React - знакомство

Что такое фреймворк (библиотека)
Что такое react
Добавляем react в свой проект
Простое приложение на react
Виртуальный DOM
Компонентный подход
Знакомство с JSX

Что такое фреймворки

Фреймворки в front-end разработке — это библиотеки или наборы инструментов, которые предоставляют структурированные способы разработки пользовательских интерфейсов (UI). Они включают готовые компоненты, шаблоны, стили и скрипты, которые помогают ускорить разработку и упростить создание сложных приложений.

Преимущества front-end фреймворков

Ускорение разработки: Фреймворки предлагают готовые компоненты и шаблоны, что позволяет разработчикам быстрее создавать и развертывать приложения.

Стандартизация кода: Использование фреймворков способствует унификации и стандартизации кода, что облегчает его поддержку и понимание другими разработчиками.

Оптимизация производительности: Многие фреймворки оптимизированы для повышения производительности, включая асинхронную загрузку компонентов и минимизацию кода.

Поддержка сообщества: Популярные фреймворки имеют активное сообщество разработчиков, которое предоставляет документацию, примеры и поддержку.

Модульность: Фреймворки часто поддерживают модульную архитектуру, позволяя разработчикам использовать только те части, которые им нужны, что делает код более управляемым и расширяемым.

Недостатки front-end фреймворков

Сложность обучения: Освоение фреймворков может занять время, особенно для новичков, так как они часто требуют понимания новых концепций и паттернов.

Ограничения: Фреймворки могут накладывать ограничения на архитектуру и дизайн приложений, что может быть не всегда удобным для уникальных требований проекта.

Зависимости: Использование фреймворков создает зависимости, что может привести к трудностям при обновлении и поддержке проектов в долгосрочной перспективе.

Избыточность: Некоторые фреймворки могут быть избыточными для простых проектов, добавляя ненужные сложности и объёма кода.

Перформанс: Несмотря на оптимизации, фреймворки могут вносить дополнительные накладные расходы на выполнение кода, особенно если они не используются эффективно.

Зачем нужны front-end фреймворки?

Фреймворки нужны для упрощения и ускорения процесса разработки, стандартизации подходов к созданию пользовательских интерфейсов, улучшения управления проектом и упрощения командной работы. Они позволяют разработчикам сосредоточиться на логике приложения, а не на рутинных задачах, таких как управление состоянием, маршрутизация или работа с DOM.

SPA (Single Page Application)

SPA (Single Page Application) — это тип веб-приложения, которое загружается на одну веб-страницу и динамически обновляет контент по мере взаимодействия пользователя, без полной перезагрузки страницы. В SPA всё приложение загружается один раз при первом посещении, и дальнейшие взаимодействия с приложением происходят без перезагрузки страницы, обычно с использованием JavaScript.

Особенности SPA

Динамическое обновление контента: SPA обновляют только те части страницы, которые необходимо изменить, вместо загрузки целых новых страниц. Это делает приложение быстрее и более отзывчивым.

Маршрутизация на стороне клиента: В SPA часто используется клиентская маршрутизация, которая позволяет пользователю переходить между различными "страницами" или представлениями внутри приложения, не перегружая страницу. Это достигается изменением URL-адресов без перезагрузки страницы.

Асинхронные запросы к серверу: SPA часто используют технологии, такие как AJAX (Asynchronous JavaScript and XML), для асинхронного обмена данными с сервером. Это позволяет получать новые данные и обновлять интерфейс без полной перезагрузки страницы.

Рендеринг на стороне клиента: В традиционных веб-приложениях сервер обрабатывает запросы и возвращает HTML-страницы. В SPA большая часть логики рендеринга происходит на стороне клиента (в браузере), а сервер чаще всего возвращает только данные в формате JSON.

React

React — это популярная библиотека JavaScript для создания пользовательских интерфейсов, разработанная и поддерживаемая Facebook. Она используется для создания компонентных интерфейсов и позволяет разработчикам строить масштабируемые и эффективные веб-приложения.

Краткая история React

React был создан в Facebook и впервые представлен в 2013 году. Основная цель его разработки заключалась в улучшении производительности и удобства поддержки пользовательских интерфейсов. Facebook использует React в своих продуктах, таких как Facebook и Instagram. С момента выпуска React быстро стал одним из самых популярных инструментов для веб-разработки, благодаря своей простоте, гибкости и производительности.

Плюсы React

Компонентный подход: React позволяет создавать интерфейсы из отдельных компонентов, что упрощает повторное использование кода, тестирование и поддержку.

