

React basics

Juha Hinkula
github: juhahinkula

React

- Javascript library for building user interfaces
- Uses declarative way to define the UI and its changes
- Developed by Facebook
- <https://facebook.github.io/react/>
- React is component based and components are reusable

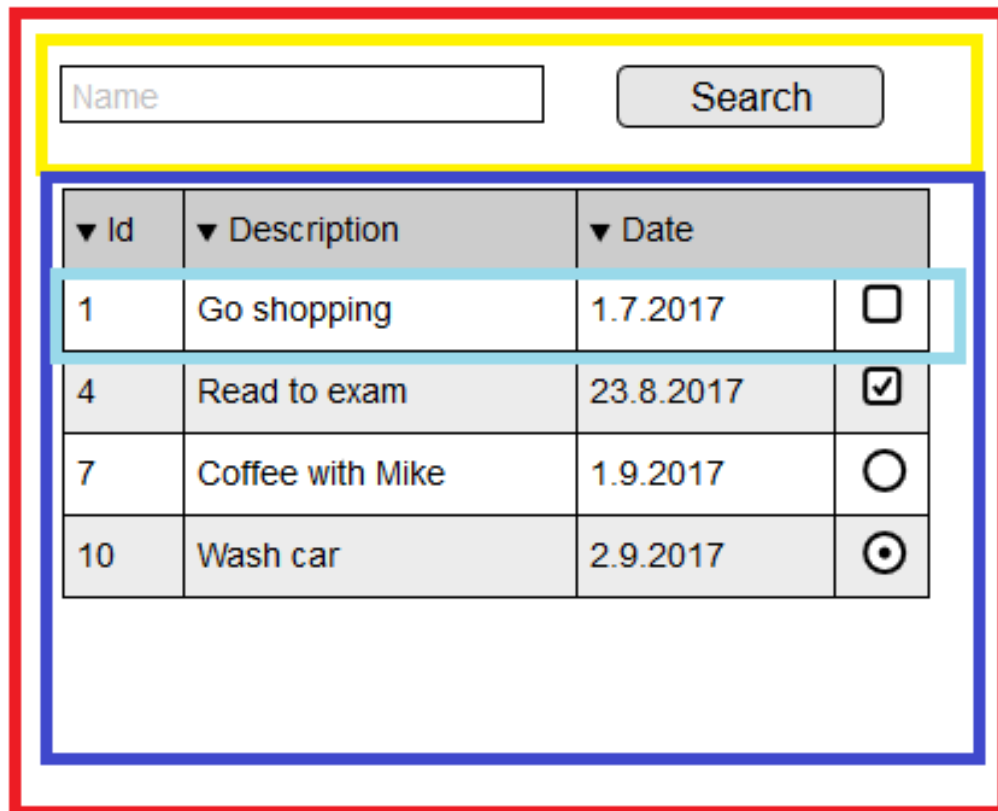
React

Red: Root component

Yellow: Search bar component

Blue: Table component

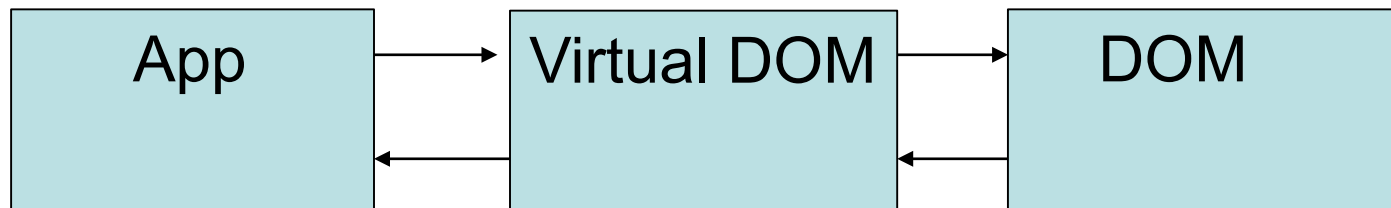
Light blue: Table row component



▼ Id	▼ Description	▼ Date	
1	Go shopping	1.7.2017	<input type="checkbox"/>
4	Read to exam	23.8.2017	<input checked="" type="checkbox"/>
7	Coffee with Mike	1.9.2017	<input type="radio"/>
10	Wash car	2.9.2017	<input type="radio"/>

React: Virtual DOM

- Constant direct changes to DOM would be inefficient and slow. DOM manipulation can trigger style changes, tree modifications and re-rendering.
- React uses **virtual DOM** to make this more efficient
- All changes are done first to virtual DOM which is then compared to current DOM. Only changed parts of the DOM are updated.
- React also batches DOM manipulations



ECMAScript 6 (ES6)

- This material, like all recent React tutorials, uses the ES6 syntax that gives us a lot of new features.
- Some examples:

