МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ОСТРОЗЬКА АКАДЕМІЯ"
Кафедра економіко-математичного моделювання та інформаційних технологій

ЗВІТ
Лабораторна робота №4
Використання REACT REDUCE

здобувачки вищої освіти
першого (бакалаврського) рівня
третього року навчання групи КН-31
спеціальності 122 Комп'ютерні науки
ОПП «Комп'ютерні науки»

Лайтер Ярини Семенівни
(прізвище, ім'я, по батькові)

База лабораторної - кафедра економіко-математичного моделювання та інформаційних технологій

Керівник лабораторної від університету

_____
(науковий ступінь, учене звання керівника)

_____
(Власне ім'я та ПРІЗВИЩЕ)

Члени комісії:

_____

Острог - 2023

Зміст

Опис

Тема: Використання react redux
Мета: Ознайомити студентів зі створенням додатків на React з використанням Redux та редукторів (reducers)

Завдання
Налаштування проекту:

1. Створіть новий проект React за допомогою Create React App або іншого інструменту на ваш вибір.
2. Встановіть бібліотеку Redux та react-redux для управління станом додатку.

Створення компонентів:

1. Створіть компонент TodoList, який буде відображати список завдань.
2. Створіть компонент TodoItem, який буде відображати окреме завдання.
3. Створіть компонент TodoForm, який буде відповідати за додавання нових завдань.

Визначення дій (actions) та Створення редукторів (reducers):

1. Створіть дії (actions) для додавання нового завдання, видалення завдання та позначення завдання як виконаного.
2. Створіть редуктор для обробки дій, пов'язаних з завданнями (наприклад, додавання, видалення, позначення як виконаного).
3. Підключіть цей редуктор до свого Redux-стору.

Підключення Redux до компонентів та Реалізація функціональності:

1. Використовуйте бібліотеку react-redux для підключення компонентів TodoList, TodoItem, та TodoForm до Redux-стору.
2. Доступ до стану і дій має бути забезпечений через властивості та дії компонентів.
3. Додайте можливість додавати нові завдання через TodoForm.
4. Дозвольте користувачам позначати завдання як виконані та видаляти завдання через TodoItem.
5. Оновлюйте список завдань після кожної дії.

Стилізація:

1. Застосуйте стилізацію за допомогою CSS або CSS-препроцесора за своїм вибором, щоб додаток виглядав привабливо.

Додаткова функціональність (необов'язково):

Додайте можливість редагування завдань.
Зберігайте стан додатку в локальному сховищі для збереження даних між сеансами.

Виконання

1. **Створення нового проекту React:**

Створюємо проект. Вводимо в терміналі комаду npm create vite@latest:

```
PS D:\OA_3\lab4> npm init vite@latest
√ Project name: ... lab4_vite
√ Select a framework: » React
√ Select a variant: » JavaScript

Scaffolding project in D:\OA_3\lab4\lab4_vite...

Done. Now run:

  cd lab4_vite
  npm install
  npm run dev
```

Рис 1.1 Виконання команди

```
PS D:\OA_3\lab4\lab4_vite> npm install redux

added 271 packages, and audited 272 packages in 19s

97 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\OA_3\lab4\lab4_vite> npm install react-redux

added 6 packages, and audited 278 packages in 4s

97 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рис 1.2 Підключення бібліотек

2. **Створення компонентів**:

   TodoList, який відображає список завдань:

Рис 2. 1 Створення компонента TodoList.jsx

TodoItem, який відображає окреме завдання:

Рис 2. 2 Створення компонента TodoItem.jsx

TodoForm, який відповідає за додавання нових завдань:

Рис 2.3 Створення компонента TodoForm.jsx

TotalCompleteItems.jsx, який показує загальну кількість виконаних компонентів:

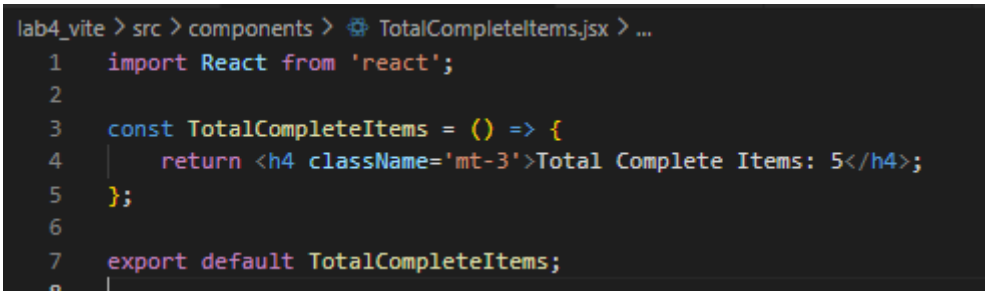

Рис 2.4 Створення компонента TotalCompleteItems.jsx

App.jsx, який відповідає за виклик компонентів:

lab4_vite > src > App.jsx > ...

