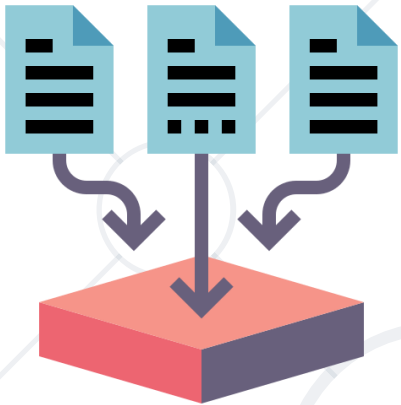


React Components – Deep Dive

Lists and Keys, Component Lifecycle, CSS Modules



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

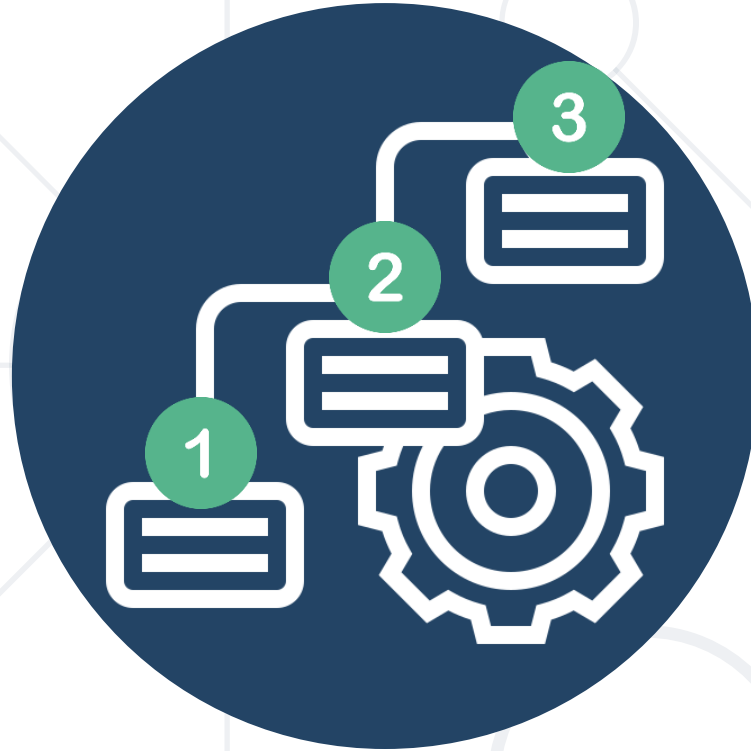
Table of Contents

1. Lists & Keys
2. Component Lifecycle
3. useEffect Hook
4. CSS Modules
5. Fetching Data



sli.do

#js-web



Lists and Keys

Identify Items, Reconciliation

Lists and Keys

- Using **map()** we can build collections of elements and include them in JSX using **{}**
- Keys should be given to the **elements inside the array** to give the elements a **stable identity**
- Keys help React identify which items have **changed**, are **added**, or are **removed**



- Using **map()** to take an array of numbers and double their values

```
const numbers = [1, 2, 3, 4, 5];  
const doubled = numbers.map((number) => number * 2);  
console.log(doubled); // [2, 4, 6, 8, 10]
```

- Rendering Multiple Components

```
const numbers = [1, 2, 3, 4, 5];  
const listItems = numbers.map((number) =>  
  <li>{number}</li>  
);
```

- 1
- 2
- 3
- 4
- 5

- Basic List Component looks like

```
function NumberList(props) {  
  const numbers = props.numbers;  
  const listItems = numbers.map((number) =>  
    <li>{number}</li>  
  );  
  return (  
    <ul>{listItems}</ul>  
  );  
}
```

- You can build **collections** of elements and include them in **JSX** using **{}**

```
function NumberList(props) {  
  const numbers = props.numbers;  
  const listItems = numbers.map((number) =>  
    <li>{number}</li>  
  );  
  return <ul>{listItems}</ul>;  
}
```

- Usually lists are rendered inside a **component**

- When you render an array of elements, React needs a **key prop** to identify elements for optimization purposes
 - If they don't have it, you will get

```
! Warning: Each child in a list should have a unique "key" prop.  
  
Check the render method of `App`. See https://fb.me/react-warning-keys for more information.  
    in person (at App.js:42)  
    in App (at src/index.js:7)
```

