Haaga-Helia

# React basics

Juha Hinkula
github: juhahinkula

Haaga-Helia

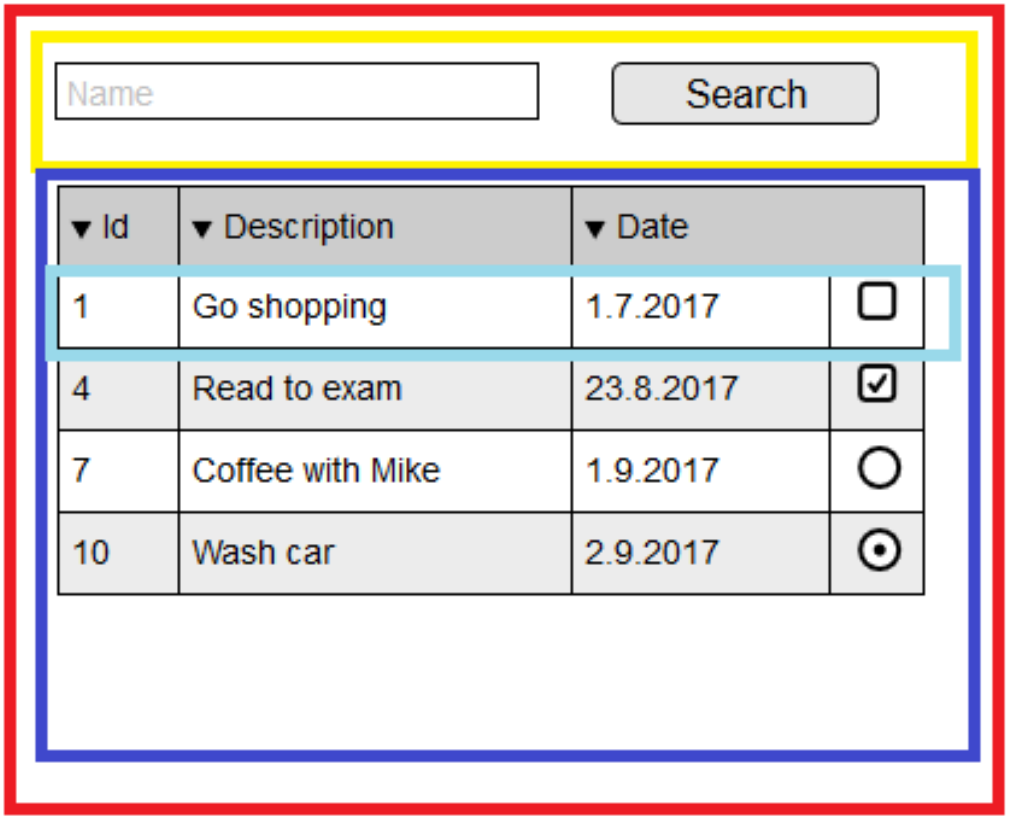
# React

- 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

Haaga-Helia

# React

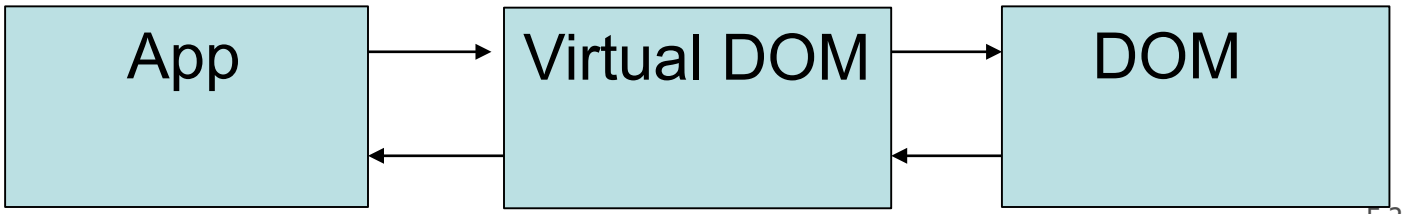
Red: Root component
Yellow: Search bar component
Blue: Table component
Light blue: Table row component