- **Classes**

```
class Shape {  
  constructor (id, x, y) {  
    this.id = id  
    this.move(x, y)  
  }  
  move (x, y) {           // a method that all Shape objects will have  
    this.x = x  
    this.y = y  
  }  
}
```

ECMAScript 6 (ES6)

- Some examples:

- **Inheritance**

```
class Circle extends Shape {  
  constructor (id, x, y, radius) {  
    super(id, x, y)  
    this.radius = radius  
  }  
}
```

ECMAScript 6 (ES6)

- Some examples:

- **Arrow functions**

```
// An anonymous arrow function without curly  
// brackets which leads to implicit return.  
// Parentheses around single parameter x can be skipped  
x => x + 1;
```

```
// Same as
```

```
function(x) {  
    return x + 1;  
}
```

ECMAScript 6 (ES6)

- Some examples:

- **let** keyword (block scope)

```
let age = 24
```

- **const** keyword (constants, block scope)

```
const PI = 3.141593
```

- **String interpolation**

```
let person = {firstname: 'Jack', lastname: 'Russell'}  
let msg = `Hello ${person.firstname} ${person.lastname}`;
```



React state & props

- There are two types of data that react components "listen" or react to: *state* and *props*
- **Props** are get from the parent and they are not going to change during the lifetime of the component
- Props are just parameters that are given to component when it is created.
- **State** is used for the **data model** that is going to change
- The component is re-rendered when state or props are changed.

HelloWorld component

- React.Component is abstract base class that will be typically subclassed by your own components

```
class HelloComponent extends React.Component {  
  render() {  
    return <div>Hello World</div>;  
  }  
}
```

```
ReactDOM.render(<HelloComponent />, document.getElementById('app'));
```



HelloWorld component with props

```
class HelloComponent extends React.Component {  
  render() {  
    return <div>Hello World {this.props.firstname}</div>;  
  }  
}
```

```
ReactDOM.render(<HelloComponent firstname="John" />,  
  document.getElementById('app'));
```



Render

- React component render() function returns elements that are going to be rendered in to the screen.
- Render must return single element but you can wrap multiple elements inside one parent element. Or you can use fragments (since React v 16.x)

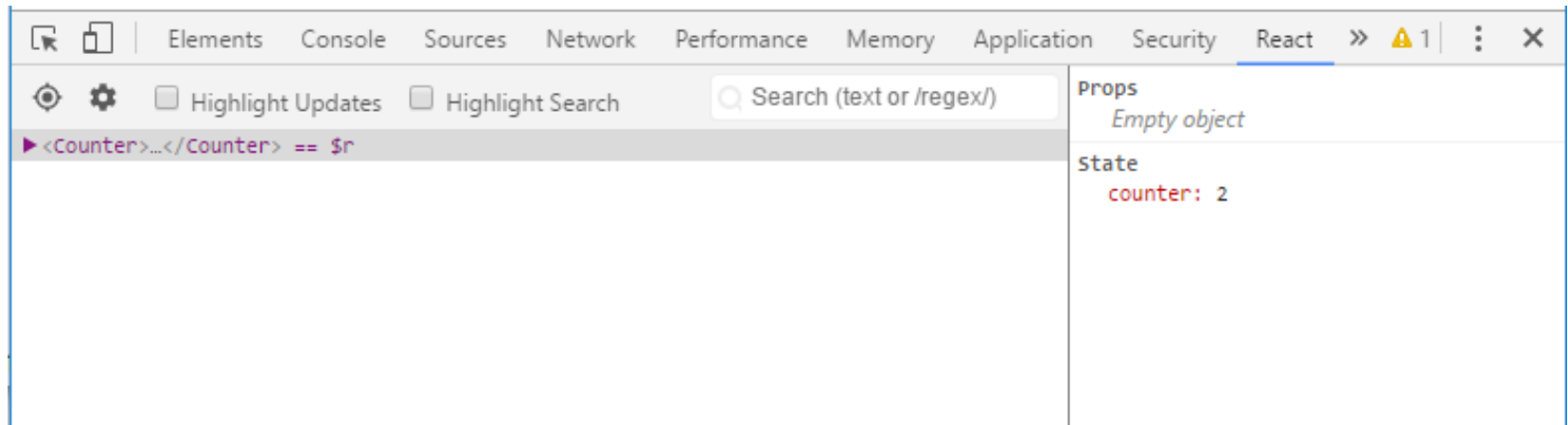
```
render() {  
  return [  
    <div>  
      <h1>This is main header</h1>  
      <p>Main text paragraph</p>  
    </div>  
  ]  
}
```

```
// Fragments  
render() {  
  return [  
    <>  
      <h1>This is main header</h1>  
      <p>Main text paragraph</p>  
    </>  
  ]  
}
```



React Developer Tools

- React Developer Tools is available as Chrome plugin or Firefox add-on.



