**1. Понимание маршрутизации**

Маршрутизация основана на том, что URL вашего приложения отображает разные состояния интерфейса. Например:

/ — главная страница.

/about — страница "О нас".

/contact — страница "Контакты".

Что нужно для маршрутизации:

Обработчик изменений URL (можно использовать window.location).

Логику для отображения компонентов на основе текущего URL.

Способы изменения URL (например, при клике на ссылки).

**2. Реализация маршрутизации без библиотек**

Мы будем использовать:

window.location для получения текущего URL.

window.history.pushState для изменения URL без перезагрузки страницы.

Состояние React (useState и useEffect) для управления компонентами, соответствующими определённым маршрутам.

Пример приложения: Простая маршрутизация

import React, { useState, useEffect } from "react";

**// 1. Создаем основные страницы приложения как отдельные компоненты**

const Home = () => <h1>Главная страница</h1>;

const About = () => <h1>О нас</h1>;

const Contact = () => <h1>Контакты</h1>;

**// 2. Компонент для маршрутизации**

const App = () => {

  // Состояние для текущего пути

  const [currentPath, setCurrentPath] = useState(window.location.pathname);

  // Обновляем состояние при изменении пути

  useEffect(() => {

    const handlePopState = () => {

      setCurrentPath(window.location.pathname);

    };

    // Слушаем изменения в истории (назад/вперед)

    window.addEventListener("popstate", handlePopState);

    return () => {

      // Очищаем слушатель, чтобы избежать утечек памяти

      window.removeEventListener("popstate", handlePopState);

    };

  }, []);

**// Функция для навигации**

  const navigate = (path) => {

    // Меняем URL через history API

    window.history.pushState({}, "", path);

    // Обновляем состояние текущего пути

    setCurrentPath(path);

  };

  // Рендер компонента в зависимости от текущего пути

  const renderComponent = () => {

    switch (currentPath) {

      case "/about":

        return <About />;

      case "/contact":

        return <Contact />;

      default:

        return <Home />;

    }

  };

  return (

    <div>

      {/\* 3. Простейшее меню \*/}

      <nav>

        <button onClick={() => navigate("/")}>Главная</button>

        <button onClick={() => navigate("/about")}>О нас</button>

        <button onClick={() => navigate("/contact")}>Контакты</button>

      </nav>

      {/\* 4. Рендер выбранной страницы \*/}

      <div>{renderComponent()}</div>

    </div>

  );

};

export default App;

3. Как это работает

**Изначальная загрузка:**

   При загрузке приложения React считывает window.location.pathname и отображает соответствующий компонент.

**Обработка изменений URL:**

   При клике на кнопку срабатывает функция navigate, которая:

     Обновляет URL с помощью window.history.pushState.

     Сохраняет новый путь в состоянии currentPath.

**Обновление интерфейса:**

   Когда путь меняется, компонент рендера вызывает renderComponent, который отображает нужный компонент на основе значения currentPath.

**Работа с кнопками "вперед" и "назад":**

   Чтобы React реагировал на изменения в истории браузера, используется слушатель события popstate.

**4. Улучшения**

Хотя этот способ работает, его можно улучшить:

Реализовать поддержку параметров в маршрутах (например, /users/:id).

Выделить маршруты и компоненты в отдельные файлы для удобства модульности.

Добавить компонент обертки `` для обработки маршрутов.

Пример с поддержкой параметров:

const renderComponent = () => {

  if (currentPath.startsWith("/user/")) {

    const userId = currentPath.replace("/user/", "");

    return <h1>Профиль пользователя: {userId}</h1>;

  }

  switch (currentPath) {

    case "/about":

      return <About />;

    case "/contact":

      return <Contact />;

    default:

      return <Home />;

  }

};

Теперь путь /user/123 отобразит текст Профиль пользователя: 123.

**5. Ограничения такого подхода**

Нет вложенных маршрутов. Это можно реализовать, но станет сложнее с увеличением количества страниц.

Нет ленивой загрузки компонентов. Это решается с использованием React.lazy и Suspense.

Требуются дополнительные усилия для сложных маршрутов (например, тех, которые содержат параметры или вложенные маршруты).

Если ваше приложение становится больше и сложнее, всё-таки лучше использовать библиотеки, такие как react-router.

**Заключение**

Использование собственного решения маршрутизации может быть полезным для небольших приложений или обучения, но для большинства проектов рекомендуется использовать проверенные библиотеки. Этот подход показывает, как работают базовые принципы маршрутизации, и позволяет глубже понять механизм работы СПА 