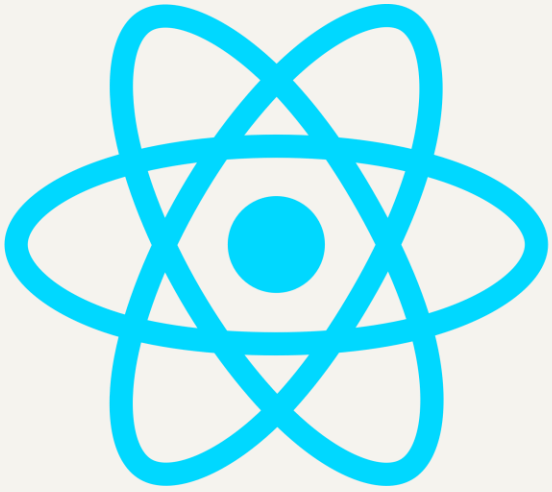


React

Material UI, viacstránkové aplikácie, fetch a Promise

UINF/PAZ1c

epizóda 11

Material UI

- Čo je  ?
  - Moderná knižnica na pekné štýlovanie HTML stránok (CSS + JSX)
  - https://mui.com/
  - Kolekcia už oštýlovaných komponentov
  - Podpora pre svetlý/tmavý režim
- Nainštalujeme ho do projektu
  - npm install @mui/material @emotion/react @emotion/styled @fontsource/roboto @mui/icons-material

Material UI

- Čo je  ?
  - Moderná knižnica na pekné štýlovanie HTML stránok (CSS + JSX)
  - https://mui.com/
- Nainštalujeme ho do projektu
  - npm install @mui/material @emotion/react @emotion/styled @fontsource/roboto @mui/icons-material

Zobrazíme si užívateľov v tabuľke

```
<TableContainer component={Paper}>
 <Table>
  <TableHead>
    <TableRow>
     <TableCell>ID</TableCell>
     ...
    </TableRow>
  </TableHead>
  <TableBody>
    {users.map((user) => (
     <TableRow onClick={() => handleRowClick(user)}>
      <TableCell>{user.id}</TableCell>
        ...
     </TableRow>
    ))}
  </TableBody>
 </Table>
</TableContainer>
```

Stiahnutie dát z REST servera

- Prehliadače majú funkciu *fetch*

```
fetch(`${restURL}/users`)
    .then(response => response.json())
    .then(data => setUsers(data as User[]))
    .catch(error => setError(new Error("Could not fetch users", {cause: error})));
```

- IO volania treba zabaľovať do "efektu"
  - Ináč sa fetch bude volať neustále (60 krát za sekundu)
  - useEffect sa volá napr. iba pri vytváraní komponentu alebo refreshi stránky
  - ak do poľa závislostí dáme nejakú premennú z useState, tak sa useEffect zavolá aj pri zmene stavu premennej

```
useEffect(() => {
   fetch(`${restURL}/users`)
      . . .
}, [])
```

Blokujúce úlohy

- Sieťové volania (IO) sú asi 100 tisíc krát pomalšie ako "normálne" volania => **blokujeme CPU**
- V GUI **neslobodno** blokovať CPU
  - Aplikácia sa bude sekať
  - Počas blokujúceho volania užívateľ nemôže nič robiť
- Java, C++: IO dáme do vlastného vlákna a nech sa stará OS
- JS/TS: Máme iba 1 vlákno a tak si (ne)blokovanie musí aplikácia riešiť sama

Kooperatívny multitasking

- IO funkcie vracajú Promise objekty (sľuby)
  - Obsahuje úlohu, ktorú OS/runtime vykonáva na pozadí
  - Runtime okrem vykonávania vášho programu kontroluje stav Promise-ov
  - Ak je sľub splnený (fulfilled), tak runtime spustí then callback, ktorý môže vracať nový Promise, atď.
  - Výnimky (rejected promise) môžeme odchytávať cez catch callback

```
fetch(`${restURL}/users`)
   .then(response => response.json())
   .then(data => setUsers(data as User[]))
   .catch(error => setError(new Error("Could not fetch users", {cause: error})));
```

  - Alternatívne môžeme pracovať s async-await, čo je imperatívny syntax cukor nad Promise-ami

```
try {
   const response = await fetch(`http://localhost:8080/users`)
   const data = await response.json()
   setUsers(data as User[])
} catch (error) {
   setError(new Error("Could not save user", {cause: error}))
}
```

Cross-Origin Resource Sharing

- Prehliadač umožňuje robiť sieťové volania iba na tú istú adresu/port z ktorého pochádza aplikácia
  - V našom prípade localhost:3000
- Ale REST server beží na localhost:8080
  - Teda fetch nebude fungovať
- CORS
  - Nastavíme REST server a "white-listujeme" adresu, kde beží webové UI (localhost:3000)
  - Browser si zistí, že server ho má vo white-liste
  - A hurá, všetko ide