- It won't stop your work

- The best way to pick a key is to use a **string** that **uniquely identifies** a list item among its siblings
- Most often you would use **ID**'s from your data as keys

```
const todoItems = todos.map((todo) =>  
  <li key={todo.id}>  
    {todo.text}  
  </li>  
);
```

- Keys only make sense in the context of the surrounding array

```
function NumberList(props) {  
  const numbers = props.numbers;  
  const listItems = numbers.map((number) =>  
    <ListItem key={number.toString()} value={number} />  
  );  
  return (  
    <ul>  
      {listItems}  
    </ul>  
  );  
}
```

Keep the key on the list item

```
function ListItem(props) {  
  return <li>{props.value}</li>;  
}
```

- Don't use indexes for keys if the order **may change**
- Keys serve as a hint to React, but they **don't get passed** to your component
 - If you need the same value, pass it explicitly as prop with a different name

```
const content = posts.map((post) =>  
  <Post  
    key={post.id} id={post.id} title={post.title}  
  />  
);
```

- Keys **don't** need to be **globally unique** (only among their siblings)

```
const sidebar = (  
  <ul>  
    {props.posts.map((post) =>  
      <li key={post.id}>  
        {post.title}  
      </li>  
    )}  
  </ul>  
);
```

```
const posts = [  
  {id: 1, title: '...', content: '...'},  
  {id: 2, title: '...', content: '...'}];
```

```
const content = props.posts.map((post) =>  
  <div key={post.id}>  
    <h3>{post.title}</h3>  
    <p>{post.content}</p>  
  </div>);
```



Component Lifecycle

Component Lifecycle

- A component has "**lifecycle methods**" that can be overridden to run code at times in the process
- A component has **3 lifecycle** phases
 - **Mounting**
 - **Updating**
 - **Unmounting**



- **Mounting** - where the component and all its children are mounted (created and inserted to the DOM)
- **Updating** - component is re-rendered because changes are made to its props or state
- **Unmounting** - occurs when a component instance is unmounted (removed from the DOM)



Effect Hook

Effect Hook

- You most likely perform: **data fetching, subscriptions** or **manually changing the DOM**
 - Operations like these are called **side effects**
 - They can **affect** other components and can't be done during the rendering
- **useEffect** hook adds the ability to perform side effects from a function component



- **useEffect** hook serves the same purpose as
 - **componentDidMount**
 - **componentDidUpdate**
 - **componentWillUnmount**
- But they are bundled into a single API

```
Import { useEffect } from 'react';
```

- **useEffect** hook accepts a function that contains imperative, possibly effectful code
 - That function will run **after the render** is committed to the screen
- By default effects run after **every completed render**
 - But you can choose to fire them only when certain value have changed

```
import React, { useState, useEffect } from 'react';

const counter = () {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate
  useEffect(() => {
    document.title = `The counter reached: ${count} times`;
  });

  // ...
}
```

- When you call **useEffect** you're telling React to run your "effect" function after flushing changes to the DOM
- Effects are declared inside the component so they have access to its **props** and **state**
- Effects may also optionally specify how to "clean up" after them by **returning a function**

- Often, effects create resources that need to be **cleaned up** before the component leaves the screen
 - To do this, the function passed to **useEffect** may return a **clean-up function**

```
useEffect(() => {  
  const subscription = props.source.subscribe();  
  return () => {  
    // Clean up the subscription  
    subscription.unsubscribe();  
  };  
});
```



