Приднестровский государственный университет им. Т.Г. Шевченко
Физико-технический институт

ОСНОВЫ ВЕБ-РАЗРАБОТКИ С ПРИМЕНЕНИЕМ ИИ-АССИСТЕНТОВ

Лабораторный практикум

Разработал:
ст. преподаватель
кафедры ИТ
Бричаг Д.В.

г. Тирасполь
2025 г.

Лабораторная работа №8

Введение в React + JSX

Цель работы:

− Перейти от классического подхода «HTML + JS» к компонентному подходу в разработке.

− Понять, как React организует интерфейс в виде компонентов, использующих состояние и свойства (state и props).

− Научиться с помощью ИИ генерировать и разбирать React-компоненты.

Теоретическая справка

Что такое React: Отказ от императивного подхода

React – это не фреймворк, а библиотека JavaScript (разработанная Meta, ранее Facebook), предназначенная для построения пользовательских интерфейсов (UI).

Проблема, которую решает React: проблема синхронизации

В классическом подходе ("jQuery-подход"), когда данные менялись (например, пришел новый ответ с сервера), вам приходилось вручную искать элемент в DOM и императивно (давая пошаговые команды) его изменять:

```
$('p.counter').text('Новое значение')
$('button').removeClass('active')
```

По мере усложнения приложения этот процесс превращается в "борьбу" с DOM. Если данные меняются в трех местах, вы должны не забыть вручную обновить все три элемента.

Решение React: декларативный подход

React переходит к декларативному подходу, основанному на принципе:

UI = функция от состояния (User Interface = f(state))

Это означает:

- вы описываете (декларируете), как должен выглядеть UI при любом состоянии данных;

- вы меняете только состояние (state);
- React сам решает, как и какие части DOM нужно обновить, чтобы они соответствовали новому состоянию. Вам больше не нужно давать команды "найти этот div и изменить его класс".

Аналогия:

Императивный (jQuery):

```
«Возьми стакан. Иди к крану. Набери воды. Принеси мне.»
```

Декларативный (React):

```
«Дай мне стакан воды.» (Процесс выполнения берет на себя React).
```

JSX – расширение JavaScript (Syntax Extension)

JSX (JavaScript XML) — это специальное расширение синтаксиса, которое позволяет писать HTML-подобную разметку прямо в файлах JavaScript.

Зачем нужен JSX?

На первый взгляд, это выглядит как смешивание HTML и JS, но на самом деле это упрощает разработку компонентов.

Вместо того чтобы вручную создавать элементы с помощью громоздких методов:

```
// Чистый React (без JSX)
function Hello() {
  return React.createElement('h1', null, 'Привет, мир!');
}
```

Вы пишете код, который визуально похож на конечный результат:

```
// С использованием JSX
function Hello() {
```

```
  // Проще читать и писать
  return <h1>Привет, мир!</h1>;
}
```

Ключевой момент: JSX — это не HTML. Это синтаксический сахар, который во время сборки (с помощью Babel) транспилируется обратно в вызовы React.createElement().

Важные отличия JSX от HTML:

- Вместо атрибута class используется className (поскольку class — зарезервированное слово в JS).
- Стили пишутся как JS-объекты: style={{ color: 'blue', fontSize: '14px' }} (вместо CSS-строки).
- Для вставки JS-переменных или выражений используется синтаксис фигурных скобок {...}.

```
// Пример использования фигурных скобок
const userName = "Алекс";
const element = <p>Привет, {userName.toUpperCase()}!</p>;
```

Компоненты, Пропсы и Состояние: Основы React

React-приложение — это дерево компонентов, каждый из которых отвечает за свою часть интерфейса и данных.

1. Компоненты (Component)

Компонент — это изолированная, переиспользуемая функция (или класс), которая принимает входные данные и возвращает элемент React, описывающий, что должно быть на экране.

Типы компонентов:

- **Функциональные компоненты** (Functional Components): Современный и предпочтительный способ. Это просто функции, возвращающие JSX.

```
function Button() {
  return <button>Нажми меня</button>;
}
```

- **Классовые компоненты** (Class Components): Более старый подход, требующий наследования от React.Component и использования метода render(). Сегодня используются редко, но могут встречаться в старом коде.

2. Пропсы (Props — Properties)

Пропсы — это механизм, с помощью которого родительский компонент передает данные дочернему. Пропсы передаются в компонент как аргументы функции и являются только для чтения (immutable).

Аналогия: Пропсы — это настройки, которые вы задаете при покупке бытовой техники (цвет, мощность). Вы можете ими пользоваться, но не можете изменить их изнутри.

```
// 1. Дочерний компонент (принимает props)
function Greeting(props) {
  // props всегда является объектом { name: '...' }
  return <h2>Привет, {props.name}!</h2>;
}

// 2. Родительский компонент (передает props)
function App() {
  // name="Михаил" — это пропс
  return <Greeting name="Михаил" />;
}
```

3. Состояние (State)

Состояние (State) — это данные, которые могут меняться внутри компонента с течением времени. State определяет, как компонент должен выглядеть в данный момент.

Для управления состоянием в функциональных компонентах используются Хуки (Hooks). Самый важный из них — useState.

Аналогия: State — это переменные, которые хранят текущее состояние объекта (например, заряженность батареи, текущее время).