Виртуальный DOM: React использует виртуальный DOM для оптимизации обновлений пользовательского интерфейса. Это делает приложение более производительным, минимизируя реальные изменения DOM.

Однонаправленный поток данных: React использует однонаправленный поток данных, что упрощает отслеживание состояния и отладку приложения.

Поддержка сообщества: React имеет обширное и активное сообщество, что обеспечивает доступ к множеству библиотек, инструментов и ресурсов.

Совместимость: React можно использовать в сочетании с другими библиотеками и фреймворками, что делает его гибким и универсальным решением.

Минусы React

Крутая кривая обучения: Хотя основы React относительно просты, более продвинутые концепции, такие как управление состоянием, контекст, хуки и маршрутизация, могут быть сложными для понимания новичками.

Проблемы с SEO: Как и другие SPA-фреймворки, React-приложения могут испытывать трудности с поисковой оптимизацией, хотя это можно частично решить с помощью серверного рендеринга (SSR).

Быстро меняющаяся экосистема: Экосистема React постоянно развивается, и разработчики должны следить за новыми релизами и изменениями, что требует времени и ресурсов.

Неопределенность в архитектуре: В отличие от более "полных" фреймворков, React фокусируется только на пользовательском интерфейсе, оставляя выбор архитектуры и дополнительных инструментов на усмотрение разработчиков. Это может быть как плюсом, так и минусом, в зависимости от контекста.

Зачем нужен React?

React используется для создания динамических и интерактивных веб-приложений. Его компонентная архитектура и мощные возможности управления состоянием позволяют разработчикам строить сложные интерфейсы, которые легко масштабировать и поддерживать. React также популярен в разработке мобильных приложений через React Native.

Компонентный подход в разработке

Компонентный подход в разработке — это методология, при которой пользовательские интерфейсы создаются из независимых, многократно используемых и легко управляемых элементов, называемых компонентами. Каждый компонент представляет собой часть интерфейса (например, кнопку, форму, карточку) и инкапсулирует в себе как логику, так и разметку, а иногда и стили.

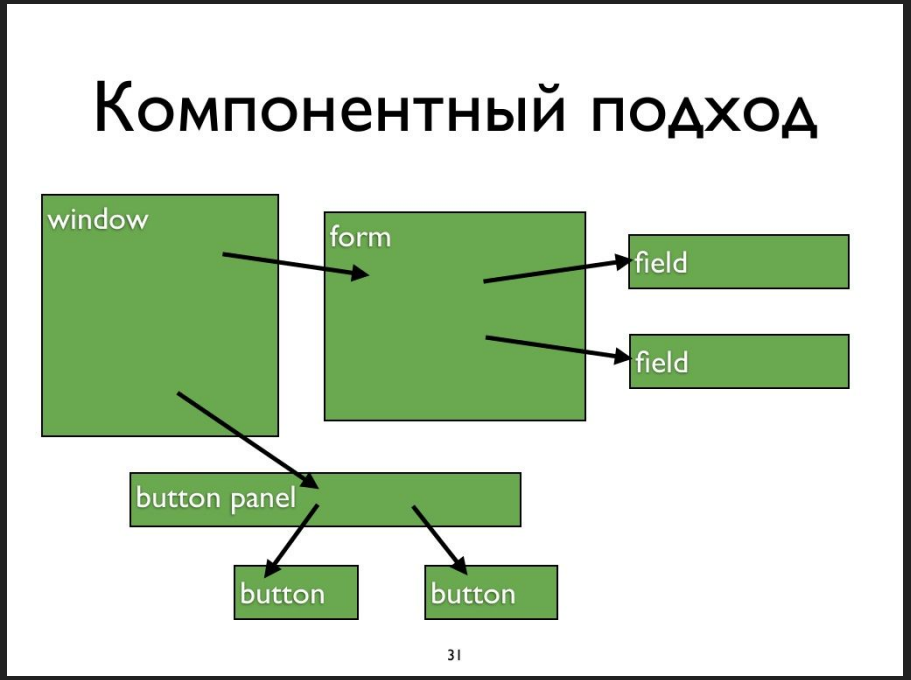
Смысл компонентного подхода

Инкапсуляция: Компоненты инкапсулируют свою функциональность и состояние, что позволяет уменьшить взаимозависимости между разными частями приложения.

Повторное использование: Один и тот же компонент может быть использован в разных частях приложения, что снижает объем дублирующегося кода и упрощает поддержку.

