Rest API 2

Прелоадер (от англ."preloader") - предварительный загрузчик, особый индикатор, который информирует пользователя о том, что страница или контент находятся в процессе загрузки. Предварительные загрузчики необходимы на любом профессиональном веб-сайте, поскольку они обеспечивают важную обратную связь с пользователем. Без предварительного загрузчика, особенно на сайтах с большими объемами контента, пользователь может решить, что сайт завис или не работает вообще.

Существует два основных вида прелоадеров: спиннер и строка состояния

components > UI > loader > Loader.jsx > Loader

```jsx
import React from 'react';
import './Loader.css'

const Loader = () => {
    return (
        <div className='loader'>
            <div class="lds-dual-ring"></div>
        </div>

    );
}

export default Loader;
```

```css
.loader{
    display: flex;
    justify-content: center;
}

.lds-dual-ring {
    display: inline-block;
    width: 80px;
    height: 80px;
  }
  .lds-dual-ring:after {
    content: " ";
    display: block;
    width: 64px;
    height: 64px;
    margin: 8px;
    border-radius: 50%;
    border: 6px solid #fff;
    border-color: rgb(120, 203, 155) transparent rgb(120, 203, 155) transparent;
    animation: lds-dual-ring 1.2s linear infinite;
  }
  @keyframes lds-dual-ring {
    0% {
      transform: rotate(0deg);
    }
    100% {
      transform: rotate(360deg);
    }
  }
```

https://loading.io/css/

```
{isPostLoading
  ? <Loader />
  : <PostList remove={removePost} posts={sortedAndSearchedPosts} title=" Список постов" />
}
```

```
const [isPostLoading, setIsPostLoading] = useState(false);

async function fetchPosts() {
  let totalCount;
  setIsPostLoading(true)
  setTimeout(async () => {
    const posts = await PostSevice.getAll(limit, page)
    setPosts(posts.response)
    totalCount = posts.count
    setTotalPages(getPagesCount(totalCount, limit))
    setIsPostLoading(false)
  }, 1000)
}
```

Элементы UI

Кнопка

components > UI > button > MyButton.jsx > ...

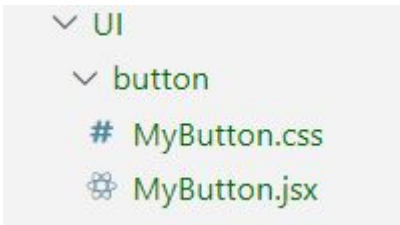
```jsx
import React from 'react';
import './MyButton.css'

const MyButton = ({children, ...props}) => {
    return (
        <button {...props} className='myBtn'>
            {children}
        </button>
    );
}

export default MyButton;
```

components > UI > button > # MyButton.css > .myBtn

```css
.myBtn{
    padding: 5px 15px;
    color: black;
    font-size: 14px;
    background: transparent;
    border: 1px solid black;
    border-radius: 10px;
    cursor: pointer;
    background: rgb(120, 203, 155);
}
```




```jsx
<div className='post__btns'>
    <MyButton onClick={() => remove(post)}>Удалить</MyButton>
</div>
```

Поле ввода - input

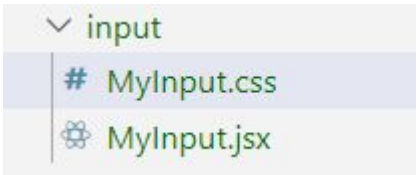
components > UI > input > MyInput.jsx > ...

```jsx
import React from 'react';
import './MyInput.css'

export default function MyInput(props) {
  return (
    <input type="text" {...props} className={'myInput'}/>
  )
}
```

components > UI > input > # MyInput.css > .myInput

```css
.myInput{
    width: 100%;
    padding: 10px 15px;
    margin: 10px 0;
    border: 1px solid black;
    border-radius: 10px;
    background: rgb(197, 225, 202);
}
```