```jsx
import React from 'react';
import 'bootswatch/dist/darkly/bootstrap.min.css';
import 'bootstrap-icons/font/bootstrap-icons.css';
import TodoList from './components/TodoList';
import TodoForm from './components/TodoForm';
import TotalCompleteItems from './components/TotalCompleteItems';

const App = () => {
    return (
        <div className='container bg-dark p-4 mt-5'>
            <h1 className='text-center'>My Todo List</h1>
            <TodoForm />
            <TodoList />
            <TotalCompleteItems />
        </div>
    );
};

export default App;
```

Рис 2.5 Створення компонента App.jsx

main.jsx, який який імпортує бібліотеки та компоненти і візуалізує додаток:

lab4_vite > src > main.jsx > ...

```jsx
import React from 'react';
import { createRoot } from 'react-dom/client';
import './index.css';
import App from './App';
import store from './redux/store';
import { Provider } from 'react-redux';

const root = createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

Рис 2.6 Код компонента main.jsx

В результаті отримуємо:

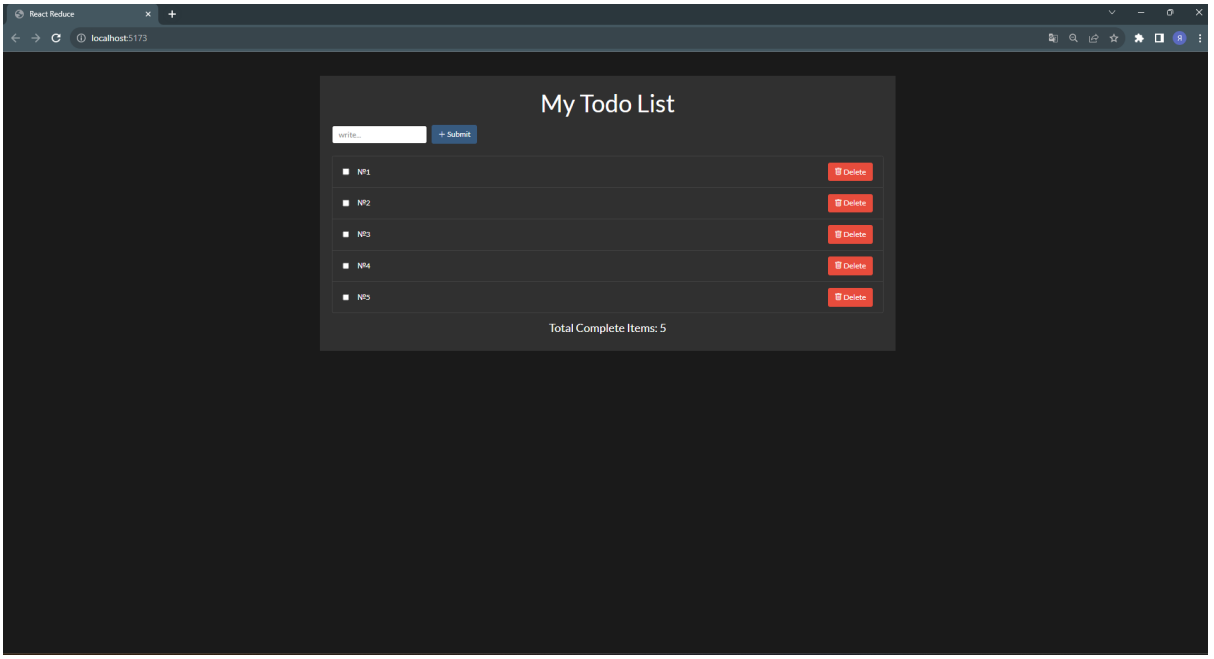

Рис 2.7. Результат програми

3. **Визначаємо дії (actions) та створюємо редуктиви (reducers)**:

Створюємо редуктор для обробки дій, пов'язаних з завданнями (додавання, видалення, позначення як виконаного). У файлі *todoSlice.jsx* є три редуктори, які представляють три різні дії: addTodo, toggleComplete і deleteTodo. Кожен редуктор виконує відповідну дію, змінюючи стан згідно з параметрами action:

lab4_vite > src > redux > todoSlice.jsx > ...

```jsx
import { createSlice } from '@reduxjs/toolkit';
import { nanoid } from 'nanoid';

export const todoSlice = createSlice({
  name: 'todos',
  initialState: [],
  reducers: {
    addTodo: (state, action) => {
      const todo = {
        id: nanoid(),
        title: action.payload.title,
        completed: false,
      };
      state.push(todo);
    },
    toggleComplete: (state, action) => {
      const index = state.findIndex((todo) => todo.id === action.payload.id);
      state[index].completed = action.payload.completed;
    },
    deleteTodo: (state, action) => {
      return state.filter((todo) => todo.id !== action.payload.id);
    },
  },
});

export const { addTodo, toggleComplete, deleteTodo } = todoSlice.actions;

export default todoSlice.reducer;
```