Exercise 1

- Copy the template code to empty html file from the address <http://bit.ly/2eKPjzh>
- Create the hello world example according to previous slides

React state

- State is initialized in the constructor

```
class MyComponent extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {firstname: ''};  
  }  
  //continues...
```

React state

- State is always changed by using `setState` method

```
this.setState(  
  {firstname: 'John'}  
);
```


React state

- **Do not** change state directly

```
this.state.firstname = 'John'; //WRONG
```

- **Always** use `setState` method. By `setState` calls React knows when the state has changed and re-rendering is needed:

React state

- **setState** calls are asynchronous therefore state is not necessarily changed immediately after the call.
- When you are changing the state with values that depend on the current state you should pass the function instead of object to setState. That makes sure that update uses the latest version of the state (<https://reactjs.org/docs/faq-state.html>).

```
this.setState((prevState) => {  
  return {count: prevState.count + 1}  
});
```





Counter example

```
class Counter extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {counter: 0};  
  }  
  
  buttonPressed = () => {  
    this.setState((prevState) => {  
      return {counter: prevState.counter + 1}  
    });  
  }  
  
  render() {  
    return (  
      <div>  
        <div>Counter: {this.state.counter}</div>  
        <button onClick={this.buttonPressed}>Press me</button>  
      </div>  
    );  
  }  
}
```



React JSX

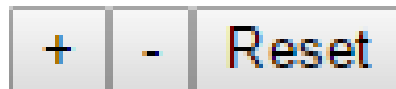
- JSX is javascript syntax extension which is recommended to use with React
- In counter example we had JSX expression
`<div>Counter: {this.state.counter}</div>`
- We can embed javascript to JSX by wrapping it in curly braces
- After compilation, JSX expressions becomes regular JavaScript objects



Exercise 2

- Modify counter exercise by adding increment, decrement and reset buttons
- Starter template <http://bit.ly/2j9HnMV>

Counter: 5



Stateless component

- Stateless (or Functional) component doesn't have state.
- Can be defined using ES6 function and it gets props as parameter.

```
const HelloMessage = (props) => {  
  return <h1>Hello {props.msg}</h1>  
}
```

```
const element = <HelloMessage msg="Again" />;  
ReactDOM.render(element, document.getElementById('root'));
```



React Hooks

- With Hooks you can create React component that has state using function instead of class (**useState** hook).
- Usage of **useState()** Hook

```
const HooksCounter = () => {  
  const [count, setCount] = React.useState(0);
```

...

- Example below creates state variable called `count` and function `setCount` is used to update its value.
- `setCount` takes one argument that is initial value of the state.



React Hooks

- Now you can update `count` state value by using `setCount` and UI will be then re-rendered.

```
setCount(count + 1)
```


React Hooks

- Below is the counter example done with Hooks.

```
const HooksCounter = () => {  
  const [count, setCount] = React.useState(0);  
  
  return (  
    <div>  
      <p>You clicked {count} times</p>  
      <button onClick={() => setCount(count + 1)}>Increment</button>  
    </div>  
  );  
}
```



React Hooks

- If you have multiple state variables

```
const [firstName, setFirstName] = React.useState("John");  
const [lastName, setLastName] = React.useState("Johnson");
```

- Or use object

```
const [name, setName] = React.useState({  
  firstName: 'John',  
  lastName: 'Johnson'  
});
```



React Hooks

- If you use object you can update values using setName function and passing new object as parameter

```
setName({ firstName: 'Jim', lastName: 'Smith'})
```

- Or if only one value is updated, you can use object spread syntax

```
setName({ ...name, lastName: 'Smith'})
```

```
// New value is now firstName: John, lastName: Smith
```



Exercise 3

- Create following counter by using React Hooks

Counter: 5

+	-	Reset
---	---	-------

React user input

- Use input element's `onChange` and `value` attributes
- `onChange` is invoked in every keystroke and it calls `inputChanged` method which updates react state

```
<input type="text" value={this.state.name}  
      onChange={this.inputChanged} />
```

- Create `inputChanged` method to update state

```
inputChanged = (event) => {  
  this.setState({name: event.target.value});  
};
```





HelloName example

```
class HelloName extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {name: ''};  
  }  
  
  inputChanged = (event) => {  
    this.setState({name: event.target.value});  
  };  
  
  render() {  
    return (  
      <div>  
        <div>Hello {this.state.name}</div>  
        <input type="text" value={this.state.name}  
          onChange={this.inputChanged} />  
      </div>  
    );  
  }  
}
```

Hello John





HelloName example using Hooks

```
const HelloName = () => {  
  const [name, setName] = React.useState('');  
  
  const inputChanged = (event) => {  
    setName(event.target.value);  
  };  
  
  return (  
    <div>  
      <div>Hello {name}</div>  
      <input type="text" value={name} onChange={inputChanged} />  
    </div>  
  );  
};
```

Hello John

- By using Hooks your code comes more compact and you can avoid `this` keyword



React Component lifecycle methods

- **constructor(props)**

- Called before component renders first time
- Set initial state here

- **componentDidMount()**

- Called when component has been rendered first time. Good for fetching data to your component.