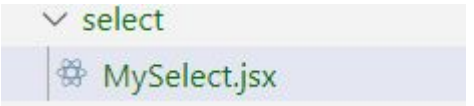

```jsx
<MyInput
    type="text"
    placeholder='Название поста'
    value={post.title}
    onChange={(e) => setPost({ ...post, title: e.target.value })}
/>
```

Select

rc > components > UI > select > MySelect.jsx > ...

```jsx
import React from 'react'

export default function MySelect({ options, defaultValue, value, onChange }) {
    return (
        <select value={value}
        onChange={event => onChange(event.target.value)}
        >
            <option disabled value="value1">{defaultValue}</option>
            {options.map(option =>
                <option key={option.value} value={option.value}>{option.name}</option>
            )}
        </select>
    )
}
```




```jsx
<MySelect
  value={filter.sort}
  onChange={selectedSort => setFilter({...filter, sort: selectedSort})}
  defaultValue='Сортировка по'
  options={[
    { value: 'title', name: ' По названию' },
    { value: 'body', name: ' По описанию' }
  ]}
/>
```

useContext

В React, данные передаются внутрь компонентов через пропсы с верхних уровней на нижние. Но иногда такой подход неудобен, при работе с глобальными данными, которые нужны одновременно во многих компонентах на разных уровнях иерархии.

Хук useContext() позволяет использовать контекст внутри компонента. Для этого нужно выполнить три действия:

1. Инициализировать контекст в том же месте, где инициализируется приложение

```
// Параметром передается значение по умолчанию
// Имя контекста выбирается произвольно
const UserContext = React.createContext({});
```

2. Подключить провайдер и передать данные в контекст через пропс value.

```
// user – данные которые лежат внутри контекста
<UserContext.Provider value={user}>
  <MyComponent />
</UserContext.Provider>
```

3. Получить данные контекста

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

const MyComponent = () => {
  // Возвращает контекст целиком
  const user = useContext(UserContext);

  return <h1>{user.name}</h1>;
}
```

```jsx
import { createContext} from "react";

export const AuthContext = createContext(null)
```

```jsx
import { AuthContext } from './context';

function App() {
  const [user, setUser] = useState(null);

  return (
    <AuthContext.Provider value={{ user, setUser}}>
      <BrowserRouter>
        <Navbar />
        <AppRouter />
      </BrowserRouter>
    </AuthContext.Provider>

  )
}
```

```jsx
import { AuthContext } from '../../../context';

const AppRouter = () => {
    const {user, setUser} = useContext(AuthContext);
    return (
        <Routes>
            <Route element={<ProtectedRoute user={user} />}>
                <Route path='/about' element={<About />} />
                <Route exact path='/posts' element={<Posts />} />
                <Route exact path='/posts/:id' element={<PostIdPage />} />
            </Route>
            <Route path='/*' element={<Login />} />
        </Routes>
    );
}
```

```jsx
const ProtectedRoute = ({ user, redirectPath = '/login' }) => {
  if (!user) {
    return <Navigate to={redirectPath} replace />;
  }

  return <Outlet />;
};
```

```jsx
const Login = () => {
    const {user, setUser} = useContext(AuthContext);
    const handleLogin = (e) => {
        e.preventDefault();
        setUser({ id: '1', name: 'robin' });
      }

    return (
        <div>
            <h1>Страница входа</h1>
            <form>
                <MyInput type='text' placeholder='Name' />
                <MyInput type='password' placeholder='Password' />
                <MyButton onClick={handleLogin}>LogIn</MyButton>
            </form>
        </div>
    );
}
```

Практическое задание

Используя API о фильме «Звездные воины» создать React приложение, которое позволит узнать информацию и героях, планетах, звездных кораблях и т.д. (смотрите что за данные можно получить с API). На главной странице выдать общую информацию и фильмах. Добавить вкладки о героях, планетах и т.д. Выбирая героя на странице о героях можно увидеть подробную карточку с его данными (остальное по аналогии). Дожидаясь загрузки должен быть организован прелоадер. Обязательно созданы отдельные компоненты для UI элементов.

Общий адрес для доступа к API https://swapi.dev/api

Адрес доступа к документации https://swapi.dev/documentation