



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ _____

КАФЕДРА _____ СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5) _____

О Т Ч Е Т

по лабораторной работе №8

по дисциплине: Разработка Интернет-приложений _____

на тему: Работа с React. Создание React-приложения _____

Студент ИУ5-54Б
(Группа)

(Подпись, дата)

(И.О.Фамилия)

Руководитель

(Подпись, дата)

(И.О.Фамилия)

2019 г.

Задание лабораторной работы

В этой лабораторной работе вы создадите React-приложение. Вам нужно будет создать одну страницу со списком репозиторий на гитхабе.

Листинги исходных модулей

- App.js:

```
import React from 'react';
import './App.css';
import axios from 'axios';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      repos: [],
    }
  }

  componentDidMount() {
    axios.get('https://api.github.com/users/isst2000/repos')
      .then(response => {
        this.setState({repos: response.data});
      })
      .catch(error => {
        console.log(error);
      });
  }

  render() {
    let repos = this.state.repos.map(repo => (
      <li key={repo.id}>{repo.name}</li>
    ));
    return (
      <div>
        <h1>Public repositories for isst2000:</h1>
        <ul>
          {repos}
        </ul>
      </div>
    );
  }
}

export default App;
```

- App.test.js:

```
import React from 'react';
import { render } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  const { getByText } = render(<App />);
  const linkElement = getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

- Index.js:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />, document.getElementById('root'));
serviceWorker.unregister();
module.hot.accept();
```

- serviceWorker.js:

```
// This optional code is used to register a service worker.
// register() is not called by default.

// This lets the app load faster on subsequent visits in production, and gives
// it offline capabilities. However, it also means that developers (and users)
// will only see deployed updates on subsequent visits to a page, after all the
// existing tabs open on the page have been closed, since previously cached
// resources are updated in the background.

// To learn more about the benefits of this model and instructions on how to
// opt-in, read https://bit.ly/CRA-PWA

const isLocalhost = Boolean(
  window.location.hostname === 'localhost' ||
  // [::1] is the IPv6 localhost address.
  window.location.hostname === '[::1]' ||
  // 127.0.0.0/8 are considered localhost for IPv4.
  window.location.hostname.match(
    /^127(?:\.(?:25[0-5]|2[0-4][0-9]|01?[0-9][0-9]?)){3}$/
  )
);

export function register(config) {
  if (process.env.NODE_ENV === 'production' && 'serviceWorker' in navigator) {
    // The URL constructor is available in all browsers that support SW.
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location.href);
    if (publicUrl.origin !== window.location.origin) {
      // Our service worker won't work if PUBLIC_URL is on a different origin
      // from what our page is served on. This might happen if a CDN is used to
      // serve assets; see https://github.com/facebook/create-react-app/issues/2374
      return;
    }

    window.addEventListener('load', () => {
      const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;

      if (isLocalhost) {
        // This is running on localhost. Let's check if a service worker still
        // exists or not.
        checkValidServiceWorker(swUrl, config);

        // Add some additional logging to localhost, pointing developers to the
        // service worker/PWA documentation.
        navigator.serviceWorker.ready.then(() => {
          console.log(
            'This web app is being served cache-first by a service ' +
            'worker. To learn more, visit https://bit.ly/CRA-PWA'
          );
        });
      } else {
        // Is not localhost. Just register service worker
        registerValidSW(swUrl, config);
      }
    });
  }
}
```

```

    }
    });
}
}

function registerValidSW(swUrl, config) {
    navigator.serviceWorker
        .register(swUrl)
        .then(registration => {
            registration.onupdatefound = () => {
                const installingWorker = registration.installing;
                if (installingWorker == null) {
                    return;
                }
                installingWorker.onstatechange = () => {
                    if (installingWorker.state === 'installed') {
                        if (navigator.serviceWorker.controller) {
                            // At this point, the updated precached content has been fetched,
                            // but the previous service worker will still serve the older
                            // content until all client tabs are closed.
                            console.log(
                                'New content is available and will be used when all ' +
                                'tabs for this page are closed. See https://bit.ly/CRA-PWA.'
                            );

                            // Execute callback
                            if (config && config.onUpdate) {
                                config.onUpdate(registration);
                            }
                        } else {
                            // At this point, everything has been precached.
                            // It's the perfect time to display a
                            // "Content is cached for offline use." message.
                            console.log('Content is cached for offline use.');
```

// Execute callback

```

                            if (config && config.onSuccess) {
                                config.onSuccess(registration);
                            }
                        }
                    }
                }
            });
        })
        .catch(error => {
            console.error('Error during service worker registration:', error);
        });
}

function checkValidServiceWorker(swUrl, config) {
    // Check if the service worker can be found. If it can't reload the page.
    fetch(swUrl, {
        headers: { 'Service-Worker': 'script' }
    })
        .then(response => {
            // Ensure service worker exists, and that we really are getting a JS file.
            const contentType = response.headers.get('content-type');
            if (
                response.status === 404 ||
                (contentType !== null && contentType.indexOf('javascript') === -1)
            ) {
                // No service worker found. Probably a different app. Reload the page.
                navigator.serviceWorker.ready.then(registration => {
                    registration.unregister().then(() => {
                        window.location.reload();
                    });
                });
            } else {

```

```

        // Service worker found. Proceed as normal.
        registerValidSW(swUrl, config);
    }
})
.catch(() => {
    console.log(
        'No internet connection found. App is running in offline mode.'
    );
});
}

export function unregister() {
    if ('serviceWorker' in navigator) {
        navigator.serviceWorker.ready.then(registration => {
            registration.unregister();
        });
    }
}

```