Haaga-Helia

# 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 virual DOM which is then compared to current DOM. Only changed parts of the DOM are updated.
- React also batches DOM manipulations



Haaga-Helia

# ECMAScript 6 (ES6)

- This material, like all recent React tutorials, uses the ES6 syntax which 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
  }
}
```

Haaga-Helia

# ECMAScript 6 (ES6)

- Some examples:
  - Inheritance

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

Haaga-Helia

# 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;
}
```

Haaga-Helia

# ECMAScript 6 (ES6)

- Some examples:
  - Constants

```
const PI = 3.141593
```

  - String interpolation

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

Haaga-Helia

# 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.

Haaga-Helia

# 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'));
```

Haaga-Helia

# 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'));
```

Haaga-Helia

# Render

- React component render() function returns elements that are going to 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)

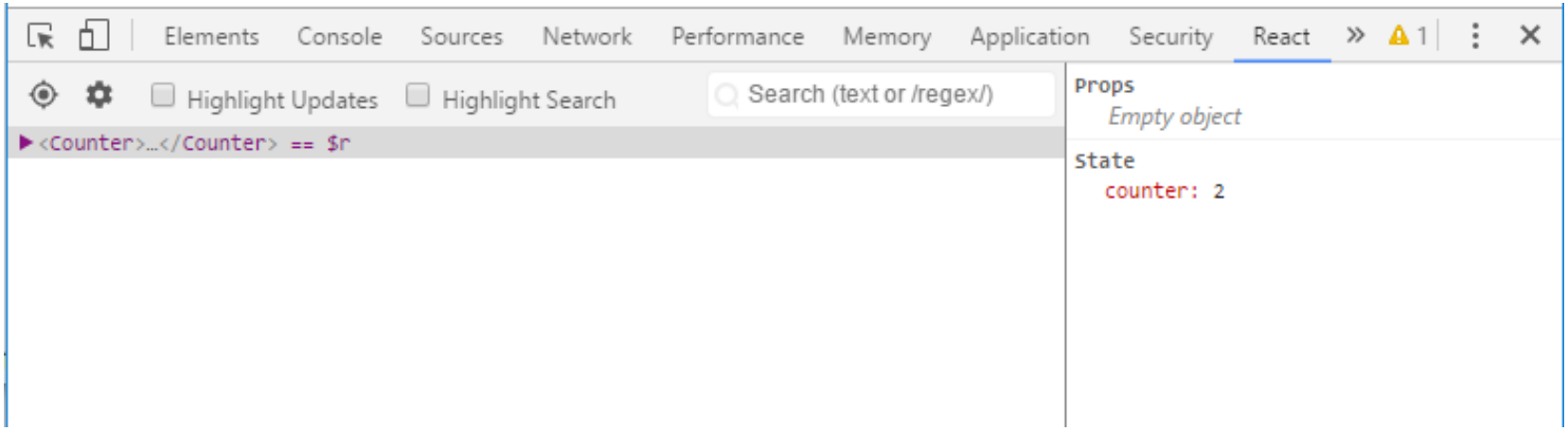
```
render() {
  return {
    <div>
      <h1>This is main header</h1>
      <p>Main text paragraph</p>
    </div>
  }
}
```

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

Haaga-Helia

# React Developer Tools

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



Haaga-Helia

# 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

Haaga-Helia

# React state

- State is initialized in the constructor

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

Haaga-Helia

# React state

- State is always changed by using `setState` method

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

Haaga-Helia

# 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:

Haaga-Helia

# 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}
});
```

Haaga-Helia

# 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>
    );
  }}
```

Haaga-Helia

# React JSX

- JSX is javascript syntax extension which is recommened to use with React
- In counter example we had JSX expression

