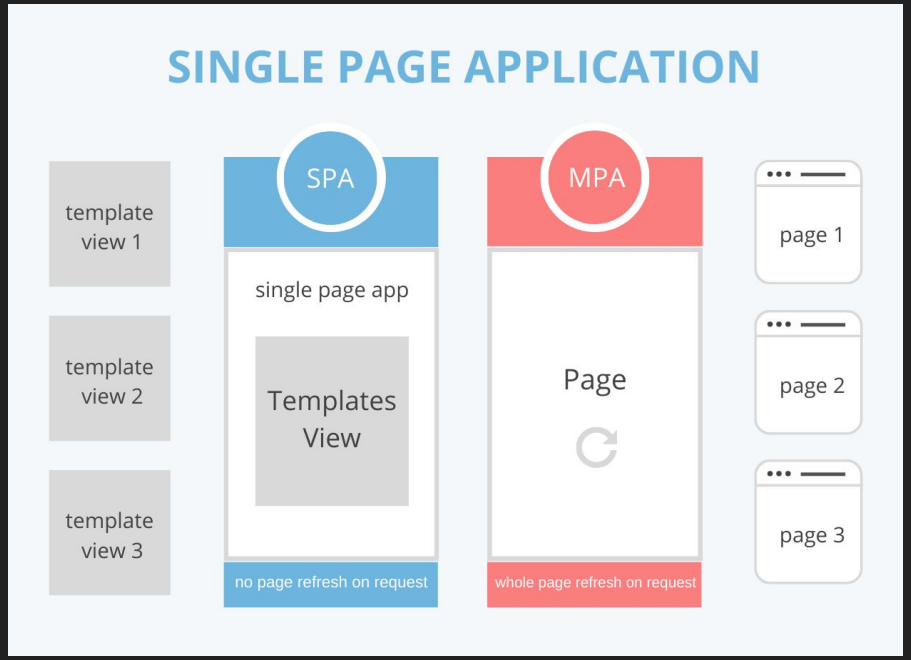
React - Маршрутизация

Маршрутизация в SPA
HTML маршрутизация
Своя библиотека маршрутизации
React Wouter
React Router DOM v.6
Next.js (SSR)

Маршрутизация в SPA (single page application)

Маршрутизация во фронтенде — это процесс управления навигацией пользователя между различными страницами или представлениями веб-приложения без перезагрузки всей страницы. Это позволяет веб-приложению быть более интерактивным и похожим на одностраничные приложения (SPA — Single Page Application), где пользователь может перемещаться между страницами, не обновляя полностью страницу в браузере.



Пример через теги <a href="..."><a/>

```jsx
// Компоненты для разных страниц
const Home = () => <h1>Главная страница</h1>;

const About = () => <h1>О нас</h1>;

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

const Error = () => <h1>404: страница не найдена</h1>;

// Функция для рендеринга нужного компонента в зависимости от пути
function routingFunction(reqPath) {
  switch (reqPath) {
    case "/":
      return <Home />;
    case "/about":
      return <About />;
    case "/contact":
      return <Contact />;
    default:
      return <Error />;
  }
}
```

```jsx
// Основное приложение с ручным роутингом
function AppWrongRouting() {
  // Стейт для хранения текущего пути
  const [currentPath, setCurrentPath] = useState(window.location.pathname);

  // Функция для рендеринга нужного компонента в зависимости от пути
  const renderComponent = useCallback(
    () => routingFunction(currentPath),
    [currentPath]
  );

  // Обновление компонента при изменении пути
  useEffect(() => {
    const handleLocationChange = () => setCurrentPath(window.location.pathname);

    window.addEventListener("popstate", handleLocationChange);

    return () => window.removeEventListener("popstate", handleLocationChange);
  }, []);

  return (
    <div>
      <nav>
        {/* Неправильная навигация через теги <a> */}
        <a href="/">Главная</a>
        <a href="/about">О нас</a>
        <a href="/contact">Контакты</a>
      </nav>

      <div>
        {/* Рендеринг соответствующего компонента */}
        {renderComponent()}
      </div>
    </div>
  );
}
```