- **componentWillUnmount()**

- Called before component is unmounted

Note! Only component that is defined using **class** have lifecycle methods

Mount (in React terminology) \approx An instance of a component is being created and inserted into the DOM



Example

```
class HelloName extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {firstname: ''};  
  }  
  
  componentDidMount() {  
    this.setState({firstname: 'John'});  
  }  
  
  render() {  
    return (  
      <div>Hello {this.state.firstname}</div>  
    );  
  }  
}
```



React Hooks

- **useEffect** hook
- In the functional component you cannot use lifecycle methods. **useEffect** hook act like **componentDidMount** and **componentDidUpdate** lifecycle metods. It is invoked each time after component has been rendered.

```
// Similar to componentDidMount and componentDidUpdate
useEffect(() => {
  // Do something here
});
```

React Hooks

- We can pass array as a second argument to **useEffect** hook.
- In the following example we pass count to useEffect hook. In that case effect re-runs only if the count state changes.

```
// This is executed only if the count state changes
useEffect(() => {
  // Do something here
}, [count]);
```



React Hooks

- You can also pass an empty array to useEffect hook. Then it act like componentDidMount (runs only after the first render)

```
// This is executed only after the first render
useEffect(() => {
  // Do something here
}, []);
```



Networking

- React provides **Fetch API** for handling web requests
- Fetch takes URL as a first argument
- Fetch is asynchronous operation and it provides promises which makes response handling easier.

```
fetch('https://mydomain.com/api')  
  .then(function(response) {  
    // Handle response  
  })  
  .catch(function(err) {  
    // Something went wrong  
  });
```



Networking

- In this material we mostly use promises but you can use `async/await` as well.

```
fetchData = async () => {  
  try {  
    const response = await fetch('https://mydomain.com/api');  
    const data = await response.json();  
  }  
  catch(error) {  
    console.error(error);  
  }  
}
```





Networking

- Method and header can be added as a second argument to fetch

```
fetch('https://mydomain.com/api', {  
  method: 'POST',  
  headers: { 'Accept': 'application/json',  
    'Content-Type': 'application/json', }  
})  
.then(function(response) {  
  // Handle response  
})  
.catch(function(err) {  
  // Something went wrong  
});
```



Networking example

- Following example uses NASA APOD api which shows astronomy picture of the date.
(<https://api.nasa.gov/api.html#apod>)
- The example makes call to rest api and shows daily image and explanation in the page.
- Rest api can be called by using following URL
https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY
- Call returns explanation and image URL as an JSON format





Networking example

■ Response

```
{
  "copyright": "Alson Wong",
  "date": "2017-09-20",
  "explanation": "Most photographs don't adequately portray the magnificent
unparalleled. The human eye can adapt to see coronal features and external
picture is a combination of forty exposures from one thousandth of a second.
features of the total solar eclipse that occurred in August of 2017. Coronal
and magnetic fields in the Sun's corona. Looping prominences appear bright
made out, illuminated by sunlight reflected from the dayside of the Full Moon."
  "hdurl": "https://apod.nasa.gov/apod/image/1709/Corona_Wong_5156.jpg",
  "media_type": "image",
  "service_version": "v1",
  "title": "The Big Corona",
  "url": "https://apod.nasa.gov/apod/image/1709/Corona_Wong_960.jpg"
}
```





Networking example

- We need states to get image url and explanation
- States will be initialized in constructor

```
constructor(props) {  
  super(props);  
  this.state = {explanation: '', imgurl: ''};  
}
```