Nastavenie CORS

- V konfiguračnej triede servera:

```
@Value("${cors.allowedOrigins}")
private String allowedOrigins;


@Bean
public WebMvcConfigurer corsConfigurer() {
 return new WebMvcConfigurer() {
   @Override
   public void addCorsMappings(CorsRegistry registry) {
    registry
      .addMapping("/**")
      .allowedOrigins(allowedOrigins.split(","))
      .allowedMethods("*");
   }
 };
}
```

- V application.yaml:

```
cors:
  allowedOrigins: ${CORS_ALLOWED_ORIGINS:http://localhost:3000,http://localhost:8080}
```

Mazanie entít

- Spravme si tlačítko na mazanie hneď v tabuľke

```
const handleDeleteClick = (userId: number | undefined) => {
   fetch(`${restURL}/users/${userId}`, {
      method: 'DELETE'
   }).then(() => setUsers(users.filter(user => user.id !== userId)))
      .catch(error => setError(new Error("Could not delete user", {cause: error})));
};
```

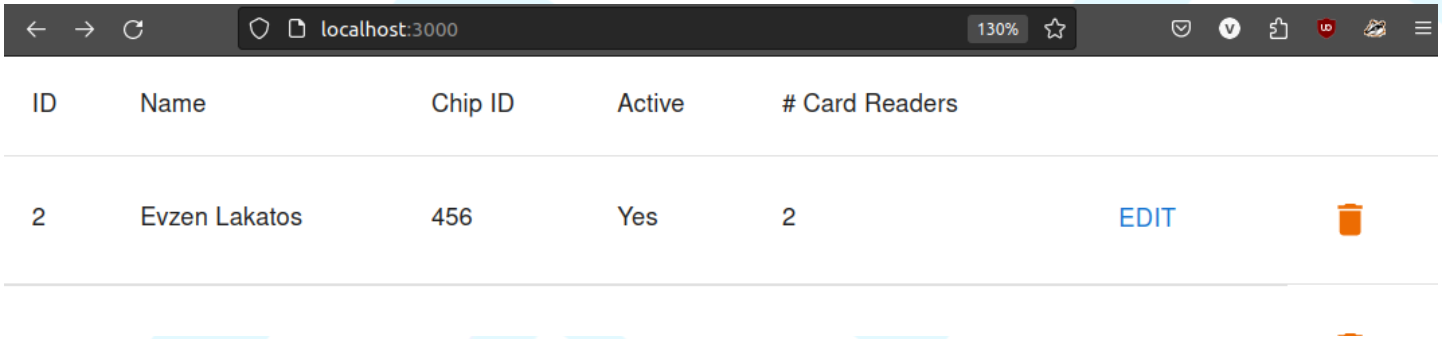
```
import DeleteIcon from '@mui/icons-material/Delete';
. . .

<TableCell>
  <IconButton onClick={() => handleDeleteClick(user.id)}
        color='warning'><DeleteIcon/></IconButton>
</TableCell>
```

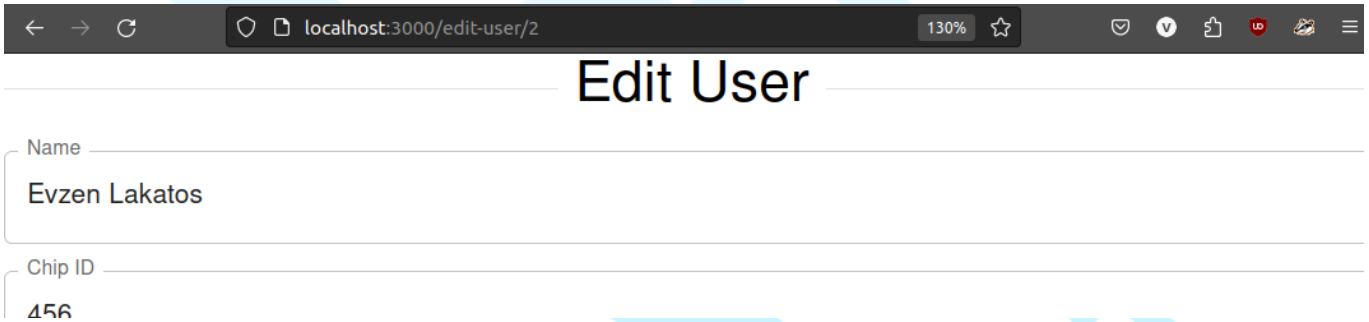
ID	Name	Chip ID	Active	# Card Readers	
2	Evzen Lakatos	456	Yes	2	
4	fero	7852	Yes	1	

Editácia entity (idea)

- V tabuľke bude mať každý riadok tlačítko EDIT



- Klik nás presmeruje na novú stránku



Viac stránková aplikácia

- Nainštalujeme router
  - npm install react-router-dom
- App.tsx:

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <UserList/>,
  },
  {
    path: "/edit-user/:id",
    element: <UserEdit/>,
  },
]);

function App() {
    return (
        <RouterProvider router={router}/>
    );
}
```

Implementácia UserEdit.tsx

- Vytiahneme si ID entity z URL parametra