Effect Hook Demo



CSS Modules

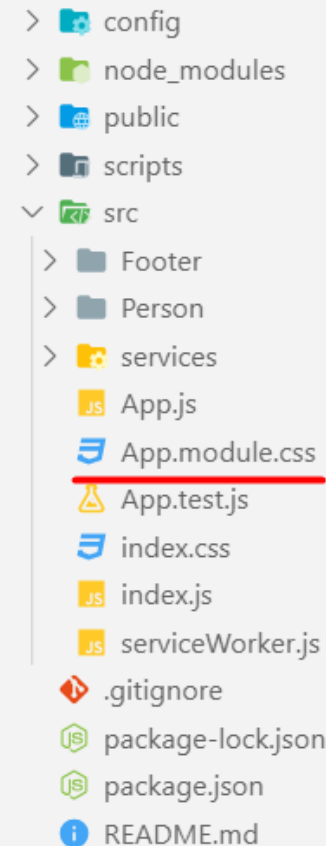
CSS Modules

- **CSS files** in which all class names and animation names are scoped locally by default
- All **URLs** and **imports** are relative
- Importing CSS Module from a JS Module
 - Exports an **object** with all mapping from local names to global names



- React supports CSS Modules alongside regular stylesheet using the **[name].module.css** file naming convention

```
.App {  
  text-align: center;  
}  
  
.btn {  
  background-color: green;  
  color: white;  
  border-radius: 15px;  
  margin: 2%;  
  padding: 0.5%;  
  font-size: 24px;  
  cursor: pointer;  
}
```



```
> config  
> node_modules  
> public  
> scripts  
▼ src  
  > Footer  
  > Person  
  > services  
    App.js  
    App.module.css  
    App.test.js  
    index.css  
    index.js  
    serviceWorker.js  
  .gitignore  
  package-lock.json  
  package.json  
  README.md
```

- CSS Modules let you use the same CSS class name in different file without worrying about naming clashes

```
.error {  
  background-color: white;  
  color: red;  
}
```

CSS File called Button.module.css

```
import React, { Component } from 'react';  
import styles from './Button.module.css';  
  
class Button extends Component {  
  render() {  
    return <button className={styles.error}>Error Button</button>;  
  }  
}
```

Importing all styles

Using error class from the css file



Using Fetch API

Fetching Remote Data

Fetch API

The **Fetch API** provides an interface for accessing and manipulating **requests** and **responses**

- **fetch()** function which provides easy way to fetch resources asynchronously
- Functionality like this was previously achieved using **XMLHttpRequest**



- **fetch()** takes one mandatory argument (the path to the resource you want to fetch)
 - Second argument is optionally (init options - object)
- **returns** a **promise**
- Once **response** is **retrieved**, there are several **methods** that defines what and how should be handled

■ Fetch API with then/catch example

```
fetch('https://api.github.com/users/k1r1L')  
  .then((response) => response.json())  
  .then((myJson) => console.log(myJson))  
  .catch((myErr) => console.error(myErr));
```

```
{...}  
  avatar_url: "https://avatars0.githubusercontent.com/u/13466012?v=4"  
  bio: "Student at Faculty of Mathematics & Informatics (FMI Sofia University) and SoftUni.\r\nExperience in C#, Java, JavaScript."  
  blog: ""  
  company: null  
  created_at: "2015-07-23T09:59:07Z"  
  email: null  
  events_url: "https://api.github.com/users/k1r1L/events{/privacy}"  
  followers: 83  
  followers_url: "https://api.github.com/users/k1r1L/followers"  
  following: 13  
  following_url: "https://api.github.com/users/k1r1L/following{/other_user}"  
  gists_url: "https://api.github.com/users/k1r1L/gists{/gist_id}"  
  gravatar_id: ""  
  hireable: null  
  html_url: "https://github.com/k1r1L"  
  id: 13466012  
  location: "Sofia, Bulgaria"  
  login: "k1r1L"  
  name: "Kiril Kirilov"  
  node_id: "MDQ6VXNlcjEzNDY2MDEy"  
  organizations_url: "https://api.github.com/users/k1r1L/orgs"  
  public_gists: 0  
  public_repos: 22  
  received_events_url: "https://api.github.com/users/k1r1L/received_events"  
  repos_url: "https://api.github.com/users/k1r1L/repos"  
  site_admin: false  
  starred_url: "https://api.github.com/users/k1r1L/starred{/owner}/{repo}"  
  subscriptions_url: "https://api.github.com/users/k1r1L/subscriptions"  
  type: "User"  
  updated_at: "2019-10-01T08:26:54Z"  
  url: "https://api.github.com/users/k1r1L"  
  <prototype>: Object { ... }
```


