgå ifrån den.

- npm install
- npm run app

React Playground

Innan vi börjar

Ta bort testet för Error Boundary i ImageLink.tsx

```
/** React function component */
export default function ImageLink(props: Props) {

    const url = `${props.view}`;
    const imageSrc = `../assets/${props.view}.jpg`;

    if (props.view === 'forest') {
        testErrorBoundary();
    }

    return (
        <Link to={url} style={{ ...linkAppearance, ...centeredContent }}>
            <img src={imageSrc} style={fullscreenAbsolute} />
            <h1 style={{ ...centeredAbsolute, ...appearance}}>{props.view}</h1>
        </Link>
    );
}
```

Portals

Rendera kod på annan definierad plats

Portals - What’s it for !?

Ibland vill man rendera specifika element på annan plats än direkt under dess parent-element.

Komponenten som renderas med Portal kommer fortfarande fungera på samma sätt som innan i koden (med state, props osv.).

Modaler är ett bra exempel på detta.

Portals - Comparison

En portal tar in children som första parameter och container som andra parameter (i vilken dom-node komponenten skall renderas ut).

Utan portals

```
render() {
  // React mounts a new div and renders the children into it
  return (
    <div>
      {this.props.children}
    </div>
  );
}
```

Med portals

```
render() {
  // React does *not* create a new div. It renders the children into `domNode`.
  // `domNode` is any valid DOM node, regardless of its location in the DOM.
  return ReactDOM.createPortal(
    this.props.children,
    domNode
  );
}
```

Portals - Example

Parent komponent där modal renderas

```
<Modal persistent shouldClose={this.closeModal}>
    <h3>Modal opened from <span style={highlighted}>{this.view}</span> view</h3>
    <button onClick={this.closeModal}>Close modal</button>
</Modal>
```

Portals - Example

Element skapas som skall innehålla modalen.

Elementet läggs till i <body> då komponenten renderats.

Elementet tas bort ur <body> då komponenten renderats.

Portal skapas och fylls med innehåll.

Parent komponent där modal renderas

```
export default class Modal extends Component<Props> {

    element: HTMLDivElement

    constructor(props: Props) {
        super(props);
        this.element = document.createElement('div');
        this.element.id = 'modal-root';
    }

    onclick = () => {
        if (!this.props.persistent) {
            this.props.shouldClose();
        }
    }

    componentDidMount() {
        document.body.appendChild(this.element);
    }

    componentWillUnmount() {
        document.body.removeChild(this.element);
    }

    render() {
        return ReactDOM.createPortal(
            <div style={{ ...fullScreen, ...centeredContent }} onClick={this.onclick}>
                {this.props.children}
            </div>,
            this.element,
        );
    }
}
```

Portals - Example

Innan

```
<html>
▶<head>…</head>
▼<body>
  ▼<div id="app-root">
    ▼<div style="display: flex; flex-direction: column; width: 100%; height: 100%;
    background: rgb(31, 31, 31);">
      ▶<div style="height: 4em; min-height: 4em; background: black; display: flex; align-
      items: stretch; padding: 0px 1em;">…</div>
      ▶<div style="position: relative; width: 100%; height: 100%;">…</div>
      </div>
    </div>
    <!-- Will be inserted via React Portal when Modal component is mounted -->
    <!-- <div id="modal-root"></div> -->
    <!-- Dependencies -->
    <script src="./node_modules/react/umd/react.development.js"></script>
    <script src="./node_modules/react-dom/umd/react-dom.development.js"></script>
    <!-- Main -->
    <script src="./dist/main.bundle.js"></script>
    <script src="./dist/vendors~main.bundle.js"></script>
    <!-- Code injected by live-server -->
  ▶<script type="text/javascript">…</script>
  </body>
</html>
```

Portals - Example

Detta behöver inte vara i <body>.