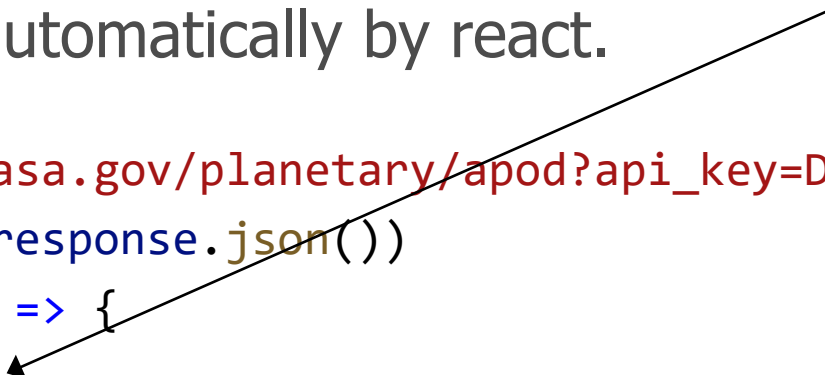




Networking example

- Fetch API call is made inside `componentDidMount()` method.
- When response arrives the parsed values are setted to states and UI is re-rendered automatically by react.

```
componentDidMount() {  
  fetch('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY')  
    .then((response) => response.json())  
    .then((responseData) => {  
      this.setState({  
        explanation: responseData.explanation,  
        imgurl: responseData.url  
      });  
    });  
}  
}
```





Networking example

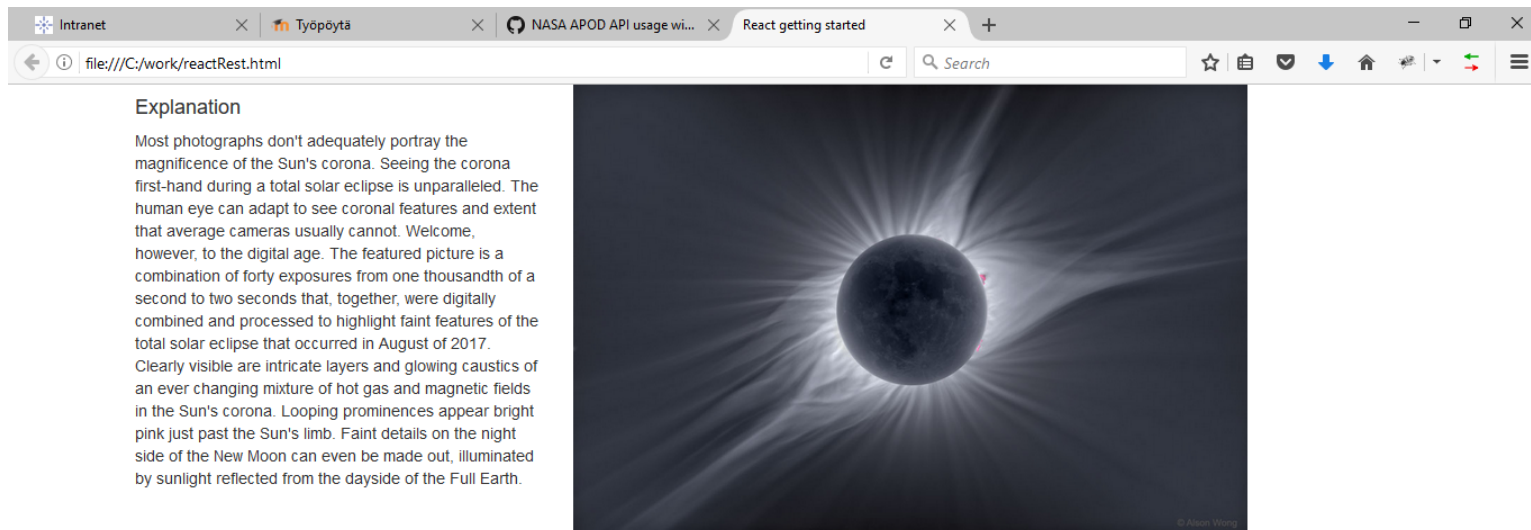
- Render method

```
render() {  
  return (  
    <div>  
      <div>Explanation: {this.state.explanation}</div>  
      <div><img src={this.state.imgurl} /></div>  
    </div>  
  );  
}
```



Networking example

Source code: <http://bit.ly/2xeh8Le>



Exercise 4

- Register to OpenWeatherMap.org to get your API key
- Use URL to get current weather from your city
`api.openweathermap.org/data/2.5/weather?q=London
&APPID=YOUR_APP_ID`
- Show weatherinfo

Temperature: 20.2 Celsius

Weather: Clear



React: handling lists

- `map()` function creates a new array with the results of calling a function for every array element
- `map()` function (javascript)

```
let arrA = [1, 2, 3];  
let arrB = arrA.map((x) => x * 2 );  
// arrB = [2, 4, 6]
```
- Following map statement returns new array with values multiplied by 2.