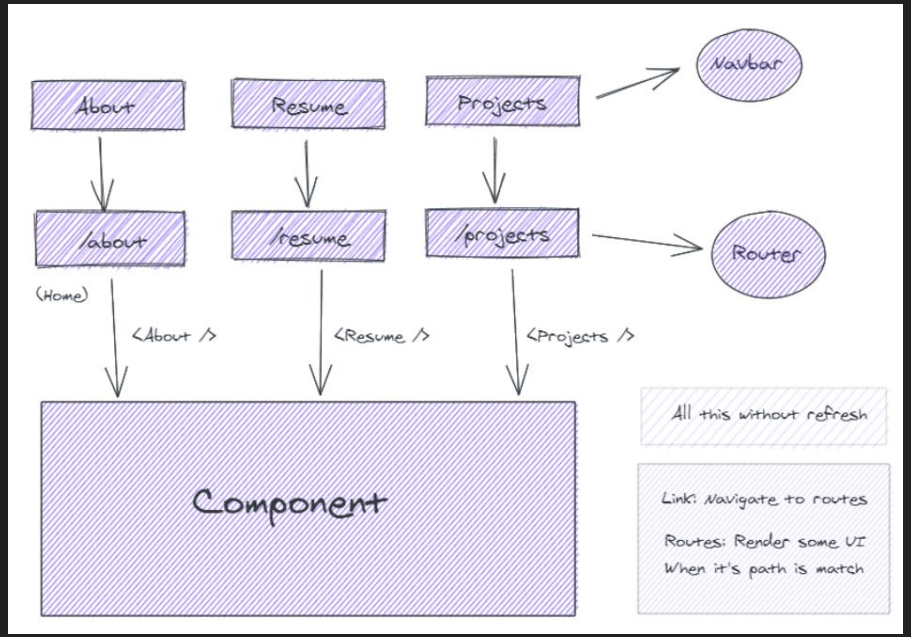
Обновление SPA это плохо

Почему этот пример работает неправильно:

- **Перезагрузка страницы:** При клике на любой из ссылок (<a href="/about">, <a href="/contact">) браузер отправляет новый HTTP-запрос, что приводит к полной перезагрузке страницы. Это нарушает принцип работы SPA, где необходимо обновлять только часть контента без перезагрузки всей страницы.
- **Отсутствие истории:** Без правильного роутинга (как в react-router-dom) здесь вручную обрабатываются только переходы по URL при первой загрузке страницы. Никакого изменения истории (history API) или асинхронной навигации не происходит.
- **Меньшая производительность:** Из-за перезагрузки страницы теряется состояние приложения, что делает работу менее эффективной и снижает производительность.

Маршрутизатор (Router)

Маршрутизатор (router) помогает сопоставить URL-адрес с определённым компонентом или представлением, которое должно быть отображено на странице. Когда пользователь переходит по ссылке или вручную вводит URL в браузере, маршрутизатор перехватывает этот запрос и рендерит соответствующий компонент, не обновляя страницу целиком.



Своя библиотека маршрутизации

- **MyRouter**: Это компонент, который следит за изменением URL и рендерит тот компонент, который соответствует текущему пути.
- **MyRoute**: Это простой компонент, который принимает путь (path) и компонент (component), который нужно рендерить, если путь совпадает.
- **MyLink**: Это компонент, который позволяет навигировать по маршрутам, обновляя URL и вызывая обновление состояния.

Эта мини-библиотека демонстрирует основные принципы маршрутизации: сопоставление путей с компонентами, обработка изменений URL и навигация между страницами. Её реализация проста, но достаточно информативна для понимания того, как работают более сложные роутеры, такие как React Router.

MyLink

Цель: Создает ссылку, которая изменяет URL в браузере без перезагрузки страницы, и обновляет текущее состояние маршрутизатора.

useCallback: Используется для создания функции обработчика клика, которая не пересоздается при каждом рендере.

handleClick: Останавливает стандартное поведение ссылки и изменяет URL в браузере, вызывая событие popstate, которое позволяет MyRouter обновлять отображаемый маршрут.

href: Ссылка отображает URL, на который нужно перейти.

```jsx
import { useCallback } from "react";
import PropTypes from "prop-types";

const MyLink = ({ to, children }) => {
  const handleClick = useCallback(
    (event) => {
      event.preventDefault();

      if (to) {
        window.history.pushState(null, "", to);
        const navEvent = new PopStateEvent("popstate");
        window.dispatchEvent(navEvent);
      }
    },
    [to]
  );

  return (
    <a href={to} onClick={handleClick}>
      {children}
    </a>
  );
};

MyLink.propTypes = {
  to: PropTypes.string.isRequired,
  children: PropTypes.node.isRequired,
};

export default MyLink;
```

