

Systemutveckling

Ramverk

25 HP

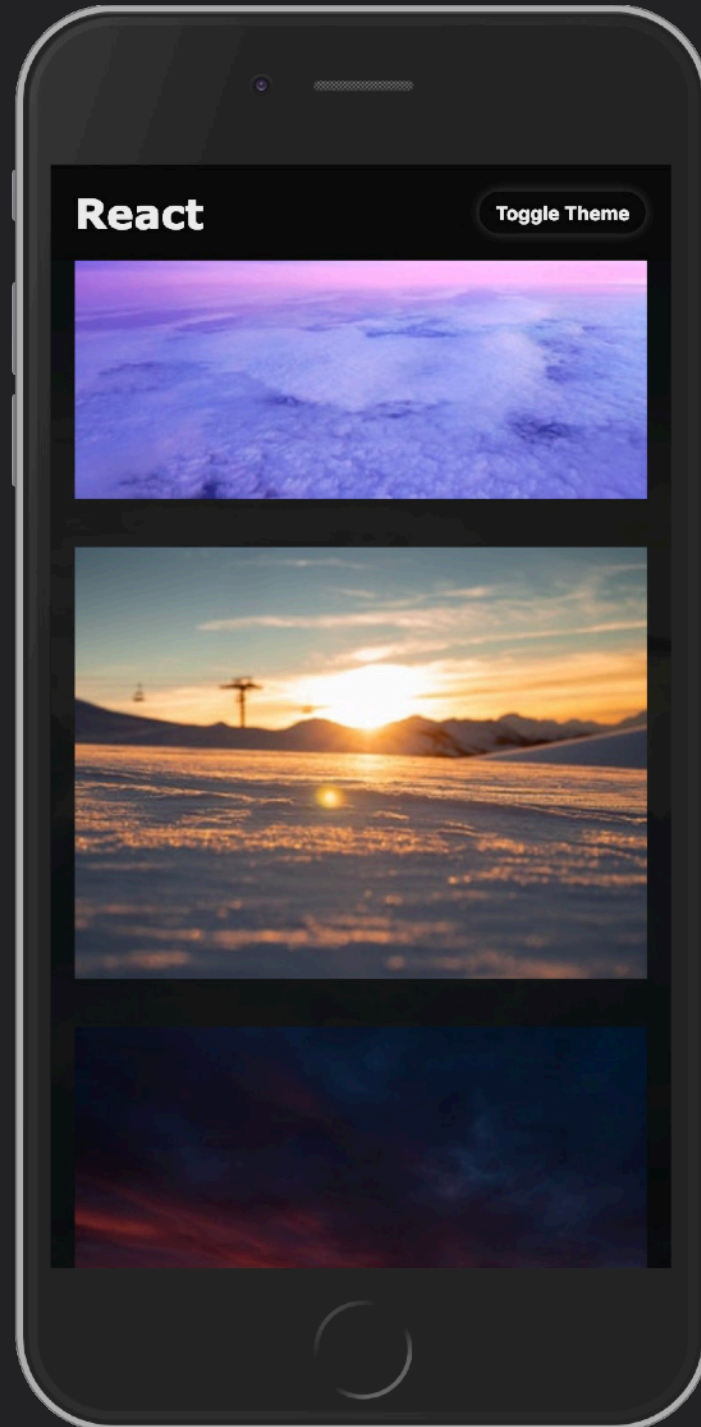
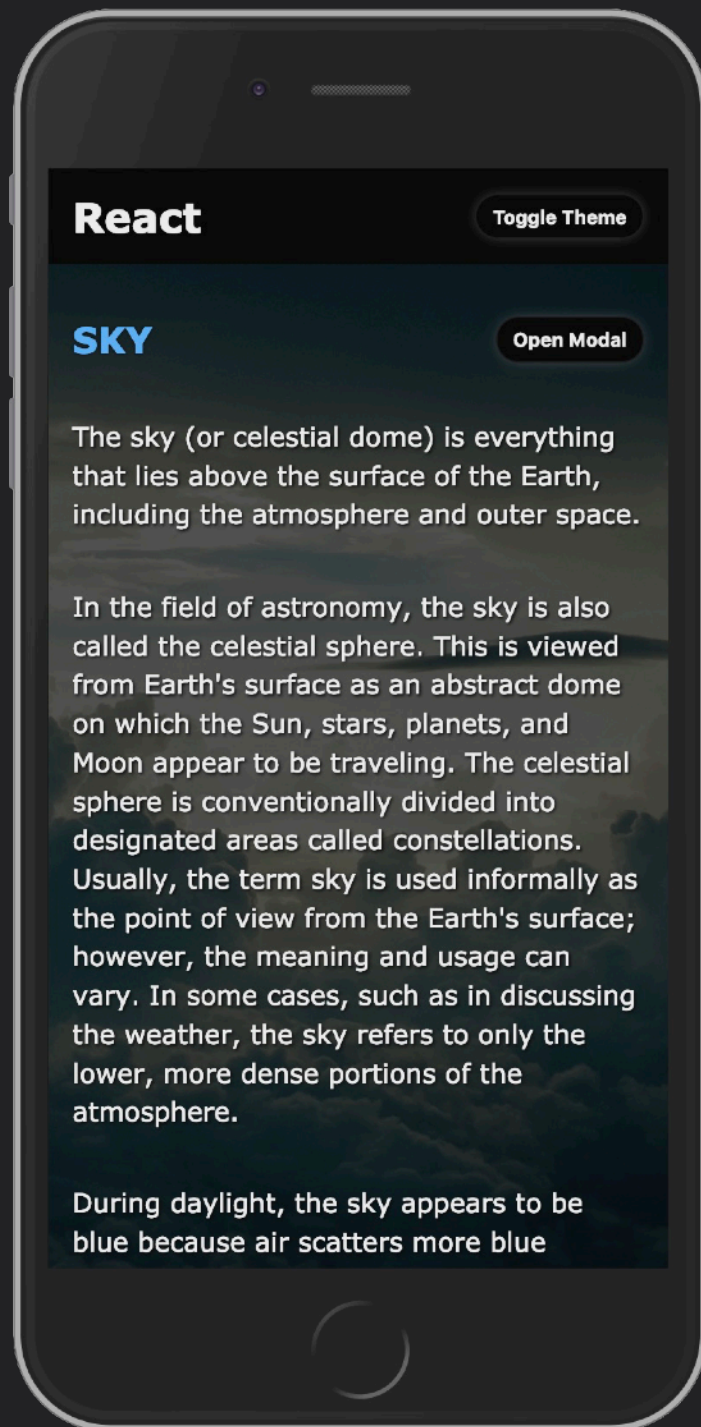
Kursupplägget

