

# Systemutveckling

## Ramverk

**25 HP**

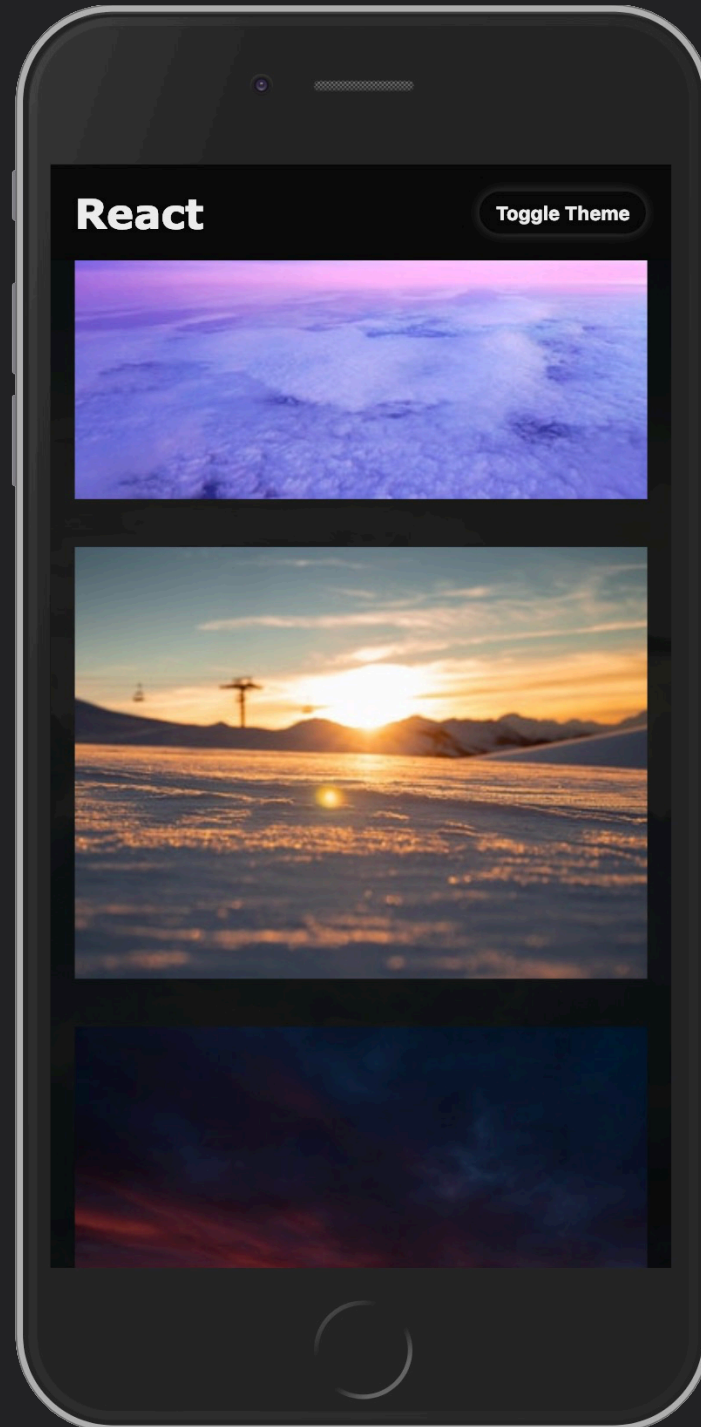
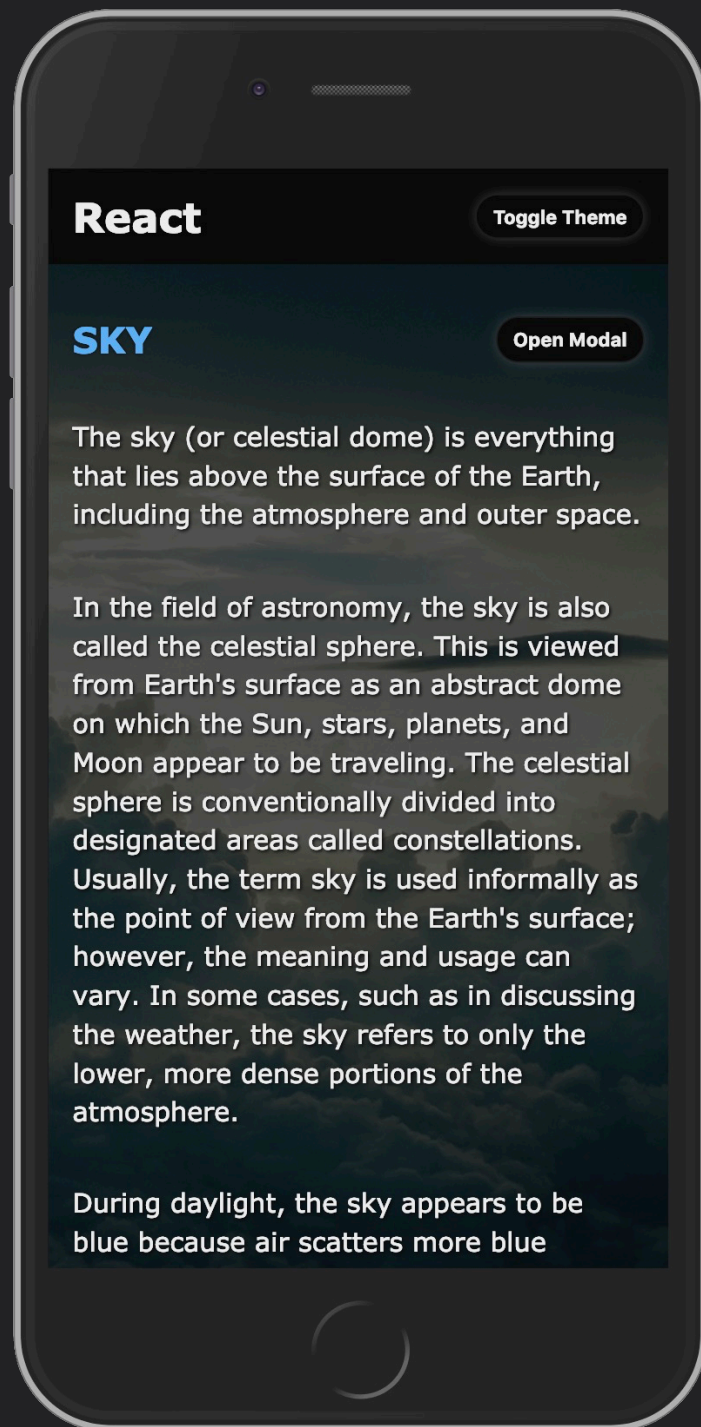
# Kursupplägget