MyRoute

Цель: Рендерит компонент в зависимости от текущего маршрута, переданного через path.

cloneElement: Создает новый элемент, копируя переданный компонент и добавляя ему ключ key, который уникален для каждого маршрута.

component: Ожидается, что это React-элемент, который будет отрендерен при совпадении пути.

```
import PropTypes from "prop-types";
import { cloneElement } from "react";

const MyRoute = ({ path, component }) => cloneElement(component, { key: path });

MyRoute.propTypes = {
  path: PropTypes.string.isRequired,
  component: PropTypes.element.isRequired,
};

export default MyRoute;
```

MyRouter

Цель: Определяет, какой компонент должен быть отображен в зависимости от текущего пути, и отображает 404-страницу для несуществующих маршрутов.

useState: Хранит текущий путь в состоянии компонента.

useEffect: Слушает событие popstate, которое происходит при навигации назад или вперед в истории браузера.

useCallback: Создает функцию обработчика для обновления текущего пути при изменении popstate.

useMemo: Оптимизирует поиск подходящего маршрута, кэшируя результат, чтобы избежать повторных вычислений.

Children.toArray: Преобразует children в массив, чтобы легко обрабатывать и искать среди них.

find: Находит первый маршрут, который соответствует текущему пути или является "catch-all" (path="*").

```jsx
import {
  useState,
  useEffect,
  useCallback,
  useMemo,
  Children,
  isValidElement,
} from "react";
import PropTypes from "prop-types";

const MyRouter = ({ children }) => {
  const [currentPath, setCurrentPath] = useState(window.location.pathname);

  const handleLocationChange = useCallback(() => {
    setCurrentPath(window.location.pathname);
  }, []);

  useEffect(() => {
    window.addEventListener("popstate", handleLocationChange);
    return () => {
      window.removeEventListener("popstate", handleLocationChange);
    };
  }, [handleLocationChange]);

  const matchedChild = useMemo(() => {
    return Children.toArray(children).find(
      (child) =>
        isValidElement(child) &&
        (child.props.path === currentPath || child.props.path === "*")
    );
  }, [children, currentPath]);

  return <>{matchedChild || null}</>;
};

MyRouter.propTypes = {
  children: PropTypes.node.isRequired,
};

export default MyRouter;
```

Пример MyRouterDOM

URL изменяется: Когда пользователь нажимает на одну из ссылок (MyLink), URL меняется, но страница не перезагружается. Это позволяет приложению оставаться SPA (Single Page Application).

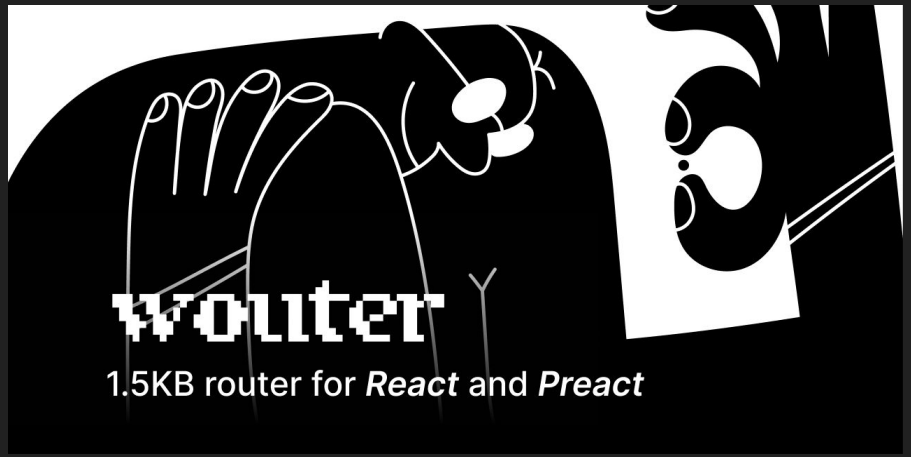
Переход между страницами: MyRouter отслеживает текущий путь в URL и сопоставляет его с путями, указанными в MyRoute. Если путь совпадает, отображается соответствующий компонент.