Модульность: Компоненты разрабатываются и тестируются отдельно, что способствует более структурированной и организованной архитектуре кода.

Облегчение поддержки и расширяемости: Изменения в одном компоненте, как правило, не влияют на другие части системы, что упрощает поддержку и развитие приложения.

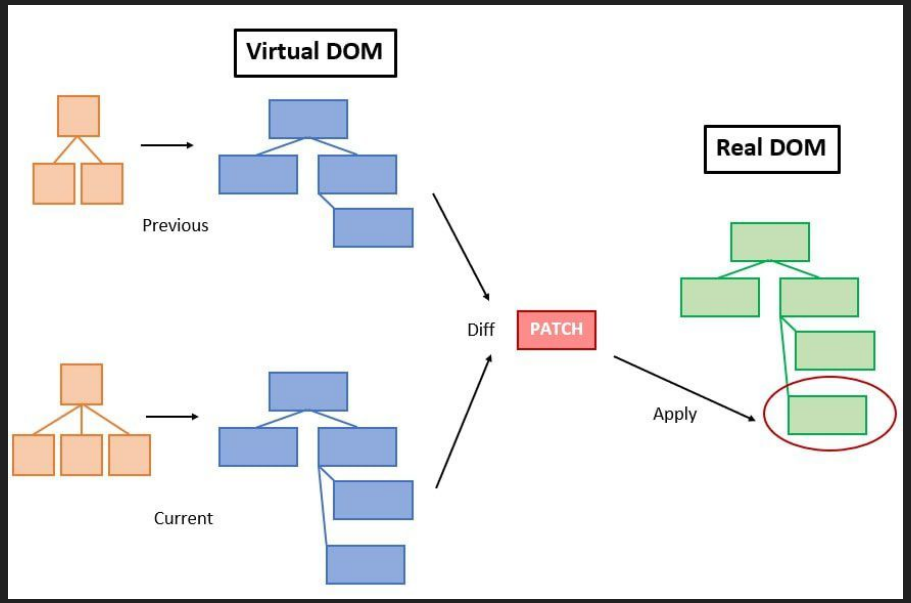


Виртуальный DOM (Virtual DOM)

Виртуальный DOM (Virtual DOM) — это концепция, используемая в некоторых фреймворках JavaScript, таких как React, для улучшения производительности и оптимизации манипуляций с документом. Виртуальный DOM представляет собой легковесную копию реального DOM, хранящуюся в памяти.

Смысл и принцип работы

Основная идея виртуального DOM заключается в том, что любые изменения в интерфейсе сначала применяются к виртуальной модели, а не напрямую к реальному DOM. Затем система сравнивает текущее состояние виртуального DOM с его предыдущей версией (процесс, известный как "сравнение" или "диффинг") и вычисляет минимальное количество изменений, необходимых для обновления реального DOM. Только эти изменения применяются к реальному DOM, что значительно снижает количество операций и повышает производительность.



Добавляем React на сайт

Шаг 1: Добавляем DOM-контейнер в HTML

Шаг 2: Добавляем script-теги

Шаг 3: Создаём React-компонент

Готово!

```
<!-- ... остальной HTML ... -->

<div id="like_button_container"></div>

<!-- ... остальной HTML ... -->
```

```
'use strict';

const e = React.createElement;

class LikeButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = { liked: false };
  }

  render() {
    if (this.state.liked) {
      return 'You liked this.';
    }

    return e(
      'button',
      { onClick: () => this.setState({ liked: true }) },
      'Like'
    );
  }
}

const domContainer = document.querySelector('#like_button_container');
const root = ReactDOM.createRoot(domContainer);
root.render(e(LikeButton));
```

```
<!-- Загрузим React. -->
<!-- Примечание: при деплое на продакшен замените «development.js» на «production.min.js». -->
<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>

<!-- Загрузим наш React-компонент. -->
<script src="like_button.js"></script>
```

Создание нового приложения react (react из под коробки)

Create React App — удобная среда для изучения React и лучший способ начать создание нового одностраничного приложения на React.

Инструмент настраивает среду для использования новейших возможностей JavaScript, оптимизирует приложение для продакшена и обеспечивает комфорт во время разработки. Вам понадобятся Node.js не ниже версии 14.0.0 и npm не ниже версии 5.6 на вашем компьютере. Для создания проекта выполните команды:

npx create-react-app my-app
cd my-app
npm start

Когда ваше приложение готово к развертыванию в продакшене, запуск команды npm run build создаст оптимизированную сборку вашего приложения в папке build.

