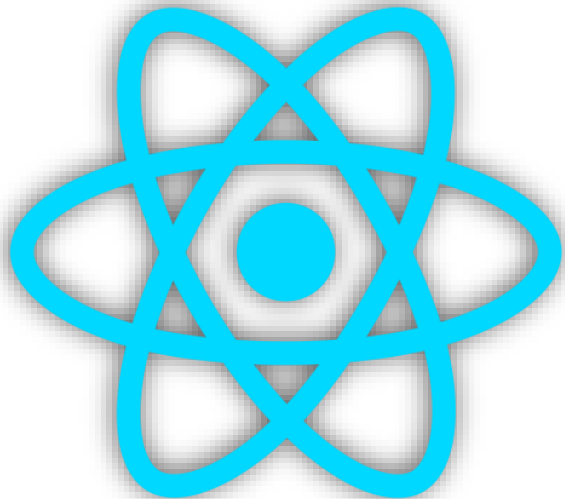


FRONT-END *Academy*

by Present Connection



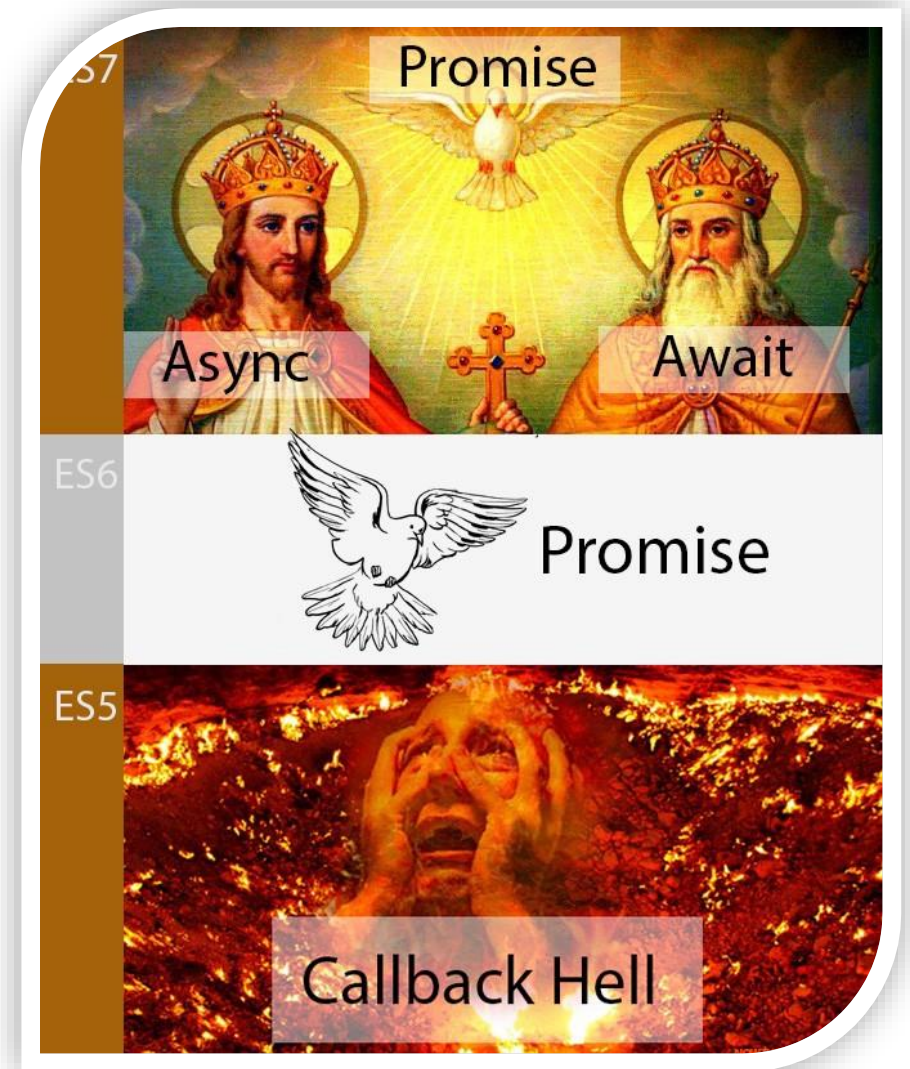
Callback hell

```
function hell(win) {  
  // for listener purpose  
  return function() {  
    loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {  
      loadLink(win, REMOTE_SRC+'/lib/async.js', function() {  
        loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {  
          loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {  
            loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {  
              loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {  
                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {  
                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {  
                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {  
                      async.eachSeries(SCRIPTS, function(src, callback) {  
                        loadScript(win, BASE_URL+src, callback);  
                      });  
                    });  
                  });  
                });  
              });  
            });  
          });  
        });  
      });  
    });  
  });  
}
```