404 Страница: Если ни один из заданных путей не совпадает с текущим URL, отображается компонент Error, что означает, что страница не найдена.

```jsx
import { MyRoute, MyLink, MyRouter } from "./my-router-dom";

const Home = () => <h1>Главная страница</h1>;
const About = () => <h1>О нас</h1>;
const PostList = () => <h1>Посты</h1>;
const Error = () => <h1>404: страница не найдена</h1>;

function AppWithMyRouterDom() {
  return (
    <div>
      <nav>
        <ul>
          <li>
            <MyLink to="/">Главная</MyLink>
          </li>
          <li>
            <MyLink to="/about">О нас</MyLink>
          </li>
          <li>
            <MyLink to="/post-list">Посты</MyLink>
          </li>
          <li>
            <MyLink to="/asdasd">Не существующая страница</MyLink>
          </li>
        </ul>
      </nav>
      <MyRouter>
        <MyRoute path="/" component={<Home />} />
        <MyRoute path="/about" component={<About />} />
        <MyRoute path="/post-list" component={<PostList />} />
        <MyRoute path="*" component={<Error />} />
      </MyRouter>
    </div>
  );
}

export default AppWithMyRouterDom;
```

React Wouter (аналог)

Wouter — это легковесная библиотека для маршрутизации в React, которая предоставляет минимальный, но функциональный способ управления маршрутами в приложениях React. Она предлагает простой и эффективный API для работы с маршрутами, при этом оставаясь небольшим по размеру и легко интегрируемым в проекты.



Как установить React-Wouter

npm install wouter

Нужен ли React Wouter

Плюсы Wouter:

- **Малый размер:** Wouter очень легковесен (менее 1KB в сжатом виде), что помогает сократить размер вашего бандла.
- **Простота использования:** API Wouter прост и легко интегрируется, что делает его отличным выбором для небольших проектов или тех, кто предпочитает минимализм.
- **Отсутствие зависимости от других библиотек:** Wouter не имеет зависимости от react-router или других библиотек, что делает его менее перегруженным.

Отличная поддержка для хуков: Wouter предоставляет удобные хуки для работы с маршрутами.

Минусы Wouter:

- **Ограниченный функционал:** Wouter может не поддерживать некоторые продвинутые функции, которые есть в более крупных библиотеках, таких как react-router-dom, например, сложную маршрутизацию, авторизацию, защиту маршрутов и т.д.
- **Меньшая поддержка сообщества:** Поскольку Wouter менее популярна, может быть сложнее найти поддержку и примеры по сравнению с более распространёнными библиотеками.

Пример React Wouter

В этом примере мы создаём простое приложение на React, используя библиотеку **Wouter** для маршрутизации.

- **Route**: Определяет, какой компонент рендерить в зависимости от текущего URL.
- **Link**: Создает ссылки, которые изменяют URL и тем самым переключают компоненты.
- **Switch**: Проверяет маршруты один за другим и рендерит первый подходящий компонент.

В <nav> находится список ссылок (<Link>), которые позволяют пользователям перемещаться между страницами приложения. Каждая ссылка изменяет URL и вызывает рендеринг соответствующего компонента.

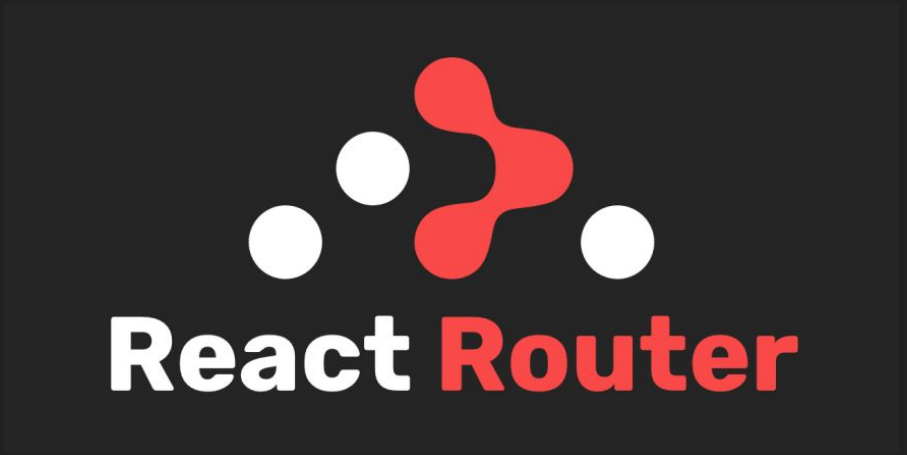
<Switch> проверяет каждый <Route> в порядке их определения и рендерит первый компонент, путь которого совпадает с текущим URL.