Рис 3. 1 Код сторінки todoSlice.jsx

Підключаємо цей редуктор до нашого Redux-стору.

lab4_vite > src > redux > store.jsx > ...

```jsx
import { configureStore } from '@reduxjs/toolkit';
import todoReducer from './todoSlice';

export default configureStore({
    reducer: {
        todos: todoReducer,
    },
});
```

Рис 3.2 Код сторінки store.jsx

4. **Підключаємо Redux до компонентів та реалізуємо функціональності:**

Використали бібліотеку react-redux для підключення компонентів TodoList, TodoItem, та TodoForm до Redux-стору.
TodoList, який відображає список завдань та Оновлюйте список завдань після кожної дії:

lab4_vite > src > components > TodoList.jsx > ...

```jsx
import React, { useEffect } from 'react';
import TodoItem from './components/TodoItem';
import { useSelector, useDispatch } from 'react-redux';

const TodoList = () => {
  const dispatch = useDispatch();
  const todos = useSelector((state) => state.todos);

  useEffect(() => {
  }, [dispatch]);

  return (
    <ul className='list-group'>
      {todos.map((todo) => (
        <TodoItem id={todo.id} title={todo.title} completed={todo.completed} />
      ))}
    </ul>
  );
};

export default TodoList;
```

Рис 4.1 Код компонента TodoList.jsx

TodoItem, який дозволяє користувачам позначати завдання як виконані та видаляти їх:

Рис 4. 2 Код сторінки TodoItem.jsx

TodoForm, який відповідає за додавання нових завдань:

Рис 4.3 Створення компонента TodoForm

TotalCompleteItems.jsx, який показує загальну кількість виконаних компонентів:

Рис 4.4 Створення компонента TotalCompleteItems.jsx

В результаті отримуємо:

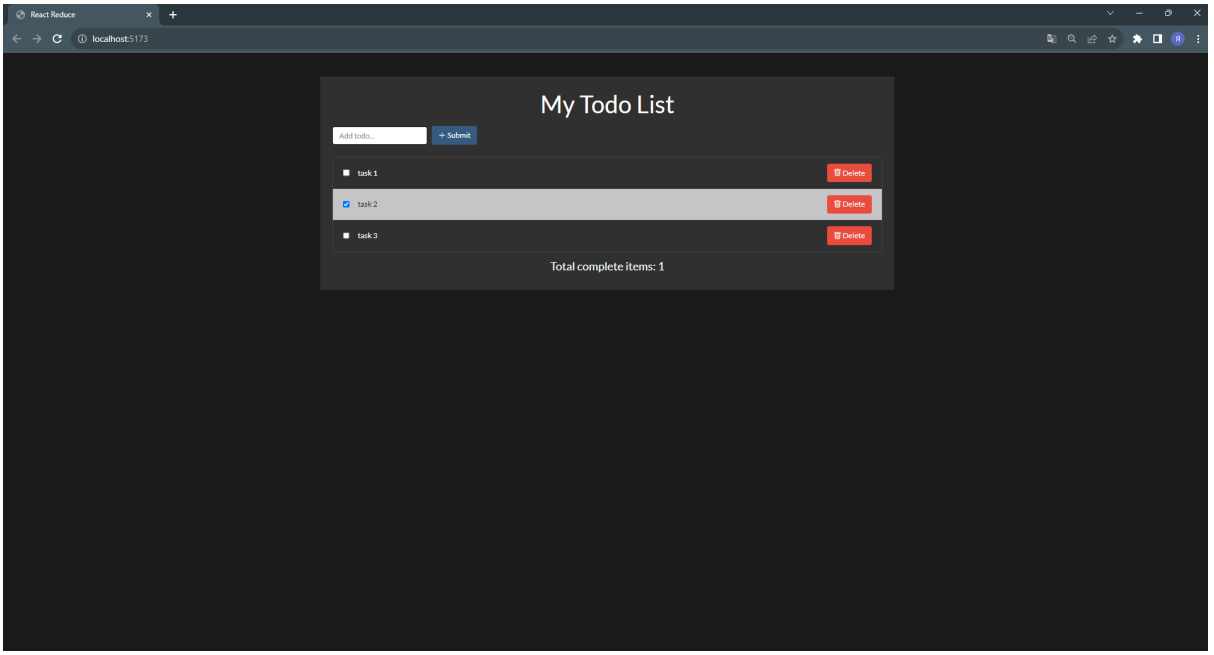

Рис 4.5. Результат програми

Висновок:

На цій лабораторній роботі ми успішно реалізували додаток на React, який використовує Redux для керування станом. Ми створили редуктори для обробки різних дій, таких як додавання, видалення та оновлення елементів.

Крім того, ми застосували стилізацію за допомогою CSS (а саме bootstrap) для кращого вигляду нашого додатка.