```
export function UserEdit() {
   const {id} = useParams<{ id: string }>();
  ...
}
```

Implementácia UserEdit.tsx

- Stiahneme si usera a vš. card readerov

```
export function UserEdit() {
  . . .
  useEffect(() => {
    if (id === undefined) {
      return;
    }

    fetch(`${restURL}/users/${id}`)
      .then(response => response.json())
      .then(data => setUser(data as User))
      .catch(error => setError(new Error("Could not fetch users", {cause: error})));

    fetch(`${restURL}/card-readers`)
      .then(response => response.json())
      .then(data => setAllCardReaders(data as CardReader[]))
      .catch(error => setError(new Error("Could not fetch card readers", {cause: error})));

   }, [id]);
  . . .
}
```

Implementácia UserEdit.tsx

- Môžeme poriešiť "chybové" situácie

```
if (error) {
  return (
    <Box sx={{display: 'flex'}}>
      <Alert severity="error">{error.message}</Alert>
    </Box>
  );
}


if (!id) {
  return (
    <Box sx={{display: 'flex'}}>
      <Alert severity="error">User ID not set</Alert>
    </Box>
  );
}

if (!user.id) {
  return (
    <Box sx={{display: 'flex'}}>
      <CircularProgress />
    </Box>
  );
}
```

Implementácia UserEdit.tsx

- Komponent bude vracať iné komponenty (TextField) na editovanie usera

```
return (
 . . .
 <TextField
  fullWidth
  label="Name"
  value={user.name}
  onChange={(e) => {
    setUser({...user, name: e.target.value} as User);
  }}
 />

 . . .

);
```

- Pozor, premennú user nesmieme meniť
- Zavoláme setUser a vytvoríme kópiu súčasného usera so zmeneným menom
- Na to má JS/TS špeciálnu syntax
  - const novýUser = {...starýUser, name: "Nové Meno"} as User

Implementácia UserEdit.tsx

- Vieme aj validovať vstupy užívateľa

```
. . .

function isUserNameValid(): boolean {
   return user.name.length > 0;
}

return (
 . . .
 <TextField
   . . .
   error={!isUserNameValid()}
   helperText={!isUserNameValid() ? "Name must not be empty" : ""}
   . . .
 />
 . . .
);
```

Implementácia UserEdit.tsx

- Funkcia na uloženie užívateľa

```
const navigate = useNavigate();

async function handleSubmit() {
    try {
        await fetch(`http://localhost:8080/users`, {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(user)
        })
    } catch (error) {
        setError(new Error("Could not save user", {cause: error}))
    }
    navigate('/');
}
```

- Funkcia `navigate` nás po uložení vráti na hlavnú stránku

Implementácia UserEdit.tsx

- Tlačidlá na uloženie a zrušenie

```
<ButtonGroup>
  <Button
    disabled={!isUserNameValid()}
    endIcon={<CheckIcon/>}
    variant="contained"
    color="primary"
    onClick={handleSubmit}>
      Save
  </Button>
  <Button variant="contained" color="inherit" onClick={() => navigate('/')}>
      Cancel
  </Button>
</ButtonGroup>
```

Úprava UserList.tsx

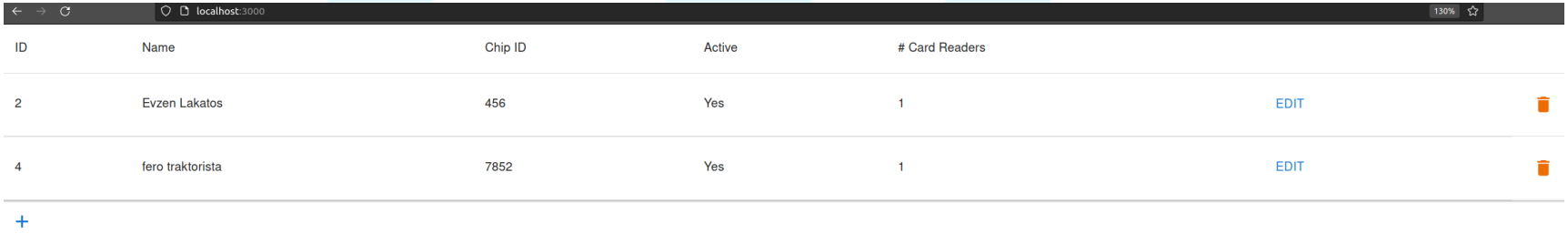
```
const handleEditClick = (userId: number | undefined) => {
   if (userId === undefined) {
      return;
   }

   navigate(`/edit-user/${userId}`);
};
```

```
<TableBody>
 {users.map((user) => (
   <TableRow onClick={() => handleRowClick(user)}>
    . . .
    <TableCell>
     <Button onClick={() => handleEditClick(user.id)}>Edit</Button>
    </TableCell>
    . . .
))}
</TableBody>
```

Pridanie entity

- Analogicky ako editácia
- Akurát pridáme tlačítko + na pridanie entity pod tabuľku



Otázky?