Если URL не совпадает ни с одним из указанных путей, рендерится компонент Error, что делает его страницей 404.

```jsx
import { Route, Link, Switch } from "wouter";

// Компоненты для разных страниц
const Home = () => <h1>Главная страница</h1>;
const About = () => <h1>О нас</h1>;
const Contacts = () => <h1>Контакты</h1>;
const Error = () => <h1>404: страница не найдена</h1>;

const AppWithReactWouter = () => (
  <>
    <nav>
      <ul>
        <li>
          <Link to="/">Главная</Link>
        </li>
        <li>
          <Link to="/about">О нас</Link>
        </li>
        <li>
          <Link to="/contacts">Контакты</Link>
        </li>
        <li>
          <Link to="/asdasd">Не существующая страница</Link>
        </li>
      </ul>
    </nav>
    <Switch>
      <Route path="/" component={Home} />
      <Route path="/about" component={About} />
      <Route path="/contacts" component={Contacts} />
      <Route component={Error} />
    </Switch>
  </>
);

export default AppWithReactWouter;
```

React Router DOM

React Router DOM — это библиотека для управления маршрутизацией в приложениях на React. Она позволяет вам создавать одностраничные приложения (SPA) с несколькими страницами или представлениями, управляя тем, какой компонент рендерится в зависимости от текущего URL.



Различия версий 5 и 6

Изменения в API и концепции маршрутизации

- Маршруты и Рендеринг
  Используется только element пропс с JSX для отображения компонентов
- Ключевое понятие "Switch" (Switch был заменен на <Routes>)

Поддержка вложенных маршрутов
Вложенные маршруты поддерживаются напрямую в JSX, что делает их более понятными и легко управляемыми.

Изменение поведения useHistory (useHistory был заменен на useNavigate)

Упрощение Link и NavLink
Поддержка параметров и состояния стала более интуитивной и понятной

Изменение обработки 404 страниц
В версии 6 улучшена поддержка 404 страниц и использования маршрутов, которые не совпадают.

Подходы написания маршрутизации

Компонентный подход: Подходит для декларативного определения маршрутов и навигации, позволяет легко управлять маршрутизацией через JSX и компоненты.

Хуки: Предназначены для программной навигации и получения информации о текущем маршруте. Они позволяют более гибко управлять состоянием маршрутизации внутри компонентов.

Оба подхода могут использоваться совместно в одном проекте, чтобы получить преимущества каждого из них.

Компонентный подход

Этот подход включает использование компонентов, предоставляемых библиотекой, для настройки и управления маршрутизацией. Основные компоненты:

- **<Routes>:** Контейнер для определения маршрутов, заменяет <Switch> в версии 6.
- **<Route>:** Определяет маршрут и связывает его с компонентом, который будет отображаться при совпадении пути.
- **<Link>:** Создает ссылки для навигации.
- **<NavLink>:** Расширяет <Link> и позволяет добавлять активные стили.
- **<Outlet>:** Используется для отображения вложенных маршрутов.

Подход через хуки

Этот подход включает использование хуков для программной навигации и получения информации о маршрутах. Основные хуки:

- **useNavigate:** Хук для программной навигации, позволяет изменять текущий маршрут.
- **useParams:** Хук для извлечения параметров из URL.
- **useLocation:** Хук для получения текущего объекта location, включая путь и состояние.
- **useMatch:** Хук для проверки, совпадает ли текущий URL с шаблоном маршрута.

Пример React-Router-DOM

BrowserRouter компонент, который оборачивает ваше приложение и управляет историей навигации в браузере. Он обеспечивает работу маршрутизации с использованием HTML5 History API. Все навигационные операции, такие как переходы между страницами, будут отслеживаться и управляться этим компонентом.

Компонент **Link** используется для создания навигационных ссылок. Он работает аналогично HTML <a> тегу, но в React Router позволяет навигировать без перезагрузки страницы. Свойство to указывает путь, по которому должна происходить навигация.

Компонент **Routes** содержит набор Route компонентов и отвечает за рендеринг нужного компонента в зависимости от текущего пути.

Компонент **Route** определяет сопоставление между путями и компонентами.