■ Fetch API with async/await example

```
(async () => {  
  try {  
    const response = await fetch('https://api.github.com/users/k1r1L');  
    const myJson = await response.json();  
    console.log(myJson);  
  } catch (myErr) {  
    console.error(myErr);  
  }  
})();
```

```
{...}  
  avatar_url: "https://avatars0.githubusercontent.com/u/13466012?v=4"  
  bio: "Student at Faculty of Mathematics & Informatics (FMI Sofia University) and SoftUni.\r\nExperience in C#, Java, JavaScript."  
  blog: ""  
  company: null  
  created_at: "2015-07-23T09:59:07Z"  
  email: null  
  events_url: "https://api.github.com/users/k1r1L/events{/privacy}"  
  followers: 83  
  followers_url: "https://api.github.com/users/k1r1L/followers"  
  following: 13  
  following_url: "https://api.github.com/users/k1r1L/following{/other_user}"  
  gists_url: "https://api.github.com/users/k1r1L/gists{/gist_id}"  
  gravatar_id: ""  
  hireable: null  
  html_url: "https://github.com/k1r1L"  
  id: 13466012  
  location: "Sofia, Bulgaria"  
  login: "k1r1L"  
  name: "Kiril Kirilov"  
  node_id: "MDQ6VXNlcjEzNDY2MDEy"  
  organizations_url: "https://api.github.com/users/k1r1L/orgs"  
  public_gists: 0  
  public_repos: 22  
  received_events_url: "https://api.github.com/users/k1r1L/received_events"  
  repos_url: "https://api.github.com/users/k1r1L/repos"  
  site_admin: false  
  starred_url: "https://api.github.com/users/k1r1L/starred{/owner}/{repo}"  
  subscriptions_url: "https://api.github.com/users/k1r1L/subscriptions"  
  type: "User"  
  updated_at: "2019-10-01T08:26:54Z"  
  url: "https://api.github.com/users/k1r1L"  
  <prototype>: Object { ... }
```

- The basic idea is to **isolate** the concern of fetching data inside components
 - Fetching data logic should be separated as **service**

```
const apiUrl = '...';  
  
export const getData = () => {  
  return fetch(apiUrl)  
    .then(res => res.json())  
    .then(data => data.results)  
    .catch(error => console.error(error))  
};
```

Import the **useState** and **useEffect** hooks

Import the service

Using the **useState** hook

Using the **useEffect** hook

Using the service

```
import { useState, useEffect } from 'react';
import { getData } from '../services/fetching-data-service';

function App() {
  const [state, setState] = useState({ data: [],
    isLoading: false });

  useEffect(() => {
    setState((state) => ({ ...state, isLoading:
      true }));
    getData().then((data) => {
      setState((state) => ({ ...state, data,
        isLoading: false }));
    });
  }, []);
}
```

Return the HTML
structure

```
return (  
  <div className='container'>  
    {state.data.map((x) => (<p key={x.id}>  
      {x.text}</p>))}  
  </div>  
);  
}  
  
export default App;
```

