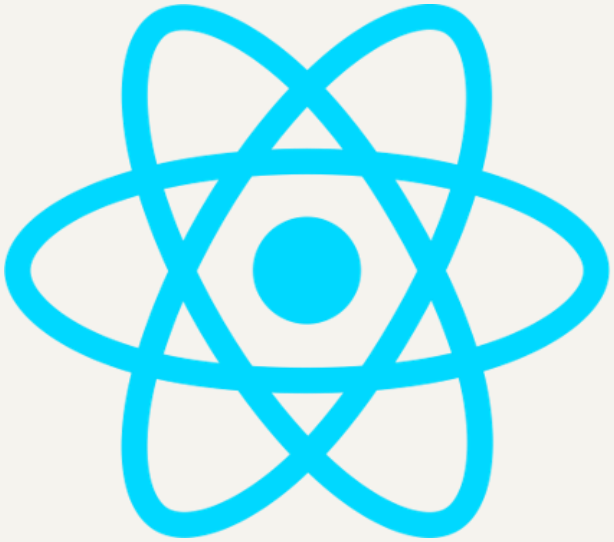


React

Tmavý mód, Servisy, Konfigurácia prostredia, Kompozícia

UINF/PAZ1c

epizóda 12

Tmavý mód

- Material UI má silnú podporu témovania
- Môžete si meniť preddefinované farby, fonty atď.
- Dve hlavné farebné palety
  - Svetlá (predvolená)
  - Tmavá
- Moderné OS podporujú svetlý/tmavý režim.
  - Moderná aplikácia by to mala rešpektovať a adekvátne sa nastaviť

Automatický tmavý mód

App.tsx

```
function App() {
  // hack na zistenie, či OS je v tmavom režime.
  const prefersDarkMode = useMediaQuery('(prefers-color-scheme: dark)');

  // useMemo je továreň, ktorá nám bude vracať novú tému pri zmene nastavenia OS
  const theme = useMemo<Theme>(
    () =>
      createTheme({
        palette: {
          mode: prefersDarkMode ? 'dark' : 'light',
        },
    }),
  [prefersDarkMode],
);

  return (
    <ThemeProvider theme={theme}>
      <CssBaseline />
      ...
    </ThemeProvider>
  );
}
```

Služby (Services)

- Časom môžeme mať viac komponentov, ktoré potrebujú používateľov
- Používateľov nemá spravovať komponent, ale perzistentná vrstva na serveri
- Náš cieľ - vytvoriť službu starajúcu sa o pole users
  - bude komunikovať s REST serverom
  - a sprostredkovávať tak pre komponenty CRUD operácie nad používateľmi na serveri

Vytvorenie služby

userService.ts

```
import {User} from "./user";
import {restURL} from "../config";

export async function getUser(userId: number): Promise<User> {
    const response = await fetch(`${restURL}/users/${userId}`);
    const data = await response.json();
    return data as User;
}

export function getUsers(): Promise<User[]> {
    return fetch(`${restURL}/users`)
      .then(response => response.json())
      .then(data => data as User[]);
}
...
```

- Služby stačí písať ako top-level funkcie
- Pozor, pracujeme s IO
  - Treba vracať Promise<*> objekty
  - Je na nás či použijeme imperatívny zápis (async-await), alebo funkcionálny zápis (then)
    - Vyberte si aký sa vám viac páči

Použitie služby

UserList.tsx

```
import {getUsers} from "../userService";

function UserList() {
    ...
    useEffect(() => {
        getUsers()
            .then(users => setUsers(users))
            .catch(error => setError(new Error("Could not fetch users", {cause: error})));
    }, [])
    ...
}
```

- Stačí nám jednoduchý import.