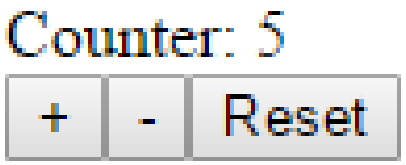
```
<div>Counter: {this.state.counter}</div>
```

- We can embed javascript to JSX by wrapping it in curly braces
- After compilation, JSX expressions becomes regular JavaScript objects

Haaga-Helia

# Exercise 2

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



Haaga-Helia

# 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'));
```

Haaga-Helia

# 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

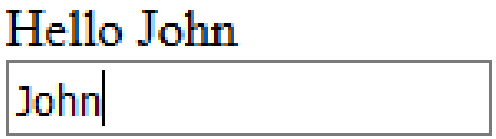
```
inputChanged = (event) => {
  this.setState({name: event.target.value});
};
```

# 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>
    );
  }
}
```



Haaga-Helia

# 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
- componentWillUnmount()
  - Called before component is unmounted

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

Haaga-Helia

# 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>
   );
  }
}
```

Haaga-Helia

# 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);
  }
}
```

Haaga-Helia

# 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
});
```

Haaga-Helia

# 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

Haaga-Helia

# Networking example

- Response


```
{
  "copyright": "Alson Wong",
  "date": "2017-09-20",
