**Begin   
Jak włączyć wsparcie?**$npx // używane do binarnych skryptów do inicjalizacji i do operacji bez projektu

$npx create-react-app first --template typescript

$npm start - creates developer server

\*\*public\*\*

index.html - <div id=”root”> <- miejsce instalacji aplikacji w reacie

\*\*src\*\*

index.tsx

root.render(

<React.ScriptMode> <App /> </>);

) /// uruchamia elementy

className = {value} /// atrybut jak class w html’u

file name & component name (e.x. App) should start with upperCaseLetters

‘

\*\*hooks\*\*

\*\*Example component\*\*

function Message() {  
 *// JSX: Will be compiled to js form a'like html* const *temp* = 'Abxc';  
 return <**h1**>**Hellow** {*temp*}</**h1**>;  
}  
  
export default Message;

import Message from "./Message";  
  
function App() {  
 return <**div**><**Message**/></**div**>;  
}  
  
export default App;

\*\*Wypisywanie tablicy jako listy

function ListGroup(){  
 const *items* = ["Hi", "There", "One"];  
 *items*.map(item => <**li**>{item}</**li**>)  
 return (  
 <><**h1**>**List**</**h1**>  
 <**ul** className="list-group">  
 {*items*.map(item => <**li** key={item} className="list-group-item">{item}</**li**>)}  
 </**ul**></>  
 );  
}

\*\*Hooks\*\*

używane do dostawania się do metod reacta

\*updating number\*

const arr[{nazwa zmiennej przechowywanej}, set{nazwa zmiennej}] = useState({value}) // [0] – varieble; [1] – updater function

const [*selectedIndex*, setSelectedIndex] = useState(-1);

onClick={() => {setSelectedIndex(index)}}

\*forms\*

do przechwytywania zmian inputa

interface ToDoFormProps{  
 *addElement*: (todo:Todo) => void;  
}  
  
export const ToDoForm: FC<ToDoFormProps> = ({addElement}) => {  
  
 const *titleRef* = useRef<HTMLInputElement>(null);  
 const handleSubmit = (e : FormEvent<HTMLFormElement>) => {  
 e.preventDefault();  
  
 const *title* = *titleRef*.*current*?.*value* ?? '';  
 const *todo*: Todo = {*title*: *title*, *id*:*title*, *isDone*: false};  
 addElement(*todo*);  
 }  
 return(  
 <>  
 <**form** onSubmit={(e) => handleSubmit(e)}>  
 <**input** ref={*titleRef*} type="text" name="" id="" required={true}/>  
 <**button** type="submit">**Send data!**</**button**>  
 </**form**>  
 </>  
 );  
};

function App() {  
 const [*todos*, todoesSet] = useState<Todo[]>([]);  
  
 const handleAt = (todo: Todo) => {  
 const *oldTodos* = [...*todos*, todo];  
 *// oldTodos.push(****todo);*** todoesSet(*oldTodos*);  
 }  
 return (  
 <**div** className="App">  
 <**ToDoList** todos={*todos*}/>  
 <**ToDoForm** addElement={handleAt}/>  
 </**div**>  
 );  
}

\*\*Additional \*\*

<></> - pomiędzy tym można zwracać parę obiektów na raz z componentu, dlatego bo bazowo tam jest używany <Fragment>

\*children\*

Do pozyskania dzieci z App.tsx jako argument

interface Prop{

children:text;

}

children: ReactNode; // Przesyła html content

**Formularze**

**\*\*NanoId\*\***

Generuje losowy string jako id

\*\*Układ katalogów\*\*

\*\*src\*\*

components  
\*features\* - wizualna reprezentacja   
toDoList – wszystko

src – features – {name e.x. ToDoList} – components/types

**GUARDS**Metody decydujące czy przepuścić użytkownika na podstawie loginu/tokneu

Response.Cookie[„access-token”, {token}) ??? jak działa w .net

var cookieOptions = new CookieOptions  
{  
 HttpOnly = true,  
 SameSite = SameSiteMode.Strict,  
 Expires = DateTime.Now.AddDays(1)  
};  
  
var cookieOptionsJS = new CookieOptions  
{  
 Expires = DateTime.Now.AddDays(1)  
};  
  