Использование element:

В React Router v6, для отображения компонентов используется свойство element вместо component, как это было в предыдущих версиях. Вы передаете компонент как JSX, например, <Home />, а не как ссылку на функцию Home.

path="*":

Это особый путь, который улавливает все маршруты, которые не были явно определены, и позволяет показать страницу 404.

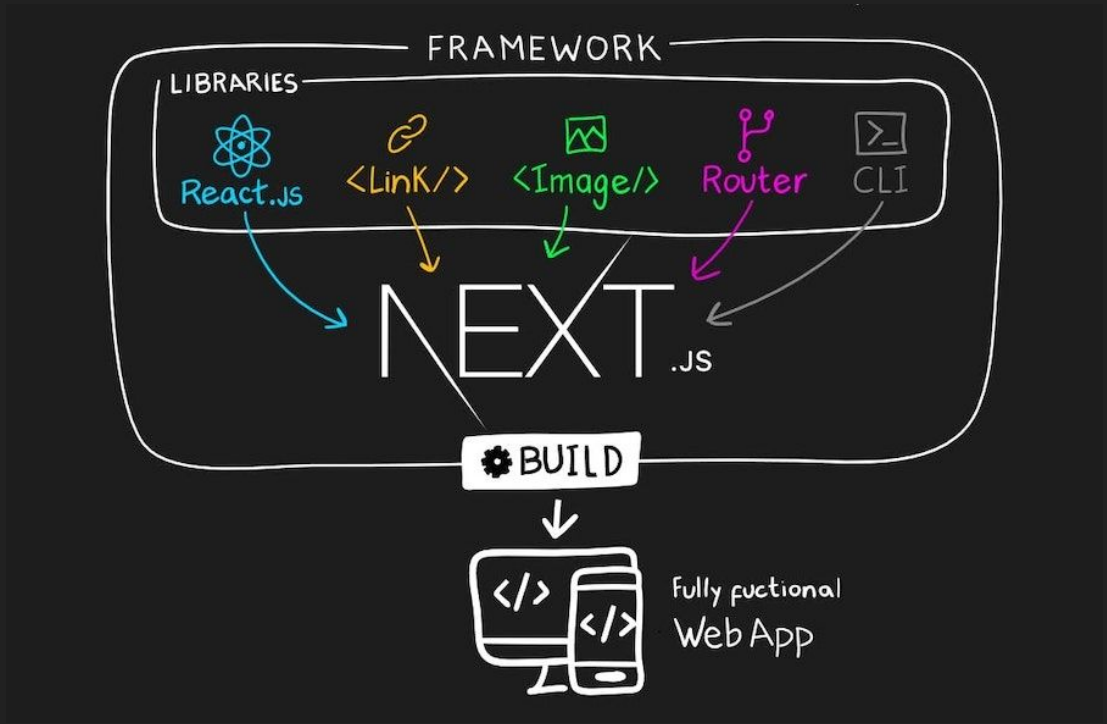
```jsx
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";

// Компоненты для разных страниц
const Home = () => <h1>Главная страница</h1>;
const About = () => <h1>О нас</h1>;
const Contacts = () => <h1>Контакты</h1>;
const Error = () => <h1>404: страница не найдена</h1>;

const AppWithReactRouterDOM = () => (
  <BrowserRouter>
    <nav>
      <ul>
        <li>
          <Link to="/">Главная</Link>
        </li>
        <li>
          <Link to="/about">О нас</Link>
        </li>
        <li>
          <Link to="/contacts">Контакты</Link>
        </li>
        <li>
          <Link to="/asdasd">Не существующая страница</Link>
        </li>
      </ul>
    </nav>
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/contacts" element={<Contacts />} />
      <Route path="*" element={<Error />} />
    </Routes>
  </BrowserRouter>
);

export default AppWithReactRouterDOM;
```

Темы изучения React Router DOM

- **Навигация**
- **Маршруты и параметры**
- **Хуки маршрутизации**
- **Управление состоянием и данными**
- **Ленивая загрузка и Suspense**
- **Обработка ошибок**
- **Переходы и анимации**
- **Авторизация и защищенные маршруты**
- **Оптимизация и производительность**