Föreläsning			Innehåll (FM)	Övningar (EM)	Inlämning
1	mån 25 mars	D	Introduktion Typescript (Basic Types, Type Inference, 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, 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)	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

## 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 4-navigation-with-routes koden från #slack och utgå ifrån den.**

**- npm install**

**- npm run app**

# Code Splitting

En snabbare initial laddning av sidan

# Code Splitting - Why?

**Ladda in sidans innehåll dynamisk i webbläsaren.**

**Bättre upplevelse för användare med sämre uppkoppling.**

**Snabbare "PageLoad" (ladda endast in den kod som behövs).**

**Dela upp koden i olika bundles.**

# Code Splitting - Bundles

**En bundle är en fil som innehåller kompilerad kod för webbläsaren.**

**Webpack eller Browserify används ofta för detta.**

**Utan codesplitting byggs endast en bundle-fil ihop för webbläsaren.**

**Codesplitting möjliggör att det kan finnas flera bundle-filer, med olika innehåll, som kan läsas in på request.**



# Code Splitting - Without code splitting

## App:

```
// math.js
export function add(a, b) {
  return a + b;
}
```



**Exporterad funktion**

```
// app.js
import { add } from './math.js';

console.log(add(16, 26)); // 42
```



**Importerad funktion som kallas på**

## Bundle:

```
function add(a, b) {
  return a + b;
}

console.log(add(16, 26)); // 42
```



**Bundle-filen som skapas**

# Code Splitting - Dynamic import

Används idag genom Webpack eller annan kompilator

Dynamic imports tillåter applikation att ladda in delar av applikationen asynkront (med flera bundles).