  "explanation": "Most photographs don't adequately portray the magnifi
unparalleled.  The human eye can adapt to see coronal features and exte
picture is a combination of forty exposures from one thousandth of a se
features of the total solar eclipse that occurred in August of 2017.  C
and magnetic fields in the Sun's corona.  Looping prominences appear br
made out, illuminated by sunlight reflected from the dayside of the Ful
  "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"
}
```

Haaga-Helia

# Networking example

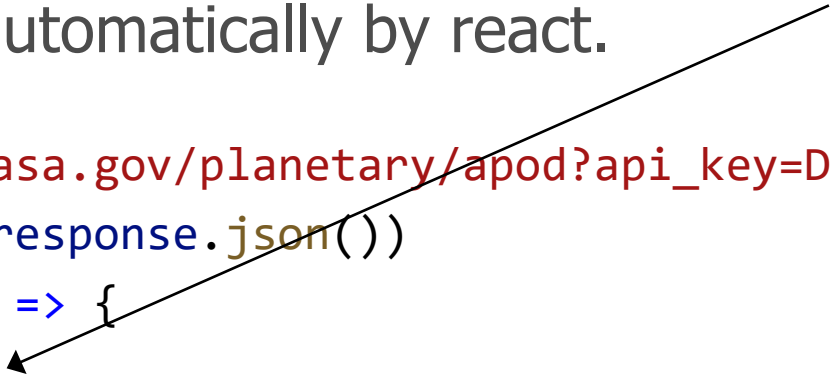
- We need states to get image url and explanation
- States will be initialized in constructor

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

Haaga-Helia

# 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

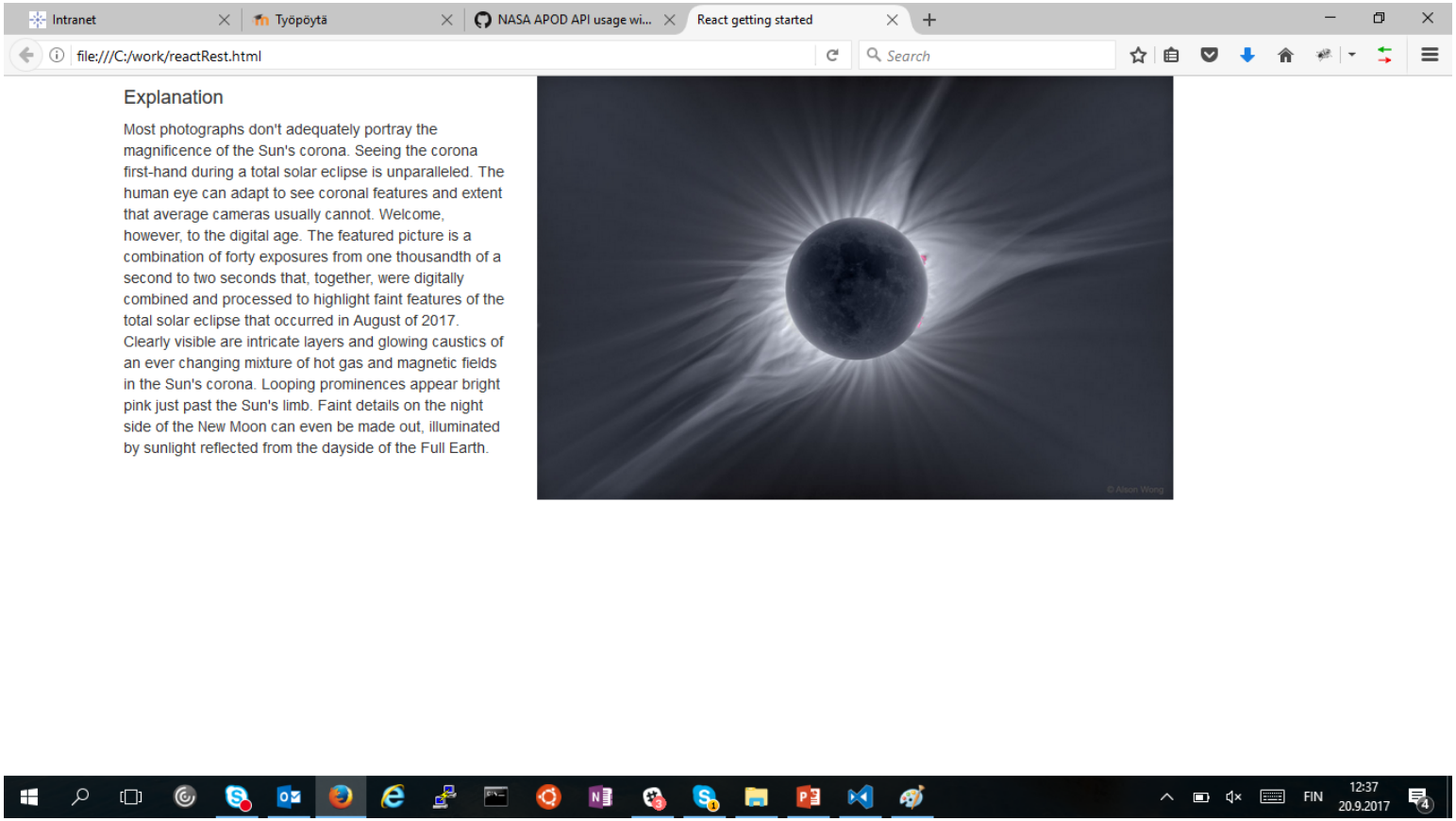
- Render method

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

Haaga-Helia

# Networking example

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



Haaga-Helia

# Exercise 3

- 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

Haaga-Helia

# 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

Haaga-Helia

# 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

Haaga-Helia

# 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>
  );
}
```