Konfigurácia prostredia

- Naša aplikácia potrebuje adresu REST servera
- Ak beží lokálne, nech je to http://localhost:8080
- Ak beží na verejnom serveri, tak ju treba prenastaviť cez premennú prostredia

```
// config.ts
export const restURL =
   import.meta.env.VITE_REST_URL ?
   import.meta.env.VITE_REST_URL : 'http://localhost:8080';
```

- Ak pri spustení servovania na NodeJS serveri zadefinujeme VITEE_REST_URL, tak sa použije táto, ináč to bude localhost

Pomocné komponenty

- V UserList chceme dať hlavičku tabuľky a riadky do samostatných komponentov

```
<Table>
  <TableHead>
    ...
  </TableHead>
  <TableBody>
    {users.map((user) => (
      <>
        {/*This is a main content row containing user details and action button*/}
        <TableRow onClick={() => handleRowClick(user)>
          ...
        </TableRow>
        {/*This is a "hidden" row containing card readers that will be shown on click*/}
        <TableRow>
          ...
        </TableRow></>
>
    ))}
  </TableBody>
</Table>
```

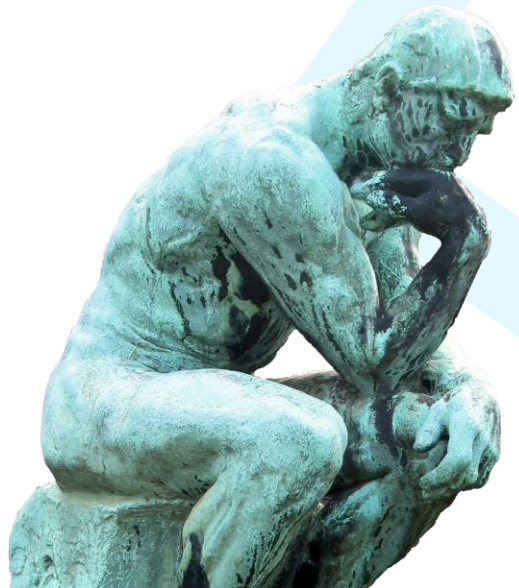
Triviálny komponent

- UserTableHead je triviálny bezstavový komponent

```
export function UserTableHead() {
  return <TableHead>
    <TableRow>
      <TableCell>ID</TableCell>
      <TableCell>Name</TableCell>
      <TableCell>Chip ID</TableCell>
      <TableCell>Active</TableCell>
      <TableCell># Card Readers</TableCell>
      <TableCell>{/*Edit*/}</TableCell>
      <TableCell>{/*Delete*/}</TableCell>
    </TableRow>
  </TableHead>;
}
```

Komponent s atribútami

- UserTableRow je komplexnejší komponent, ktorý potrebuje User-a
- User-a môžeme stiahnuť pomocou useState + useEffect + fetchUser
- Ale jeho rodič (UserList) už predsa má potrebného User-a



Hmm, komponent je funkcia.
Funkcia môže dostať na vstup atribút.

Bingo, dáme komponentu atribút!

Props (rekvizity)

- Komponent akceptuje max 1 atribút
- Navyše s pevne daným názvom - props
- Naštastie rekvizity môžu byť ľubovoľného typu

```
// UserTableRow.tsx
type Props = {
   user: User,
   showDetails: boolean,
   onClick: (clickedUser: User) => void,
   onEditClick: (userId: (number | undefined)) => void,
   onDeleteClick: (userId: (number | undefined)) => void
}
```

- Potrebujeme na vstup usera a handlery na naše eventy
  - Rozkliknutie riadku
  - Klik na edit tlačidlo
  - Klik na delete tlačidlo