\_httpContextAccessor.HttpContext.Response.Cookies.Append("JWTToken", token, cookieOptions);  
\_httpContextAccessor.HttpContext.Response.Cookies.Append("is-logged","true",cookieOptionsJS);

**FRONT**

SPA – single page application. Koncept mówiący o pojedynczej stronie, która ma podmieniane elementy w zależności od tego pod jakim linkiem znajduje się użytkownik

**Zmienne środowiskowe**

maincatalog/.env

**REACT\_APP\_API\_BASE\_URL={url}**

maincatalog/config.ts

export const API\_URL = process.env.REACT...;

**ReactRouter**

**$npm i react-router-dom**

\*\*src

\*Pages – main component wyświetlający się. Zasadniczo w nim będzie zapisana cała pojedyncza strona. Main, Login, Error (wrong url), AccountSelect, Account, Transfer, TransferHistory, AdminPage, Requests

Layout – Center, Main, Header, Footer, Content. Te elementy będą miały podmienione wartości w zależności od url. Żeby tego nie robić w App.tsx

Routing.tsx – public (nieuwierzytelniony)

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

inside **App**return ( <BrowserRouter> <Routing/> </> )

Center służy wycentrowaniu pojedynczego obiektu. W przypadku banku zrobie np MainPublic

inside Center & Main

return ( <div> <Outlet> </> ) – mówi programowi że dzieci mają tutaj się znajdować

**universal cookie** – służy do odczytu ciasteczek i parsowania ich

$npm i universal-cookie

**\*hooks** – zapisujemy hookie

useIsLogged.ts

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

**\*features** – zapisujemy elementy umieszczane na stronach. Warto aby strony (np. login) były dość czyste i żeby w nich umieścić np

**login** – inside **LoginForm.tsx**&login-form.types.ts (interface z danymi takie jak w dto). **api.ts** (obsługa żądań). **notification.ts**:

**- LoginForm.tsx  
Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie**

**- api.ts**

import {API\_URL} from "../../config";  
import {useCookies} from "react-cookie";  
  
export const useLoginApi = () => {  
 const *authCookieName* = 'auth\_token'  
 const [, *setCookie*,] = useCookies([*authCookieName*]);  
  
 return async (username: string, password: string) => {  
  
 const *response* = await fetch(`${API\_URL}/account/authenticate`, {  
 *method*: 'POST',  
 *headers*: {  
 'Content-Type': 'application/json',  
 'Access-Control-Allow-Origin': '\*',  
 },  
 *body*: *JSON*.stringify({  
 *Email*: username,  
 *Password*: password,  
 })  
 });  
  
 if (*response*.*status* !== 200) throw new *Error*('Login failed');  
  
 const *JWT* = await *response*.text();  
  
 *setCookie*(*authCookieName*, *JWT*, {  
 *expires*: new *Date*(Date.now() + 1000 \* 60 \* 15), *// expires in 15 minutes  
 sameSite*: true  
 });  
  
 *console*.log("login not failed")  
 }  
}

**- notification.ts**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

**Main** – return (<div><Header><Content> <Outlet/> </Content> <Footer/>

**Content** – jak przekazać elementy dzieci do komponentu?   
Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

**Quick type**

Do pisania automatycznych konwersji z json’a

Dto wrzucamy do \*types\* - po prostu tak naprawdę przepisujemy dto które zwraca backend (simple-book.ts e.x)

CreatedAt – ma być stringiem

## **Api fetch?**

**KY** – do przechwytywania http requestów

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie**A screen shot of a computer screen

Description automatically generated**

**useEfect –** runs whenever there is a change in a state of an object

**<link>** - używany w reacie do przemieszczania się między routamii

**Bearer auuthorization**

// Przykład ustawienia nagłówka z tokenem JWT w JavaScript (np. przy użyciu fetch API)

const token = "twój\_token\_jwt";

const headers = {

Authorization: `Bearer ${token}`,

};

fetch("https://example.com/protected-resource", { headers })

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error("Error:", error));