Efter

```
<html>
▶<head>…</head>
▼<body>
  ▼<div id="app-root">
    ▼<div style="display: flex; flex-direction: column; width: 100%; height: 100%;
    background: rgb(31, 31, 31);">
      ▶<div style="height: 4em; min-height: 4em; background: black; display: flex; align-
      items: stretch; padding: 0px 1em;">…</div>
      ▶<div style="position: relative; width: 100%; height: 100%;">…</div>
      </div>
    </div>
    <!-- Will be inserted via React Portal when Modal component is mounted -->
    <!-- <div id="modal-root"></div> -->
    <!-- Dependencies -->
    <script src="./node_modules/react/umd/react.development.js"></script>
    <script src="./node_modules/react-dom/umd/react-dom.development.js"></script>
    <!-- Main -->
    <script src="./dist/main.bundle.js"></script>
    <script src="./dist/vendors~main.bundle.js"></script>
    <!-- Code injected by live-server -->
  ▶<script type="text/javascript">…</script>
  ▶<div id="modal-root">…</div>
  </body>
</html>
```

Modal renderad i body med Portal.

Portals - Event bubbling

Även då en Portal kan finnas vart som helst i DOM:en, kan den hanteras som en ”normal” komponent.

Detta inkluderar även ”Event bubblig” (kalla på funktioner ifrån parent).

A `Parent` component in `#app-root` would be able to catch an uncaught, bubbling event from the sibling node `#modal-root`.

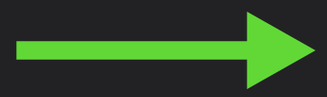
```
<html>
  <body>
    <div id="app-root"></div>
    <div id="modal-root"></div>
  </body>
</html>
```

Portals - Event bubbling

```
class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {clicks: 0};
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    // This will fire when the button in Child is clicked,
    // updating Parent's state, even though button
    // is not direct descendant in the DOM.
    this.setState(state => ({
      clicks: state.clicks + 1
    }));
  }

  render() {
    return (
      <div onClick={this.handleClick}>
        <p>Number of clicks: {this.state.clicks}</p>
        <p>
          Open up the browser DevTools
          to observe that the button
          is not a child of the div
          with the onClick handler.
        </p>
        <Modal>
          <Child />
        </Modal>
      </div>
    );
  }
}
```



Portals - Event bubbling

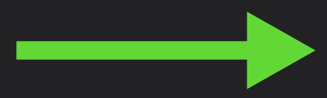
```
// These two containers are siblings in the DOM
const appRoot = document.getElementById('app-root');
const modalRoot = document.getElementById('modal-root');

class Modal extends React.Component {
  constructor(props) {
    super(props);
    this.el = document.createElement('div');
  }

  componentDidMount() {
    // The portal element is inserted in the DOM tree after
    // the Modal's children are mounted, meaning that children
    // will be mounted on a detached DOM node. If a child
    // component requires to be attached to the DOM tree
    // immediately when mounted, for example to measure a
    // DOM node, or uses 'autoFocus' in a descendant, add
    // state to Modal and only render the children when Modal
    // is inserted in the DOM tree.
    modalRoot.appendChild(this.el);
  }

  componentWillUnmount() {
    modalRoot.removeChild(this.el);
  }

  render() {
    return ReactDOM.createPortal(
      this.props.children,
      this.el,
    );
  }
}
```



Portals - Event bubbling

```
function Child() {
  // The click event on this button will bubble up to parent,
  // because there is no 'onClick' attribute defined
  return (
    <div className="modal">
      <button>Click</button>
    </div>
  );
}

ReactDOM.render(<Parent />, appRoot);
```

Catching an event bubbling up from a portal in a parent component allows the development of more flexible abstractions that are not inherently reliant on portals. For example, if you render a `<Modal />` component, the parent can capture its events regardless of whether it's implemented using portals.