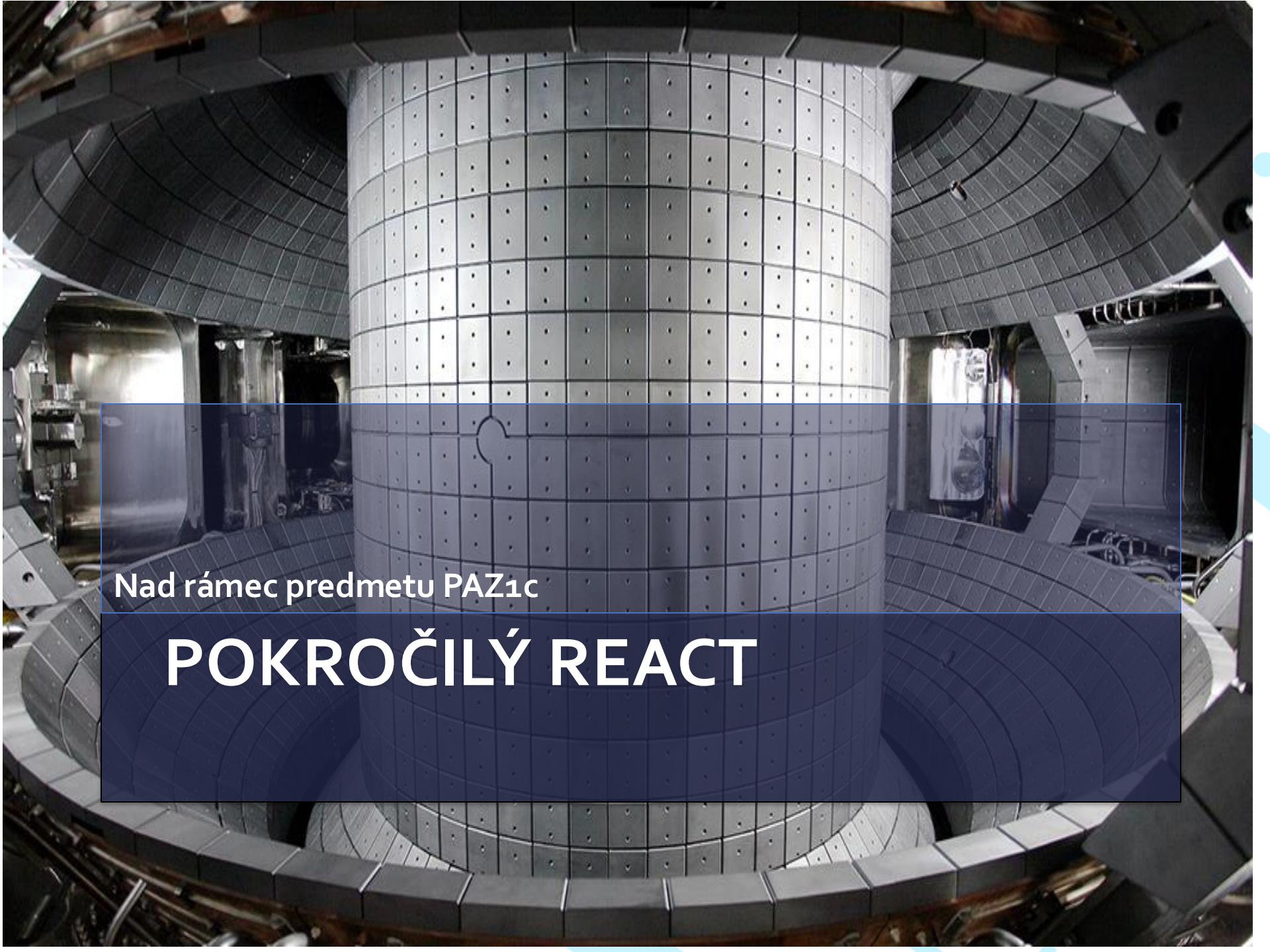
Komponent s rekvizitami

```
// UserTableRow.tsx
export function UserTableRow(props: Props) {
   const {user, showDetails, onClick, onEditClick, onDeleteClick} = props;

   return <><TableRow onClick={() => onClick(user)}>
      <TableCell>{user.id}</TableCell>
      <TableCell>{user.name}</TableCell>
      <TableCell>{user.chipId}</TableCell>
      <TableCell>{user.active ? 'Yes' : 'No'}</TableCell>
      <TableCell>{user.cardReaders ? user.cardReaders.length : 0}</TableCell>
      <TableCell>
         <Button onClick={() => onEditClick(user.id)}>Edit</Button>
      </TableCell>
      <TableCell>
         <IconButton onClick={() => onDeleteClick(user.id)}
                  color='warning'><DeleteIcon/></IconButton>
      </TableCell>

   </TableRow>...<>;
}
```