React: handling lists

- This example creates an array of listitems and set it to react state.
- **Note!** **key** string attribute is needed in the lists. That helps react to identify which rows have changed.

```
this.state = {listItems: []};  
componentDidMount() {  
  const numbers = [1, 2, 3, 4, 5];  
  this.setState({listItems: numbers});  
}  
render() {  
  const rows = this.state.listItems.map((number, index) =>  
    <li key={index}>Listitem {number}</li>  
  );  
  return (  
    <div>  
      <ul>{rows}</ul>  
    </div>  
  );  
}
```

- Listitem 1
- Listitem 2
- Listitem 3
- Listitem 4
- Listitem 5



React: Asteroids Example

- This example fetch an array of listitems from the REST web service and set it to react state.
- The rest service to be used is NASA Asteroids API. It returns the list of asteroids based on their closest approach date to earth
(<https://api.nasa.gov/api.html#NeoWS>)

- Example query

https://api.nasa.gov/neo/rest/v1/feed?start_date=2015-09-07&end_date=2015-09-08&api_key=DEMO_KEY



React: Asteroids Example

- We need an array for the list of asteroids returned from the rest call.
- The list state is initialized in the constructor

```
constructor(props) {  
  super(props);  
  this.state = {listItems: []};  
}
```

React: Asteroids Example

- Fetch call is done inside `componentDidMount()` method

```
componentDidMount() {  
  fetch('https://api.nasa.gov/neo/rest/v1/feed?start_date=2017-10-  
    09&end_date=2017-10-09&api_key=DEMO_KEY')  
    .then((response) => response.json())  
    .then((responseData) => {  
      this.setState({  
        listItems: responseData.near_earth_objects["2017-10-09"],  
      });  
    })  
}
```



React: Asteroids Example

■ render() method

```
render() {  
  const itemRows = this.state.listItems.map((asteroid) =>  
    <tr key={asteroid.name}>  
      <td>{asteroid.name}</td>  
      <td>{asteroid.close_approach_data[0].miss_distance.kilometers}</td>  
    </tr>  
  )  
  return (  
    <div>  
      <h2>Closest asteroids today</h2>  
      <table><tbody>  
        <tr><th>Name</th><th>Min distance</th></tr>  
        {itemRows}  
      </tbody></table>  
    </div>  
  );  
}
```





React: Asteroids Example

- Source code: <http://bit.ly/2xns7zE>

Closest asteroids today

Name	Distance (km)
(2017 RN2)	28852688
(2002 FD6)	55694380
(2002 QD7)	19416478
(2014 GC35)	41886912
(2014 WA)	70845200
(2016 VZ)	41359896



Exercise 5: step 1

- Github repository list: Use Github API
 - <https://api.github.com/search/repositories?q=react>
- Show the list of repositories by keyword (fullname + URL)

Repositories

Name	URL
facebook/react	https://github.com/facebook/react
reactphp/react	https://github.com/reactphp/react
duxianwei520/react	https://github.com/duxianwei520/react
discountry/react	https://github.com/discountry/react
bailicangdu/react-pxq	https://github.com/bailicangdu/react-pxq
Cathy0807/react	https://github.com/Cathy0807/react
azat-co/react	https://github.com/azat-co/react



Exercise 6: step 2

- Add inputbox for search keyword
- Add button which executes the fetch by given keyword when it is pressed

Repositories

<input type="text" value="java"/>	<input type="button" value="Search"/>
Name	URL
hmkcode/Java	https://github.com/hmkcode/Java
pubnub/java	https://github.com/pubnub/java
DuGuQiuBai/Java	https://github.com/DuGuQiuBai/Java
agileorbit-cookbooks/java	https://github.com/agileorbit-cookbooks/java
dockerfile/java	https://github.com/dockerfile/java
json-iterator/java	https://github.com/json-iterator/java
gaopu/Java	https://github.com/gaopu/Java



React Forms

- Form handling is a little bit different with React. HTML form will navigate to other page when it is submitted.
- Common case is that we want to invoke javascript function that has access to form data after submission.



React Forms

```
inputChanged = (event) => {  
  this.setState({sometext: event.target.value});  
}
```

```
addTodo = (event) => {  
  event.preventDefault(); // ignores the default action  
  // Do something with form data  
}
```

...inside render method

```
<form onSubmit={this.addTodo}>  
  <input type="text" onChange={this.inputChanged}  
    value={this.state.sometext}/>  
  <input type="submit" value="Add"/>  
</form>
```





React Forms

■ Multiple input elements

- Add a name attribute to inputs and use name field in eventhandler (name attribute values should be the same as states)

```
inputChanged = (event) => {  
  this.setState({[event.target.name]: event.target.value});  
}
```

..inside render method

```
<form onSubmit={this.addPerson}>  
  <input type="text" name="firstname"  
    onChange={this.inputChanged} value={this.state.firstname}/>  
  <input type="text" name="lastname"  
    onChange={this.inputChanged} value={this.state.lastname}/>  
  <input type="submit" value="Add"/>  
</form>
```