Portals - Assignment (Add modal)

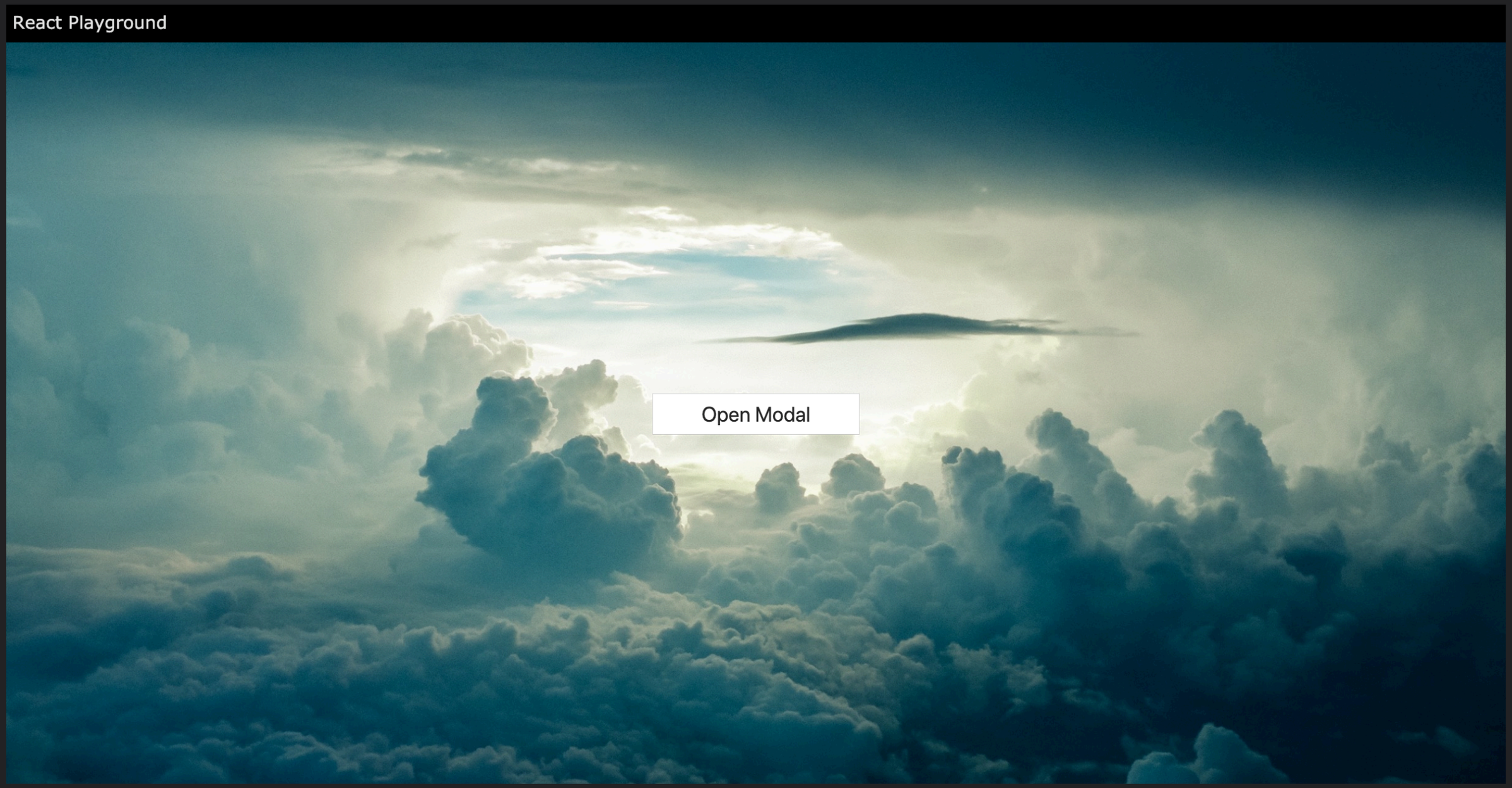
Lägg till en modal som läggs till som en ”sibling” till .

Denna modal-komponent skall via en knapp kunna öppnas i komponenten <detailview />.