async await

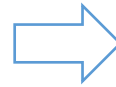
- ◆ Atsirado ES7
- ◆ Lengvesnis/grāžesnis būdas dirbti su Promises
- ◆ Jeigu funkcija yra async ji visada grāžina Promise



async

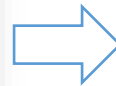
```
async function startAcademy() {  
  return 'Started!';  
}
```

```
console.log(startAcademy());
```



```
Promise { 'Started!' }
```

```
startAcademy().then( (result)=> {  
  console.log(result)  
});
```



```
Started!
```

async


```
async function startAcademy() {  
  return 'Started!';  
}
```

=


```
function startAcademy() {  
  return Promise.resolve('Started!');  
}
```

await

- ◆ Laukia kol Promise grąžins rezultatą
- ◆ Veikia tik async funkcijos viduje



```
await somePromise;  
async function myFunction() {  
  |   await somePromise;  
}
```



```
async function myFunction() {  
  |   await somePromise;  
}
```

await

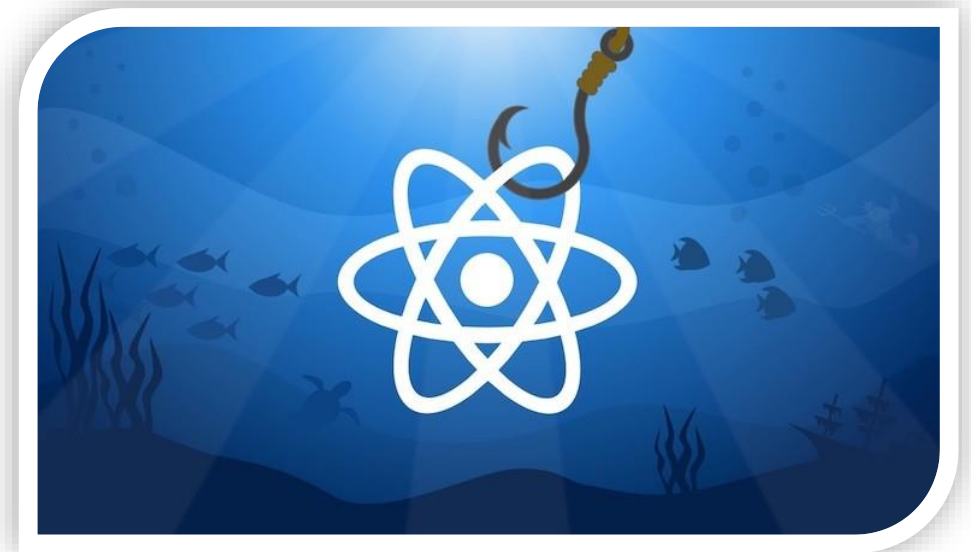
```
function startAcademy() {  
  let waitForStudents = new Promise((resolve) => {  
    setTimeout(() => resolve("we are here!"), 1000)  
  });  
  
  waitForStudents.then(()=>{  
    console.log('Started!');  
  });  
}
```

```
async function startAcademy() {  
  let waitForStudents = new Promise((resolve) => {  
    setTimeout(() => resolve("we are here!"), 1000)  
  });  
  
  let students = await waitForStudents;  
  console.log('Started!');  
}
```