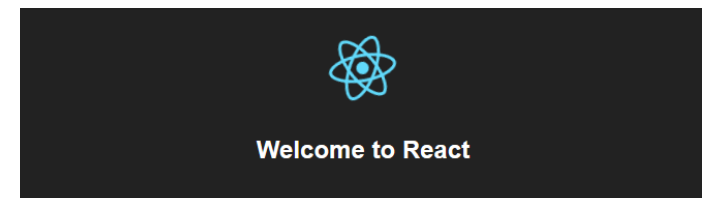


React app

- Facebook has created react app starter kit called 'Create React App'
- Create react app needs Node version ≥ 8.10
- Usage

```
npx create-react-app my-app  
cd my-app  
npm start
```

- Navigate to <http://localhost:3000>

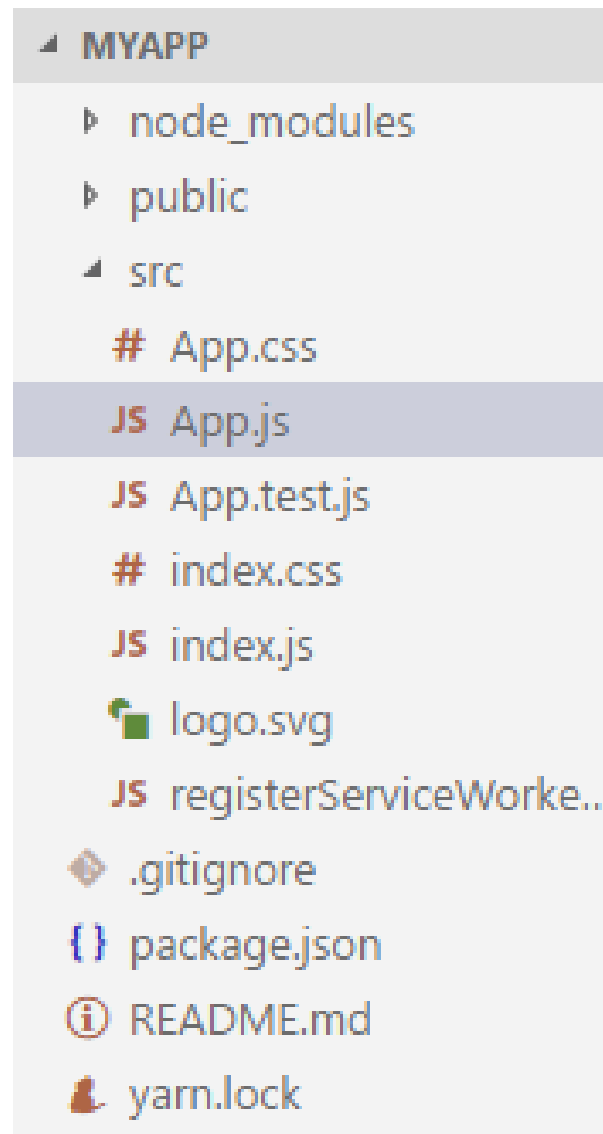


To get started, edit `src/App.js` and save to reload.



React

- Create-react-app creates following folder structure
- App.js is the main javascript file where modifications are made
- If you have bigger app with multiple components, it is better to create own folder for these



React

App.js file

- **import** statements are used to import libraries, react components, stylesheet and assets to app
- **export** statement allows you to import component to another file by using import statement

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a className="App-link" href="https://reactjs.org" target="_blank" rel="noopener noreferrer">
          Learn React
        </a>
      </header>
    </div>
  );
}
export default App;
```



React

index.js file

imports App component and renders it to index.html files 'root' element.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />, document.getElementById('root'));
serviceWorker.unregister();
```



React: Todolist example

- Create a new React App using create-react-app
- Create a new functional component called Todolist.js inside the src folder.
- Todolist example has only one description field called 'desc'.
- We need one state for the description and one array state for all todos. Let's introduce states using useState hook.

```
const [desc, setDesc] = useState('');  
const [todos, setTodos] = useState([]);
```



React: Todolist example

- Form is used to type description for a new todo item.
- `addTodo` method is invoked when form is submitted

...inside the return statement

```
<form onSubmit={addTodo}>  
  <input type="text" onChange={inputChanged}  
    value={desc}/>  
  <input type="submit" value="Add"/>  
</form>
```



React: Todolist example

- `addTodo` method adds new todo item to todos array state.
- Method uses spread notation (...) to add new item to the end of the existing array.

```
const inputChanged = (event) => {  
  setDesc(event.target.value);  
}
```

```
const addTodo = (event) => {  
  event.preventDefault();  
  setTodos([...todos, desc]);  
}
```

React: Todolist example

- Next, we add table element to the return statement and render all todos inside the table using map function.

```
return (  
  <div>  
    <form onSubmit={addTodo}>  
      <input type="text" onChange={inputChanged} value={desc}/>  
      <input type="submit" value="Add"/>  
    </form>  
    <table><tbody>  
      {todos.map(todo => <tr><td>{todo}</td></tr>)}  
    </tbody></table>  
  </div>  