Kommer snart till vanilla-JS 🙌

Innan:

```
import { add } from './math';  
  
console.log(add(16, 26));
```

Efter:

```
import("./math").then(math => {  
  console.log(math.add(16, 26));  
});
```

# Code Splitting - React.Lazy

Är byggt på Dynamic Import.

React.lazy är en funktion som låter dig rendera en dynamisk import som en vanlig komponent.

React.Lazy är inte tillgänglig för server-side rendering (endast clientside)

# Code Splitting - React.Lazy

Detta kommer automatiskt ladda en bundle som innehåller "OtherComponent" då den renderas.

Innan

```
import OtherComponent from './OtherComponent';

function MyComponent() {
  return (
    <div>
      <OtherComponent />
    </div>
  );
}
```



Efter



```
const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <div>
      <OtherComponent />
    </div>
  );
}
```

Måste vara en dynamic import!

# Code Splitting - React.Lazy & suspense

Komponenten "Suspense" tillåter en temporär rendering innan en bundle har laddat klart.

Vanligtvis brukar detta vara "Spinners/Loaders".

Denna temporära rendering implementeras med hjälp utan attributet "fallback" på komponenten <Suspense>.

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}> ←
        <OtherComponent />
      </Suspense>
    </div>
  );
}
```

Innehållet i fallback kommer renderas då "OtherComponent" inte hunnit laddas in.

# Code Splitting - React.Lazy & suspense

**Du kan även placera flera flera dynamiskt importerade komponenter inom en <Suspense> komponent. Den kommer då rendera ut sin "fallback" tills det att samtliga bundles är laddade.**

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));
const AnotherComponent = React.lazy(() => import('./AnotherComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <section>
          <OtherComponent />
          <AnotherComponent />
        </section>
      </Suspense>
    </div>
  );
}
```

# Code Splitting - Route-based

Försök att använda code-splitting vid sidbyte.

Risken är då liten att användare interagerar med andra komponenter samtidigt.

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import React, { Suspense, lazy } from 'react';

const Home = lazy(() => import('./routes/Home'));
const About = lazy(() => import('./routes/About'));

const App = () => (
  <Router>
    <Suspense fallback={<div>Loading...</div>}>
      <Switch>
        <Route exact path="/" component={Home}/>
        <Route path="/about" component={About}/>
      </Switch>
    </Suspense>
  </Router>
);
```

# Code Splitting - Name to bundles

För att döpa sina bundles används: `/* webpackChunkName: "BUNDLE_NAME" */`

```
const MasterView = React.lazy(() => import(/* webpackChunkName: "masterView" */ './MasterView'));  
const DetialView = React.lazy(() => import(/* webpackChunkName: "detailView" */ './detailView'));
```



# Code Splitting - Förberedelse

## index.html

```
<!-- Main -->
<script src="./dist/bundle.js"></script>
<script src="./dist/main.bundle.js"></script>
<script src="./dist/vendors~main.bundle.js"></script>
```

## Webpack.config.js

```
module.exports = {
  mode: "development",
  entry: "./src/index.tsx",
  output: {
    filename: "bundle.js",
    path: __dirname + "/dist"
  },
  {
    filename: "main.bundle.js",
    path: __dirname + "/dist",
    publicPath: 'dist/',
    chunkFilename: '[name].bundle.js',
  },
  optimization: {
    splitChunks: {
      chunks: 'all',
    },
  },
},
```

## tsconfig.json