Haaga-Helia

# 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 |

Haaga-Helia

# **Exercise 4**: 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)

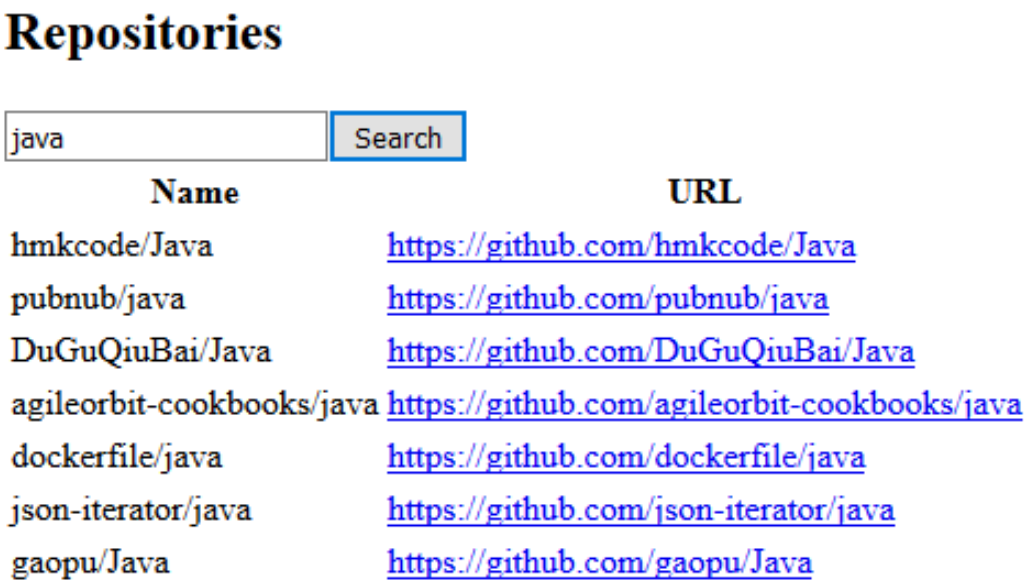
**Repositories**

| 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 |

Haaga-Helia

# Exercise 4: step 2

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



Haaga-Helia

# 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>
```



Haaga-Helia

# 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>
```

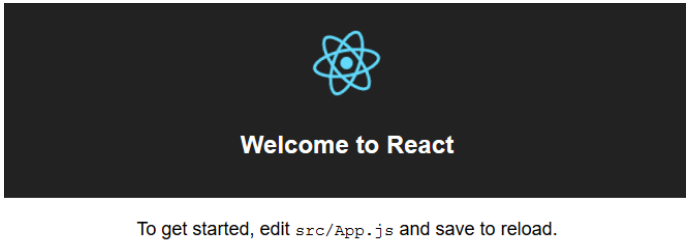
Haaga-Helia

# 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
```

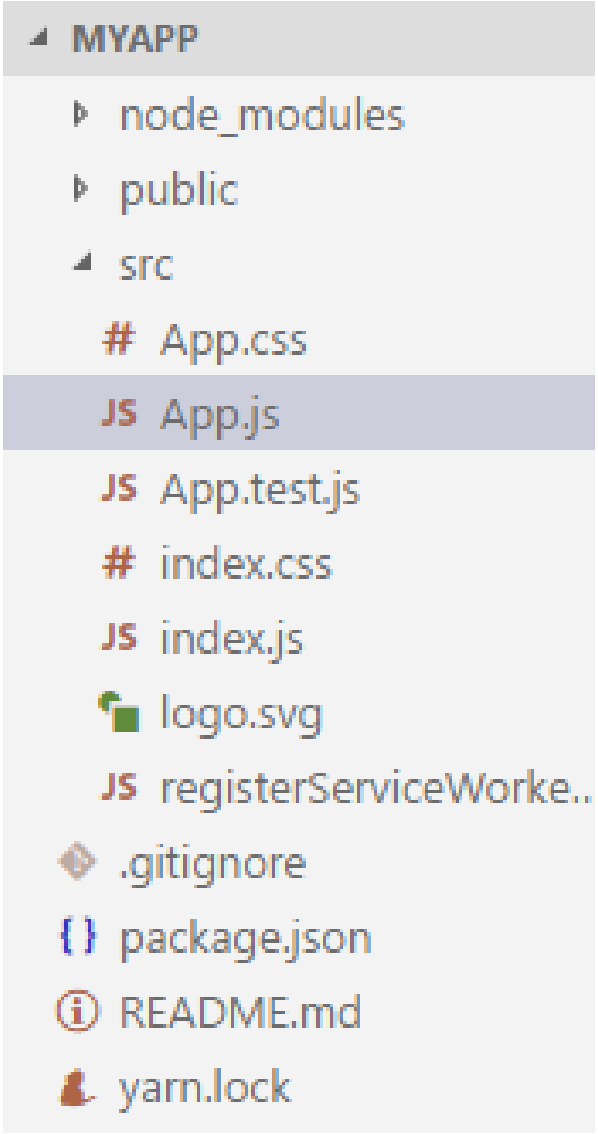
- Navigate to http://localhost:3000



Haaga-Helia

# 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



Haaga-Helia

# 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, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
      <div className="App-header">
        <img src={logo} className="App-
        logo" alt="logo" />
        <h2>Welcome to React</h2>
      </div>
      <p className="App-intro"></p>
      </div>
    );
  }
}

export default App;
```