```



React: Todolist example

- Finally, we render our Todolist component inside the App.js return statement

```
import React from 'react';
import './App.css';
import Todolist from './Todolist';

function App() {
  return (
    <div className="App">
      <header className="App-header">TodoList</header>
      <Todolist />
    </div>
  );
}

export default App;
```



Haaga-Helia

React: Todolist example

- See the source code (Todolist.js) <http://bit.ly/2wyoBkx>

Todolist

- Coffee with Mike
- Go swimming
- Study React.js

Hint! You can center table items by adding following lines to App.css file

```
table {  
  margin-left:auto;  
  margin-right:auto;  
}
```



Exercise 6

- Add new date column to the previous todomlist example

Hints:

- Add new state and input field (use name attributes in the input fields)
- addTodo method: insert todo **objects** inside todos array

Simple Todolist

Add todo:

Description: Date:

Date	Description
10.11.2017	Go shopping
23.11.2017	Meet friends in downtown



Exercise 7

- Add delete button to your todo list table which deletes todo item

Hints:

- Create a new method which is called when button is pressed. Set button's id to row index which can be then used in method to delete correct item.
- Use `filter()` method to delete one item from the array
`todos.filter((todo, i) => i !== index)`

Simple Todolist

Add todo:

Description: Date:

Date	Description	
12.11.2017	Go shopping	<input type="button" value="Delete"/>
23.11.2017	Meet friends in downtown	<input type="button" value="Delete"/>



React: Separate components

- Let's split todolist example application to multiple components.
- We will add new **stateless** component Called TodoTable and separate it from App component
- Add new file TodoTable.js in the src folder. The skeleton code of the compnent is shown below. **Note!** It is stateless component and defined by using ES6 function.

```
import React from 'react';
const TodoTable = (props) => {
  return (
    <div>
    </div>
  );
}
export default TodoTable;
```



React: Separate components

- With stateless component you don't need constructor and render methods.
- You can access **props** without **this** keyword.

```
return (  
  <div>  
    <table>  
      <tbody>  
        <tr><th>Date</th><th>Description</th></tr>  
        {props.todos.map((item, index) => <tr key={index}>  
          <td>{item.date}</td><td>{item.description}</td></tr>)}  
      </tbody>  
    </table>  
  </div>  
)
```



React: Separate components

- Import TodoTable component to Todolist component (modify Todolist.js file)

```
import TodoTable from './TodoTable';
```

- Remove the old table from the Todolist.js return statement and add TodoTable component.

```
<TodoTable todos={todos} />
```



Exercise 8

- Split your todo list application to separate components: Todolist and TodoTable

Simple Todolist

Add todo: —

Description: Date:

Date	Description	
12.11.2017	Go shopping	<input type="button" value="Delete"/>
23.11.2017	Meet friends in downtown	<input type="button" value="Delete"/>

React: 3rd party components

- React has a lot of 3rd party components that can be used in your own application
- One good source to find components is [js.coach](https://www.js.coach/)
- Let's now use react-table component in our Todolist application (<https://github.com/react-tools/react-table>)
- react-table component provides sorting, filtering, paging etc.
- Go to cmd/terminal and install react-table component

```
$ npm install --save react-table
```



React: 3rd party components

- Import component and stylesheet to TodoList component

```
import ReactTable from 'react-table';
```

```
import 'react-table/react-table.css';
```

- Define the columns inside render method

```
const columns = [{  
  Header: 'Date',  
  accessor: 'date' // String-based value accessors!  
}, {  
  Header: 'Description',  
  accessor: 'description',  
}]
```



React: 3rd party components

- Return ReactTable from the render method

```
return (  
  <div className="App">  
    <ReactTable data={this.props.todos}  
      columns={columns} sortable='true'  
      defaultPageSize='10' />  
  </div>  
);
```

- Now we have a responsive table with sorting and paging





Haaga-Helia

React: 3rd party components

Simple Todolist

Description:

Date:

Date	Description
4.8.2017	Go swimming
6.8.2017	Wash the car
21.8.2017	Movie with friends

Page of 1



Exercise 9

- Use todolist from the exercise 7. Use react-table component in the todolist table.

Hints: See how to render other elements to table cell from

<https://react-table.js.org/#/story/cell-renderers-custom-components>

Use Cell renderer and current row number = row.index

Simple Todolist

Add todo:

Description:

Date:

Date	Description	
1.3.2019	Meet friends	<input type="button" value="Delete"/>
7.3.2019	Go to library	<input type="button" value="Delete"/>
23.3.2019	Go to dinner	<input type="button" value="Delete"/>