Как работает useState:

```
function Counter() {
  // [count, setCount] = [текущее значение, функция для изменения]
  const [count, setCount] = React.useState(0);

  // При клике:
  // 1. Вызываем setCount(count + 1).
  // 2. React обновляет значение count.
  // 3. React автоматически перерисовывает ТОЛЬКО этот компонент.
  return (
    <div>
      <p>Счётчик: {count}</p>
      <button onClick={() => setCount(count + 1)}>+</button>
    </div>
  );
}
```

Важно: Вы никогда не должны менять state напрямую (count = 5). Вы всегда должны использовать функцию-сеттер (setCount(5)). Это гарантирует, что React узнает об изменении и запустит перерисовку.

Цель лабораторной работы заключается в том, чтобы освоить компонентный подход. Ваша задача — увидеть, как React полностью меняет взгляд на разработку:

- jQuery: Мы думали о действиях ("что сделать, чтобы изменить UI?")
- React: Мы думаем о данных и структуре ("как должен выглядеть UI при таких-то данных?")

React освобождает разработчика от необходимости вручную управлять DOM, позволяя сосредоточиться на логике и структуре данных.

Практическая часть

Сформируем простое техническое задание. Необходимо:

Реализовать простое To-Do приложение, используя библиотеку React JS:

- подготовить рабочую среду;
- сгенерировать код приложения;
- выполнить сборку;
- проверить работоспособность.

Подготовка проекта

Vite — это современный инструмент, который запускает ваш React-проект за считанные секунды, в отличие от более старого create-react-app.

1. Создайте проект (Команда создаст папку lab8-react-intro):

```
npm create vite@latest lab8-react-intro -- --template react
```

(Если спросит, выберите React и JavaScript)

2. **Перейдите в папку и установите зависимости:**

```
cd lab8-react-intro
npm install
```

3. **Запустите сервер разработки:**

```
npm run dev
```

Откройте http://localhost:5173 (или тот адрес, что Vite укажет в консоли). Вы должны увидеть стартовую страницу React.

4. Очистка:

Откройте папку в VS Code. Зайдите в src/App.jsx и удалите все содержимое из return(). Оставьте только пустой . Также удалите файл App.css, чтобы он не мешал.

Используем ИИ для генерации

Ваша цель — получить "рыбу" (boilerplate) кода. Вместо отдельных промптов, используйте один **"мастер-промпт"** для ИИ:

```
«Сделай минимальное ToDo-приложение на React. Используй функциональные компоненты и хук useState.
Главный компонент App.jsx должен хранить массив задач в состоянии.
Компонент AddTaskForm.jsx должен содержать <input> и <button>, и через пропсы получать функцию addTask.
Компонент ToDoList.jsx должен получать массив задач tasks и функцию removeTask через пропсы.
Компонент ToDoItem.jsx должен отображать текст задачи и кнопку 'Удалить'.»
```

Реализация ToDo-приложения (Сборка)

После того как ИИ сгенерировал код, ваша задача — правильно его "собрать". Создайте в папке src/ новые файлы для каждого компонента.

Минимальный набор компонентов:

- **App.jsx** — главный компонент
- **ToDoList.jsx** — список задач
- **ToDoItem.jsx** — отдельная задача
- **AddTaskForm.jsx** — форма для добавления

Пример структуры:

```
// App.jsx
import { useState } from 'react';
import AddTaskForm from './AddTaskForm';
import ToDoList from './ToDoList';

function App() {
  const [tasks, setTasks] = useState([]);
  const addTask = (text) => setTasks([...tasks, { id: Date.now(), text }]);
  const removeTask = (id) => setTasks(tasks.filter(t => t.id !== id));

  return (
    <div className="app">
      <h1>Мои задачи</h1>
      <AddTaskForm addTask={addTask} />
      <ToDoList tasks={tasks} removeTask={removeTask} />
    </div>
  );
}
```

```
export default App;
```

Понимание ключевых понятий

После генерации с помощью ИИ – попросите его пошагово объяснить, что делает каждая часть кода

```
Объясни, как в этом коде работает состояние useState и передача пропсов.
```

```
Разбери компонент ToDoItem построчно.
```

Доработка

1. Добавьте стили к списку задач.
2. Реализуйте сохранение задач в localStorage.
3. Добавьте кнопку «Очистить всё».
4. Попросите ИИ сгенерировать README.md с описанием приложения.

Работа с git и GitHub

1. Инициализируйте репозиторий:

```
git init
git add .
git commit -m "Lab8: React ToDo App"
```

2. Создайте новый репозиторий на GitHub.
3. Свяжите локальный проект с GitHub:

```
git remote add origin https://github.com/<user>/lab8-react-intro.git
git branch -M main
```

```
git push -u origin main
```

Отчёт (в README.md)

В README.md добавить:

- Заголовок: Лабораторная работа №8. Введение в React + JSX
- Скриншоты приложения.
- Краткое объяснение:
  - Что такое компонент, пропсы и состояние.
  - Какую часть кода помог написать ИИ.
  - Что вы поняли о компонентном подходе.

Результаты работы

В итоге у вас должно быть:

1. Рабочее React-приложение (ToDo).
2. Исходный код с компонентами и состоянием.
3. Доработан дополнительный функционал
4. Репозиторий на GitHub и отчёт в README.md.

Критерии оценки

- Приложение запускается и работает (2 балла).
- Реализовано состояние (useState) и передача пропсов (2 балла).
- Реализована сохранение данных (2 балла).
- Реализована очистка (2 балла).
- Код выложен на GitHub и оформлен отчёт (2 балла).