Kompozícia pomocných komponentov

UserList.tsx

```
<Table>
  <UserTableHead />
  <TableBody>
    {users.map((user) => (
      <>
        <UserTableRow user={user}
                showDetails={selectedUser?.id === user.id}
                onClick={handleRowClick}
                onEditClick={handleEditClick}
                onDeleteClick={handleDeleteClick}
        />
      </>
    ))}
  </TableBody>
</Table>
```



Nad rámec predmetu PAZ1c

POKROČILÝ REACT

Potomkovia komponentu

- Komponent s nastaviteľným detským komponentom

```
import { PropsWithChildren } from "react"

type Props = { name: string }

export function Foo(props: PropsWithChildren<Props>) {
   return props.children
}
```

Pokročilejšie knižnice

Lepší fetch

- Tanstack Query
  - Jednoduchší, vhodnejší pre vlastné projekty
  - https://tanstack.com/query/latest/docs/framework/react/quick-start
- RTK Query
  - Zložitejší, no má toho viac; v korporátoch
  - https://redux-toolkit.js.org/rtk-query/overview
- Automatické cachovanie, handlovanie chýb

React Context (vstavaná knižnica)

- Čo ak máme hierarchiu komponentov a potrebujeme nastaviť n-tý?
- Main.tsx -> App -> UserList -> UserEdit -> ...
- Potrebujem niečo vytvoriť v Main a dať to do UserEdit
- Musím to dať do props v App aj UserList, aj keď to nepotrebujú
- *createContext* a *useContext* umožní "preskočiť" potomkov a dať stav priamo do konkrétneho potomka

Manažovanie globálneho stavu

- Zmena props a context spôsobia "rerender"
- Rerender je pomalý a častý rerender "zaseká" appku
- Najviac sa používa knižnica Redux
  - Skôr vo veľkých firmách
  - https://redux-toolkit.js.org/tutorials/typescript
- Zustand je novšia a jednoduchšia
  - Lepšie pre začiatočníkov
  - https://zustand.docs.pmnd.rs/getting-started/introduction

SEO - hlavná limitácia Reactu

- React robí tzv. client-side web appky (CSR)
  - TypeScript a TSX sa preloží do JavaScriptu a ten sa pošle browseru a následne JavaScript v browseri generuje HTML
- Search Engine Optimization (SEO)
  - Vyhľadávače ako Google nevedia indexovať CSR
  - Ak appka je prístupná iba po prihlásení, tak to je OK
    - (Internet Banking, informačné systémy)
  - Ak appka má časti voľné prístupné, tak to je problém
    - Sociálne siete, portály na predaj lístky

Server Side Rendering React

- server-side web appky (SSR)
  - TypeScript a TSX sa preloží do JavaScriptu a ten urobí HTML, ktoré pošle browseru
- Vhodné ak appka má svoje časti voľne prístupné
  - Sociálne siete, portály na lístky
- **Next.JS**, **Remix**
  - React frejmworky (majú v sebe všetky pokročilé knižnice)
  - React, ale robia server-side rendering (SSR) a static-site generation (SSG)
  - Dobré SEO - vyhľadávače tieto appky vedia čítať

Electron

- https://www.electronjs.org/
- Runtime na desktop aplikácie v JS/TS
- Používa osekaný Chromium (open-source Chrome)
- Použijete ľubovoľnú web UI technológiu
  - React, Vue, Angular, Svelte, ...
- Oproti web appke môžete pristupovať k súborom priamo
- Visual Studio Code, Lens
- Takmer všetky nové desktop appky sa dnes už robia takto
- Náročnejšie na HW ako JavaFX, ale krajšie
- Pozor, nemáte nič zdarma ako "Scene Builder"

React Admin

- Nadstavba nad React
- https://marmelab.com/react-admin/
- Používa Material UI
- Veľmi vhodné na robenie komplexných administračných rozhraní

Ďakujem za pozornosť
počas celého predmetu paz1c

Nech vás sprevádza Sila!