Hooks

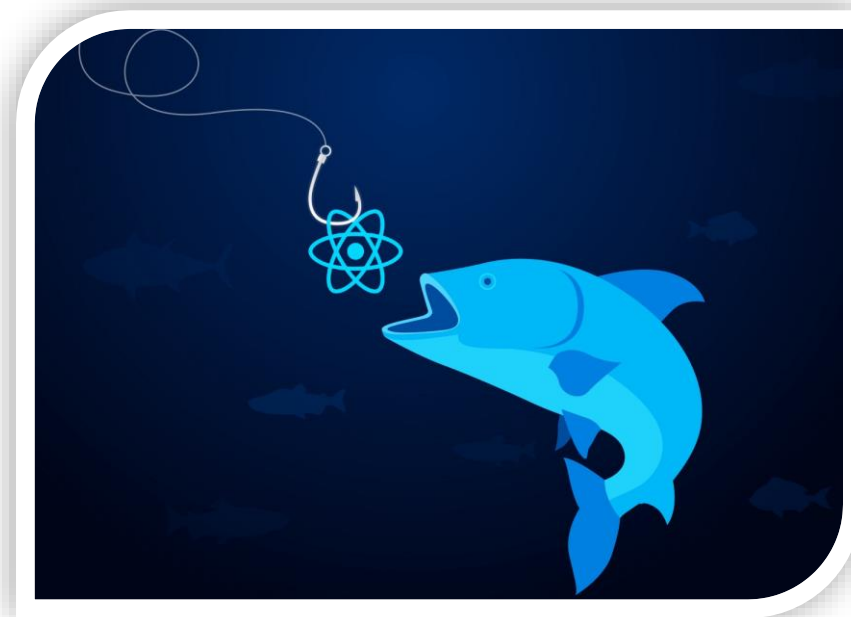
<https://reactjs.org/docs/hooks-overview.html>

- ◆ Funkcijos, kurios leidžia „užsikabinti“ už state ir lifecycle naudojantis tik funkciniais komponentais
- ◆ Neveikia class komponentuose
- ◆ Nėra jokių planų, kad klasių komponentai bus panaikinti
- ◆ Yra kviečiami tik komponentuose, viena išimtis kai kuriate savo hook.



Kodėl naudoti Hooks?

- ◆ Naudojantis klasių komponentais labai sunku pernaudoti state logiką tarp jų, hooks leidžia ją pernaudoti nepakeičiant komponentų hierarchijos
- ◆ Kompleksinius komponentus labai sunku suprasti ir juos palaikyti
- ◆ Klasės painioja tiek žmones tiek kompiuterius.
 - ◆ Reikia žinoti kaip veikia this
 - ◆ Nepamiršti bind
 - ◆ Atskirtis tarp funkciniu ir klasių komponentų ne visada aiški
 - ◆ Klasės sunkiai minify'jinasi



state

- ◆ Saugo komponento duomenis kurie keičiasi
- ◆ State galima keisti komponento viduje
- ◆ Gali būti naudojama tiek class tiek funkciniam komponentuose
- ◆ Jeigu keičiame state komponentas automatiškai persipiešia


```
class AcademyWithState extends Component
{
  state = {
    number:0
  };

  render()
  {
    return <div>{this.state.number}</div>
  };
}
```


state

- ◆ Norėdami pakeisti state naudojame setState metodą
- ◆ Nemodifikuokime state tiesiogiai, nes komponentas nebus perpiešiamas

```
this.state.number = 5;
```




```
this.setState({ number: 5 });
```




state

- ◆ State yra asynchroninė
- ◆ Jeigu norime keisdami state pasiekti jos ansktesnę versiją. Naudojame funkcija, nes kitu atveju state dar gali būti neatsinaujinus



```
this.setState({  
  number: this.state.number + 1  
});
```



```
this.setState((prevState, props) =>({  
  number: prevState.number + 1  
}));
```

State hook

- ◆ Gražina porą: dabartinę reikšmę ir funkcija su kuria ją galima keisti
- ◆ Vienintelis argumentas initial state
- ◆ Galima naudoti daug kartų viename komponente
- ◆ Array destructing

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Effect hook

- ◆ Naudojamas side efektams: data fetching, DOM manipuliacija, subscription ir kita.
- ◆ Yra paleidžiami per kiekvieną render
- ◆ Padeda užsikabinti ant komponento lifecycle event'ų

```
// componentDidMount  
useEffect(() => {  
  |   getData();  
}, []);
```

Effect hook

```
// componentWillUnmount  
useEffect(() => {  
  |   return clean();  
}, []);
```

```
// componentWillReceiveProps  
useEffect(() => {  
  |   return getData();  
}, [props.property]);
```

```
// componentDidUpdate  
useEffect(() => {  
  |   return getData();  
});
```

Custom hook

- ◆ Funkcija
- ◆ Naudojami perpanaudoti logiką esančią komponentuose.
- ◆ Prasideda su use

Custom hook panaudojimas

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

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

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

function FriendListItem(props) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