Föreläsning			Innehåll (FM)	Övningar (EM)	Inlämning
1	mån 25 mar	D	Introduktion Typescript (Basic Types, Type Inference, Variable Declarations, Iterators and Generators)	TypeScripts Hemsida TypeScript in 5 minutes + övning	
2	ons 27 mar	V	Typescript forts. (Functions, Classes, Interfaces, Modules)		
3	fre 29 mar	V	Introduktion React (SPA, Virtuellt DOM, Kap. 1-6)	React's 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)	
9	fre 12 apr	V	React Playground (7-portals)	Muntlig Presentation Kodövning (RP)	Inlämning 1 in Inlämning 2 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	tors 18 apr	V	React Playground (10-context)	Kodövning (RP)	
13	tis 23 apr	D	Algoritmer och välja Ramverk	Jobba med inlämning	Handledning
14	tors 25 apr	V	Tentaplugg		Inlämning 2 lämnas in
15	fre 26 apr	V	TENTAMEN		
Inlämning 1			Fördjupning inom ett valfritt ramverk med skriftlig och muntlig presentation samt kodexempel. Valbara ramverk Vue, Angular, Ember, Redux, Moment, Lodash, Express(node), Nest(node), Meteor(node)		
Inlämning 2			Bygga ut React Playground		
Tentamen			Skriftlig tenta med X antal frågor som har fördefinierade svar att kryssa i		












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

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

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

Context

Pass data through the component tree without Props

Context - When to use

Tillgängliggöra data mellan komponenter utan att använda props.

Användbart när många komponenter på olika nivåer behöver dela samma data.

Detta används ofta för språköversättningar, teman, skärmstorlekar, inloggad användare osv.

Data som tillgängliggörs genom context kan anses vara global.

Context - Before you use

Slippa skicka ner props genom flertalet komponenter likt detta:

```
<Page user={user} avatarSize={avatarSize} />
// ... which renders ...
<PageLayout user={user} avatarSize={avatarSize} />
// ... which renders ...
<NavigationBar user={user} avatarSize={avatarSize} />
// ... which renders ...
<Link href={user.permalink}>
  <Avatar user={user} size={avatarSize} />
</Link>
```


Context - Before you use

Ett sätt att undkomma detta kan följande lösning användas:

```
function Page(props) {
  const user = props.user;
  const userLink = (
    <Link href={user.permalink}>
      <Avatar user={user} size={props.avatarSize} />
    </Link>
  );
  return <PageLayout userLink={userLink} />;
}

// Now, we have:
<Page user={user} avatarSize={avatarSize} />
// ... which renders ...
<PageLayout userLink={...} />
// ... which renders ...
<NavigationBar userLink={...} />
// ... which renders ...
{props.userLink}
```

{...} betyder att samtliga props skickas vidare.