Haaga-Helia

# 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 registerServiceWorker from './registerServiceWorker';

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

Haaga-Helia

# React: Todolist example

- Todolist example has only one field called ’description’.
- We need one state for description and one array state for all todos

```
constructor(props) {
  super(props);
  this.state = {description: '', todos: []}
}
```

Haaga-Helia

# React: Todolist example

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

**...inside render method**

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

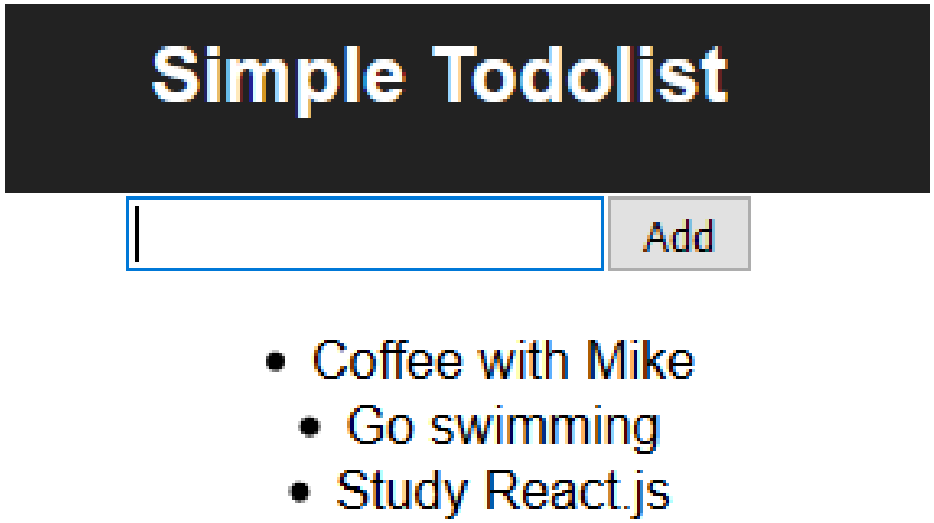
```
inputChanged = (event) => {
  this.setState({description: event.target.value});
}

addTodo = (event) => {
  event.preventDefault();
  this.setState({
    todos: [...this.state.todos, this.state.description]
  });
}
```

Haaga-Helia

# React: Todolist example

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



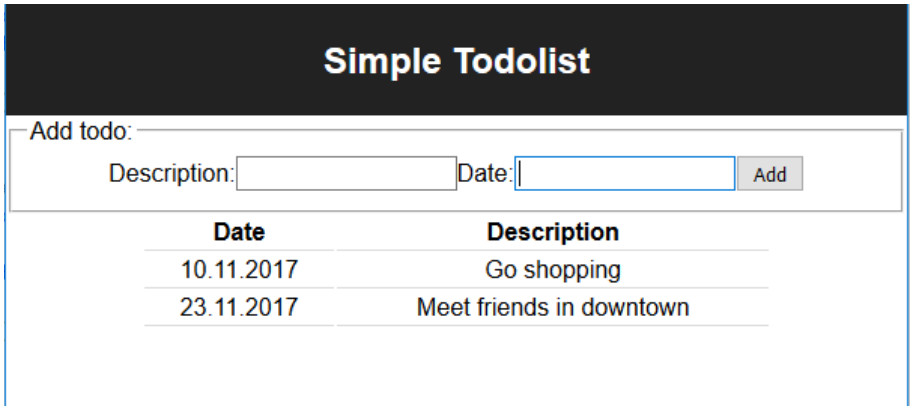
Haaga-Helia

# Exercise 5

- Add new date column to todolist example

  Tips:

  - Add new state and input field (use name attributes)
  - addTodo method: insert todo objects inside todos array
- Change list to table element and show both columns in the table (create your own styling)

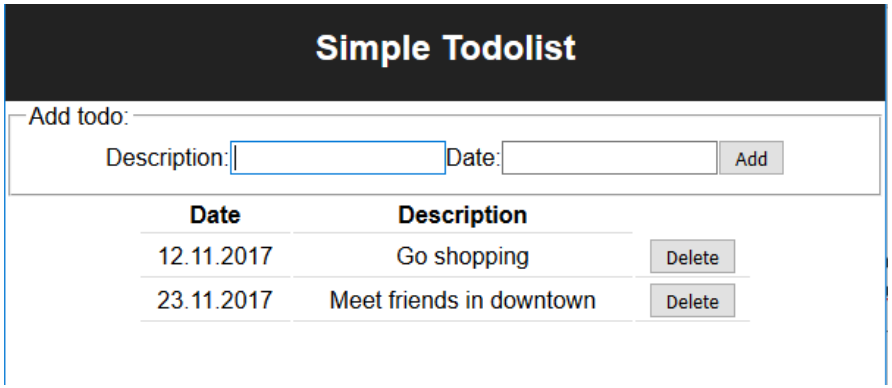


Haaga-Helia

# Exercise 6

- 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)
```



Haaga-Helia

# 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 to 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;
```

Haaga-Helia

# 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>
);
```

Haaga-Helia

# React: Separate components

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

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

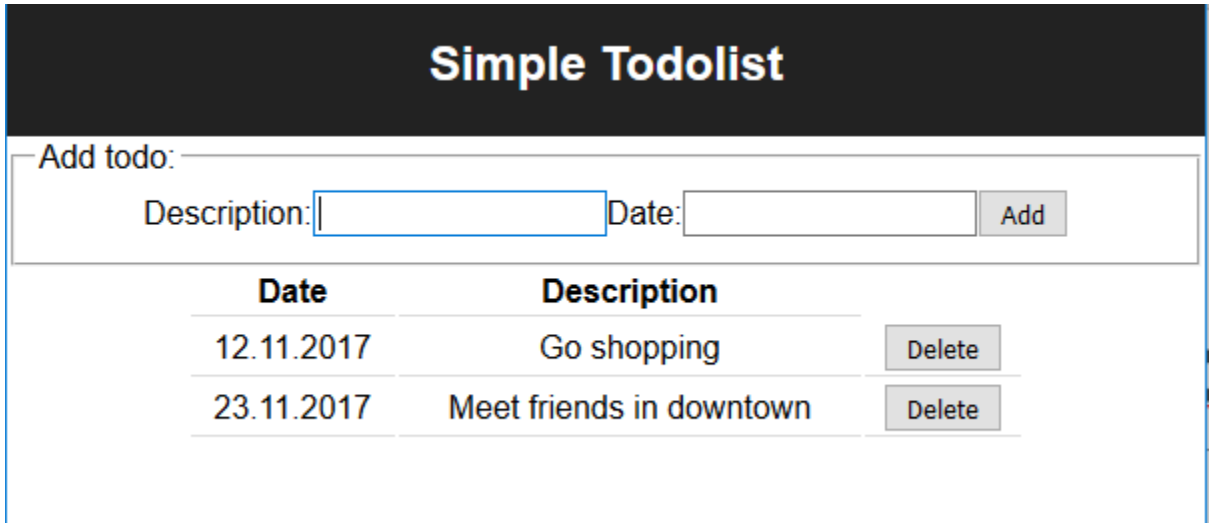
- Remove the old table from the App.js render method and add TodoTable component.