Custom hook panaudojimas

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });

  return isOnline;
}
```

```
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

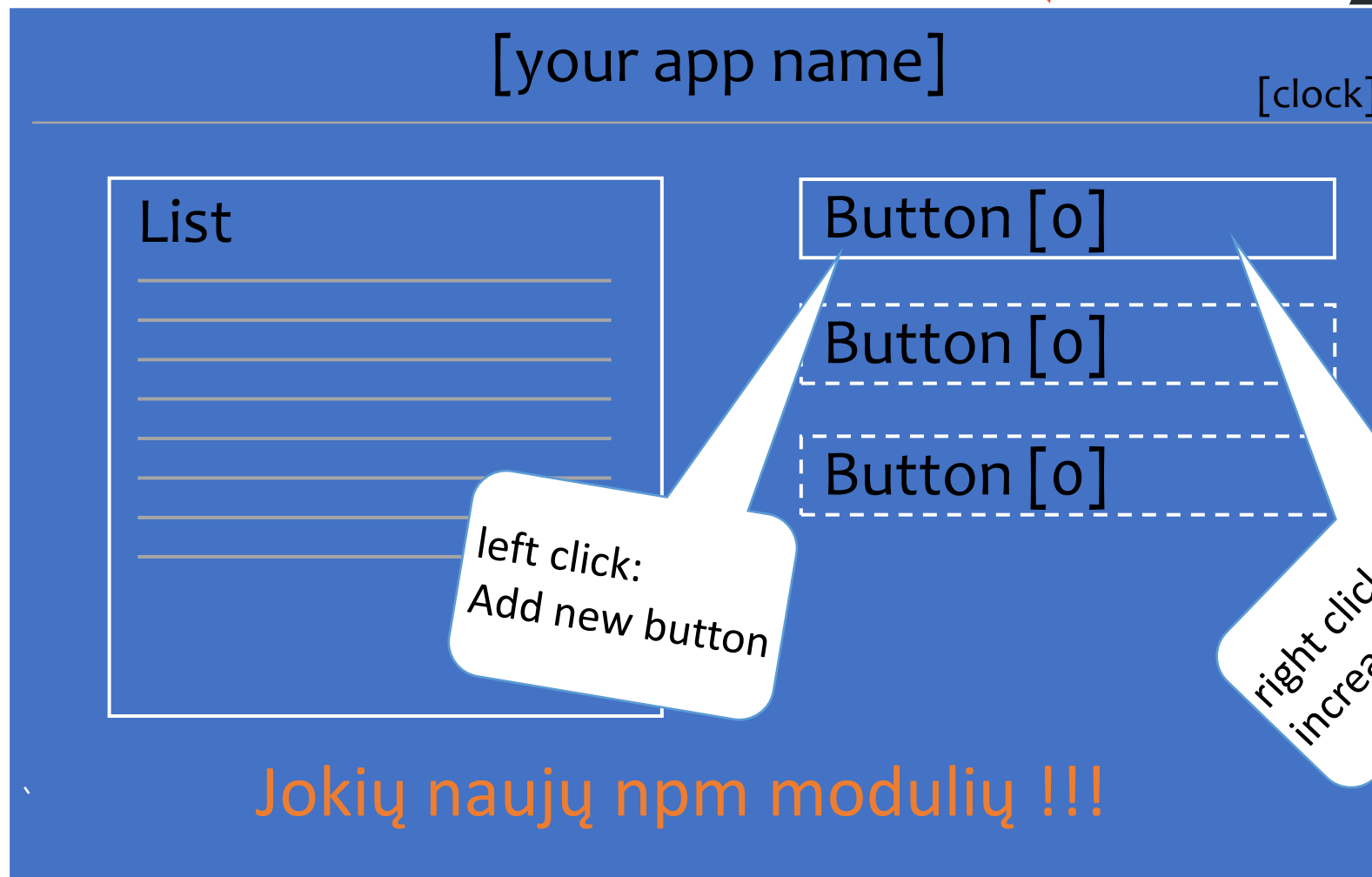
```
function FriendListItem(props) {
  const isOnline = useFriendStatus(props.friend.id);

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

DEMO



Workshop



Jokių naujų npm modulių !!!

Namų darbai

- ◆ Toliau dirbam prie projektų, pajungiam duomenis iš serverio pakurtiems puslapiams.