Context - When to use

Utan context

```
class App extends React.Component {
  render() {
    return <Toolbar theme="dark" />;
  }
}

function Toolbar(props) {
  // The Toolbar component must take an extra "theme" prop
  // and pass it to the ThemedButton. This can become painful
  // if every single button in the app needs to know the theme
  // because it would have to be passed through all components.
  return (
    <div>
      <ThemedButton theme={props.theme} />
    </div>
  );
}

class ThemedButton extends React.Component {
  render() {
    return <Button theme={this.props.theme} />;
  }
}
```

Context - When to use

Med context

```
const ThemeContext = React.createContext('light');

class App extends React.Component {
  render() {
    // Use a Provider to pass the current theme to the tree below.
    // Any component can read it, no matter how deep it is.
    // In this example, we're passing "dark" as the current value.
    return (
      <ThemeContext.Provider value="dark">
        <Toolbar />
      </ThemeContext.Provider>
    );
  }
}

// A component in the middle doesn't have to
// pass the theme down explicitly anymore.
function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

class ThemedButton extends React.Component {
  // Assign a contextType to read the current theme context.
  // React will find the closest theme Provider above and use its value.
  // In this example, the current theme is "dark".
  static contextType = ThemeContext;
  render() {
    return <Button theme={this.context} />;
  }
}
```

Context - createContext()

`createContext()` är en function som skapar en context-objekt.

När React renderar en komponent som använder en context kommer den läsa av värdet ifrån närmsta matchande providern.

Input-parametern till `createContext()` används endast då en matchande provider inte hittas (används ofta i syfte att testa koden).

```
const MyContext = React.createContext(defaultValue);
```

Context - Provider

Kan finnas flera Providers till en context, dock kommer den närmsta matchande providern uppåt i "React-trädet" alltid användas.

Förväntas skicka ner en Prop som heter value.

Ändras värdet av propen value kommer omrendering hos berörda element/komponenter ske.

value = Provider State.

Context - createContext

`contextType` anger vilken context en specifik klass/komponent hör till.

”`static contextType`” kan sättas i en klass för att definiera vilken context/provider som skall användas för den specifika klassen.

”`this.context`” ger värdet av value som är nerskickad av närmsta matchande provider.

```
class MyClass extends React.Component {  
  static contextType = MyContext;  
  render() {  
    let value = this.context;  
    /* render something based on the value */  
  }  
}
```

Context - consumer

Consumer används för att göra data ifrån en context tillgänglig till "functional components".

En consumer renderar ut en funktion som tar dess närmsta matchande providers "value" som inputparameter.

```
<MyContext.Consumer>  
  {value => /* render something based on the context value */}  
</MyContext.Consumer>
```

Context - Example

Skapar vår context samt ett objekt av "themes".



```
export const themes = {
  light: {
    foreground: '#000000',
    background: '#eeeeee',
  },
  dark: {
    foreground: '#ffffff',
    background: '#222222',
  },
};

export const ThemeContext = React.createContext(
  themes.dark // default value
);
```

Consumer används för att göra data ifrån en context tillgänglig till "functional components".



```
import {ThemeContext} from './theme-context';

class ThemedButton extends React.Component {
  render() {
    let props = this.props;
    let theme = this.context;
    return (
      <button
        {...props}
        style={{backgroundColor: theme.background}}
      />
    );
  }
}

ThemedButton.contextType = ThemeContext;

export default ThemedButton;
```


Context - Example

Vår komponent "App" ansvarar för att ha rätt "theme" i sitt state.

Providern av "ThemeContext" skickar ner en prop i propen "value".

Function för att ändra state i App skickas ner till <Toolbar>.



```
import {ThemeContext, themes} from './theme-context';
import ThemedButton from './themed-button';

// An intermediate component that uses the ThemedButton
function Toolbar(props) {
  return (
    <ThemedButton onClick={props.changeTheme}>
      Change Theme
    </ThemedButton>
  );
}
```

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      theme: themes.light,
    };

    this.toggleTheme = () => {
      this.setState(state => ({
        theme:
          state.theme === themes.dark
            ? themes.light
            : themes.dark,
      }));
    };
  }

  render() {
    // The ThemedButton button inside the ThemeProvider
    // uses the theme from state while the one outside uses
    // the default dark theme
    return (
      <Page>
        <ThemeContext.Provider value={this.state.theme}>
          <Toolbar changeTheme={this.toggleTheme} />
        </ThemeContext.Provider>
        <Section>
          <ThemedButton />
        </Section>
      </Page>
    );
  }
}

ReactDOM.render(<App />, document.root);
```

Läsanvisningar

React Docs (Context)

<https://reactjs.org/docs/context.html>

NU HAR VI FÅTT MED
ALLA DELAR 🕶️

Tack