Знакомство с JSX

JSX (JavaScript XML) — это синтаксическое расширение для JavaScript, используемое в библиотеке React. Оно позволяет писать HTML-подобный код прямо внутри JavaScript, что упрощает создание и структуру интерфейсов.

Преимущества JSX

- **Упрощение написания кода:** JSX позволяет писать разметку и логику в одном месте, что делает код более читаемым и понятным.
- **Интуитивно понятный синтаксис:** Похожие на HTML конструкции в JSX делают создание интерфейсов более естественным и удобным для разработчиков.
- **Мощные возможности интеграции:** Встроенные выражения и возможность обработки JavaScript-кода прямо в JSX позволяют создавать сложные и динамичные интерфейсы.

Недостатки JSX

- **Изучение и настройка:** JSX требует дополнительного обучения и настройки инструментов, таких как Babel, для работы с ним.
- **Сложность отладки:** Отладка может быть сложнее из-за преобразования JSX в JavaScript. Однако современные инструменты разработчика и поддержка от React упрощают этот процесс.

Как JSX работает

Трансформация JSX:

JSX не является валидным JavaScript-кодом, поэтому его нужно трансформировать в обычный JavaScript с помощью трансляторов, таких как Babel. При трансформации JSX превращается в вызовы React.createElement.

Процесс компиляции:

Трансформация JSX позволяет использовать его как синтаксический сахар, упрощая создание элементов и компонентов React. Браузеры не понимают JSX напрямую, поэтому он должен быть преобразован в стандартный JavaScript.

```
const element = <h1>Hello, world!</h1>;
```

превращается в:

jsx

```
const element = React.createElement('h1', null, 'Hello, world!');
```

Основные особенности JSX

HTML-подобный синтаксис

Встраивание выражений

Атрибуты и пропсы

JSX и компоненты

Обработка событий

Условные операторы и циклы

HTML-подобный синтаксис:

JSX позволяет использовать синтаксис, похожий на HTML, для описания структуры пользовательского интерфейса.

Этот синтаксис выглядит как HTML, но является расширением JavaScript.

```
const element = <h1>Hello, world!</h1>;
```

Встраивание выражений:

В JSX можно встраивать выражения JavaScript внутри фигурных скобок {}. Это позволяет динамически изменять содержимое:

```
const name = 'John';
const greeting = <h1>Hello, {name}!</h1>;
```

Атрибуты и пропсы:

Атрибуты в JSX похожи на атрибуты HTML, но они используются с именами в camelCase.

Также можно передавать пропсы (свойства) в компоненты.

```
const element = <img src="logo.png" alt="Logo" />;
```

```
<MyComponent title="Hello" />
```

JSX и компоненты:

JSX используется для создания компонентов React. Компоненты могут быть функциональными или классовыми:

```jsx
// Функциональный компонент
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

// Классовый компонент
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Обработка событий:

В JSX можно обрабатывать события, используя camelCase синтаксис. Например:

```
function handleClick() {
  alert('Button clicked!');
}

const button = <button onClick={handleClick}>Click me</button>;
```

Условные операторы и циклы:

Условия и циклы не могут быть использованы напрямую в JSX, но их можно обрабатывать внутри JavaScript-кода. Например:

```
const isLoggedIn = true;
const greeting = isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign up.</h1>;
```

Пример использования метода map в JSX

Объяснение примера

Массив данных: Мы имеем массив users, содержащий объекты с id и name.

Метод map: Мы используем метод map, чтобы пройтись по каждому элементу массива users. Для каждого пользователя мы создаем элемент списка (<li>).

Ключи: Внутри map мы добавляем атрибут key к каждому элементу списка. Это помогает React идентифицировать элементы при их изменении, добавлении или удалении, что способствует более эффективному обновлению интерфейса.

JSX внутри map: JSX-выражение внутри метода map позволяет непосредственно создавать элементы на основе данных. В этом примере, user.name отображается в элементе списка.

```jsx
import React from 'react';

function UserList() {
  const users = [
    { id: 1, name: 'Alice' },
    { id: 2, name: 'Bob' },
    { id: 3, name: 'Charlie' }
  ];

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

export default UserList;
```

Ресурсы

React начало - ТЫК

React (hello world) - ТЫК

Добавление react-а на свой сайт - ТЫК

Создание нового приложения react - ТЫК

React MDN документация (обзор) - ТЫК

Официальная документация react (на английском) - ТЫК