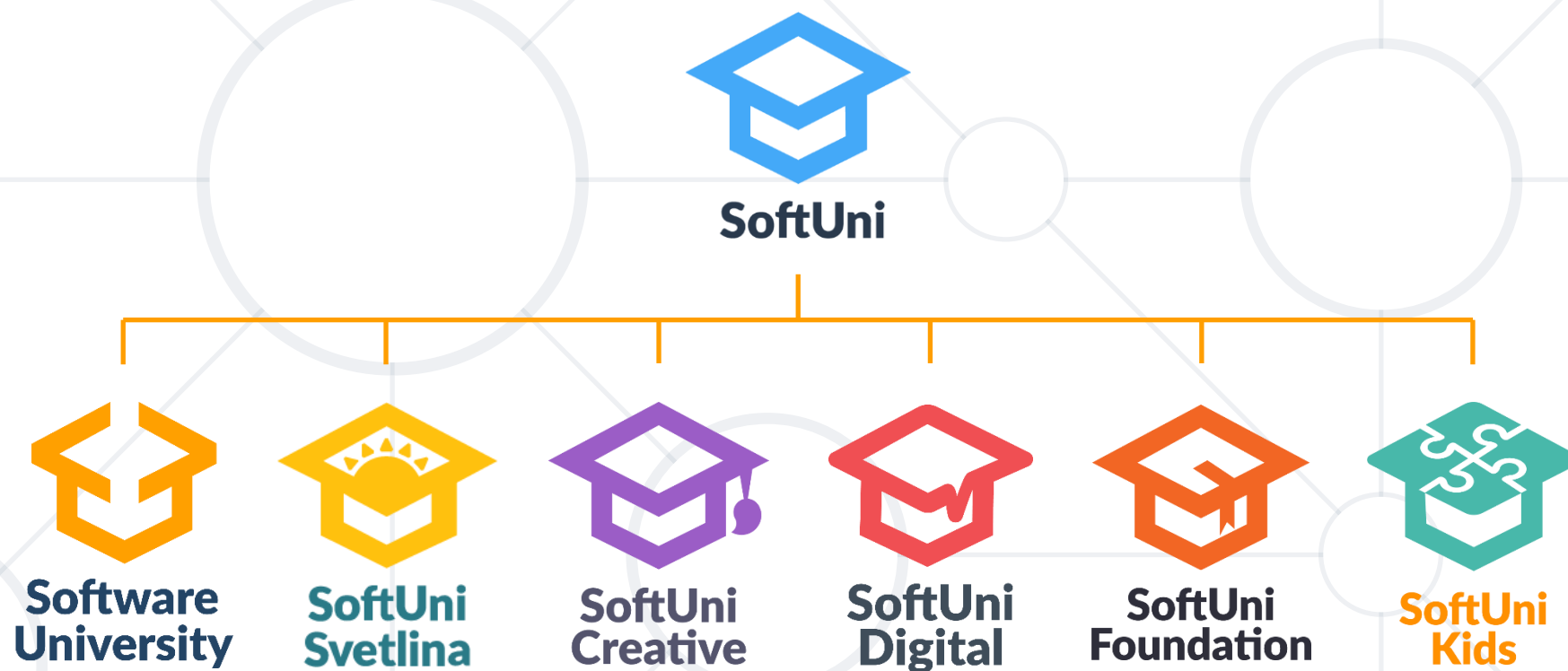


Demo

- Lists and Keys
 - Collection of components with **unique key**
- Component Lifecycle
 - **Mounting, Update** and **Unmounting**
- useEffect **HOOK**
- CSS Modules
- Using the **Fetch API**



Questions?



SoftUni Diamond Partners

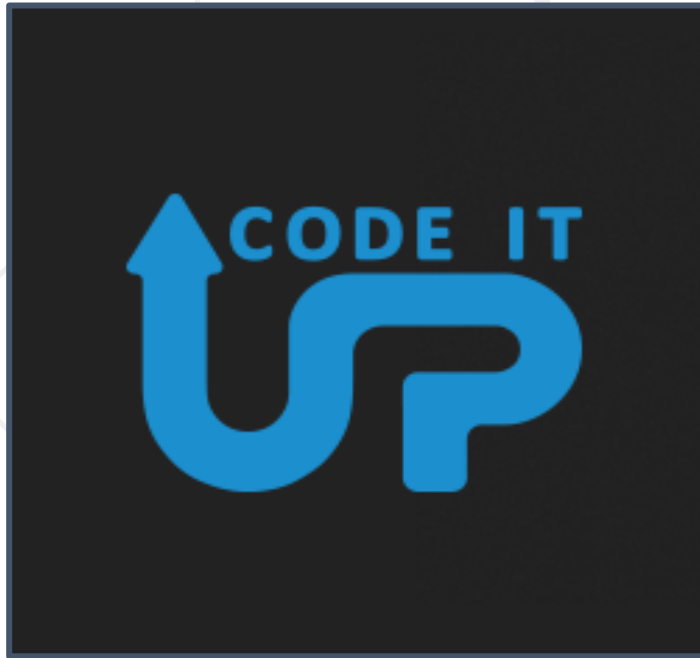


SCHWARZ



**SUPER
HOSTING
.BG**





VIRTUAL RACING SCHOOL



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