```
{
  "compilerOptions": {
    "outDir": "./dist/",
    "sourceMap": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "strict": true,
    "module": "commonjs",
    "module": "esnext",
    "moduleResolution": "node",
    "target": "es5",
    "jsx": "react",
    "allowSyntheticDefaultImports": true,
    "esModuleInterop": true,
    "esModuleInterop": true,
    "lib": [ "es2015", "dom" ]
  },
  "include": [
    "./src/**/*"
  ]
}
```

# Code Splitting - Spinner + exempel

<https://www.react-spinners.com/>

Glöm inte npm install

## package.json

```
"license": "ISC",
"dependencies": {
  "@types/react": "^16.8.6",
  "@types/react-dom": "^16.8.2",
  "@types/react-router-dom": "^4.3.1",
  "react": "^16.8.3",
  "react-dom": "^16.8.3",
  "react-router-dom": "^4.3.1",
  "react-router-dom": "^4.3.1",
  "react-spinners": "^0.5.1"
},
```

## spinner.tsx

```
import React, { CSSProperties } from 'react';
import { PropagateLoader } from 'react-spinners';
import { centeredContent, fullScreen } from '../css';

export default function() {
  return (
    <div style={{ ...centeredContent, ...fullScreen }}>
      <PropagateLoader color="white" size={1.5} sizeUnit="em"/>
    </div>
  );
}

const appearance: CSSProperties = {
  color: 'white',
  fontSize: '1.5em'
}
```

## app.tsx

```
import Spinner from './spinner';

export default function App() {
  return (
    <Suspense fallback={<Spinner/>}>
      <Router>
        <Layout/>
      </Router>
    </Suspense>
  );
}
```

# Error Bondaries

Fånga upp fel och visa ett alterantivt UI

# Error Bondaries - Why?

**Ett Javascript error i UI:t borde inte få hela appen att krascha.**

**Detta kan undvikas med hjälp av Error boundary.**

**Error boundary är React-komponenter som fångar upp dess children errors.**

**Vid error kan hjälper Error boundary oss rendera annan kod.**

# Error Bondaries - Example

State som definierar  
om error hittats sätts  
till false som default.



```
class ErrorBoundary extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { hasError: false };  
  }
```

Ändrar errorState  
om ett error hittas.



```
  static getDerivedStateFromError(error) {  
    // Update state so the next render will show the fallback UI.  
    return { hasError: true };  
  }
```

Körs efter  
getDerivedStateFromError  
och loggar ut erroret.



```
  componentDidCatch(error, info) {  
    // You can also log the error to an error reporting service  
    logErrorToMyService(error, info);  
  }
```



```
  render() {  
    if (this.state.hasError) {  
      // You can render any custom fallback UI  
      return <h1>Something went wrong.</h1>;  
    }  
  
    return this.props.children;  
  }  
}
```

```
<ErrorBoundary>  
  <MyWidget />  
</ErrorBoundary>
```

# Error Bondaries - Good to know

## Note

`Error` boundaries only catch `errors` in the components **below** them in the tree. An `error` boundary can't catch an `error` within itself.

## Note

`getDerivedStateFromError()` is called during the "render" phase, so side-effects are not permitted. For those use cases, use `componentDidCatch()` instead.

## Note

Error boundaries do **not** catch errors for:

- Event handlers ([learn more](#))
- Asynchronous code (e.g. `setTimeout` or `requestAnimationFrame` callbacks)
- Server side rendering
- Errors thrown in the error boundary itself (rather than its children)

# Error Bondaries - Events Handlers

Använd try/catch för att hantera event-errors.



```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { error: null };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    try {
      // Do something that could throw
    } catch (error) {
      this.setState({ error });
    }
  }

  render() {
    if (this.state.error) {
      return <h1>Caught an error.</h1>
    }
    return <div onClick={this.handleClick}>Click Me</div>
  }
}
```

# Läsanvisningar

## React Docs (Code Splitting)

<https://reactjs.org/docs/code-splitting.html>

## React Docs (Error Boundaries)

<https://reactjs.org/docs/error-boundaries.html>



# Nästa lektion

React Playground  
(Portals)

**Tack**