```
<TodoTable todos={this.state.todos} />
```

Haaga-Helia

# Exercise 7

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



Haaga-Helia

# 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
- 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
```

Haaga-Helia

# 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',
}]
```

Haaga-Helia

# React: 3rd party components

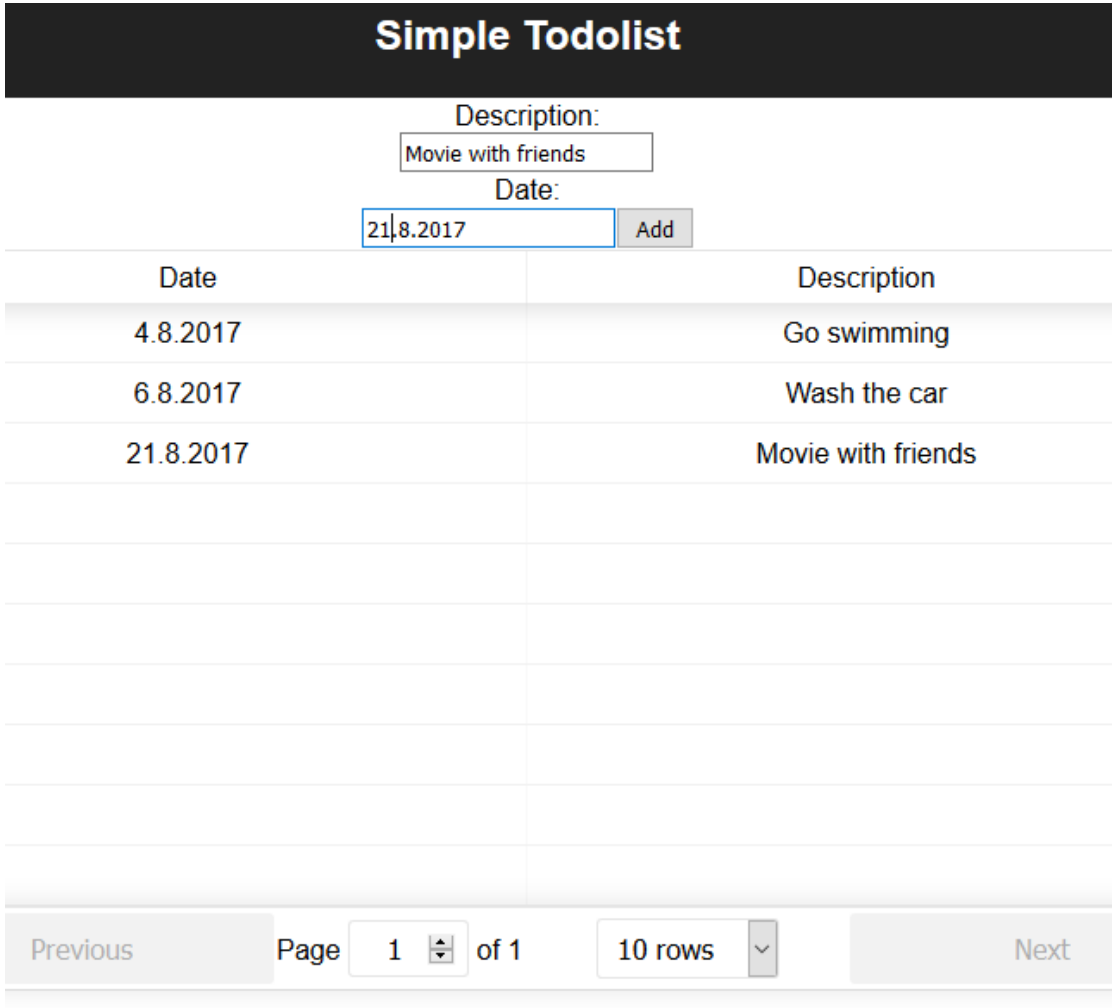
- Return ReactTable from the render method

```
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



Haaga-Helia

# Exercise 8

- 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

Row number = row.index