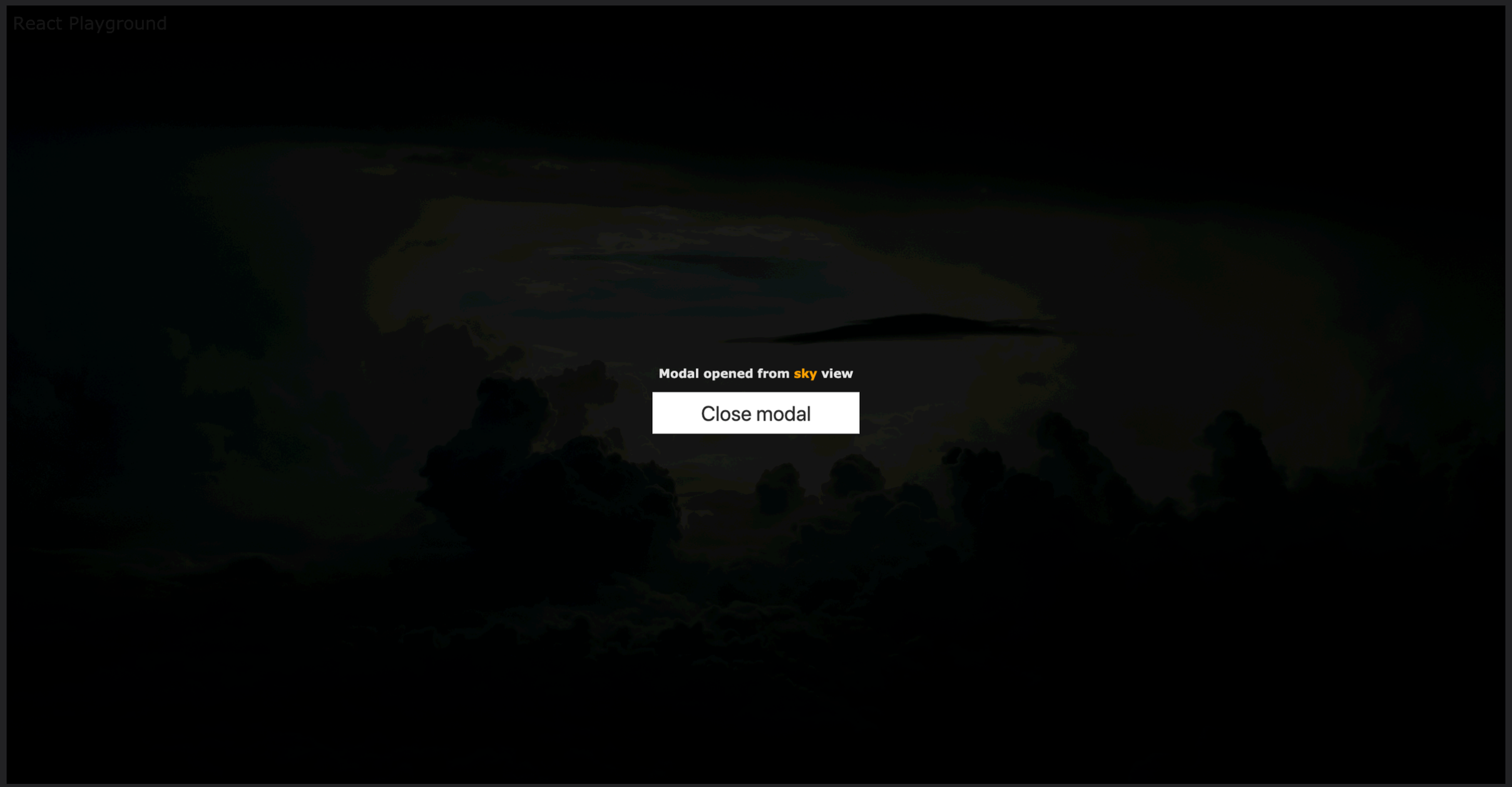
Modalen skall stängas genom att klicka någonstans på modalen när den är öppen.

I modalen skall namnet på rätt ”segment” visas upp.

Portals - Assignment example



Portals - Assignment example



Läsanvisningar

React Docs (Portals)

https://reactjs.org/docs/portals.html

Nästa lektion

React Playground

(code-split-app-start)

Tack