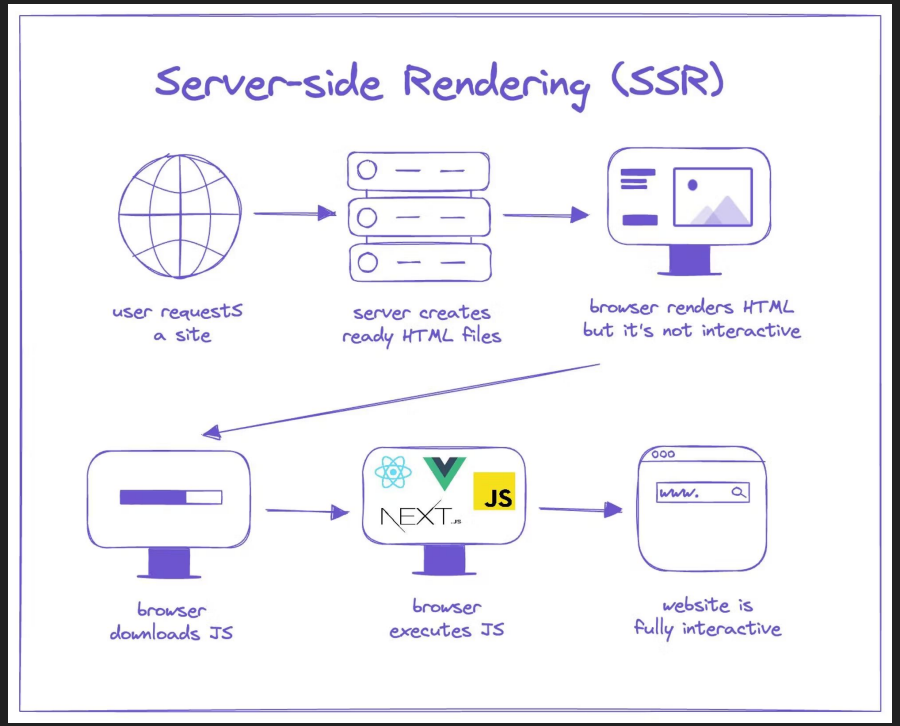
Фреймворк над фреймворком (next.js) (SSR/SSG)

Next.js — это фреймворк для создания React-приложений, который предоставляет мощные возможности для серверного рендеринга (SSR) и статической генерации (SSG). Он был разработан компанией Vercel и стал одним из самых популярных фреймворков для React благодаря своей простоте использования и множеству встроенных функций.



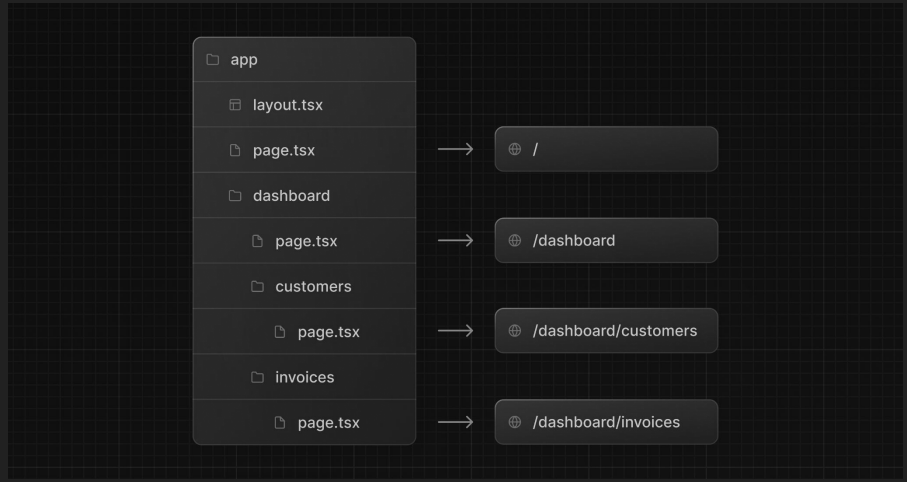
Что такое SSR

SSR (Server-Side Rendering) — это метод рендеринга веб-страниц на сервере перед отправкой их клиенту. Вместо того чтобы отправлять пустую HTML-страницу с пустым содержимым и затем заполнять ее на клиенте с помощью JavaScript, сервер формирует полную HTML-страницу, которую клиент получает уже с готовым контентом.



Маршрутизация в next.js

Маршрутизация в Next.js основана на файловой системе, что делает её очень простой и интуитивно понятной. Это означает, что структура вашего проекта и файлы в директории pages определяют маршруты вашего приложения. Динамические маршруты, API Routes и клиентская навигация через компонент <Link> обеспечивают гибкость и удобство при разработке. Такой подход упрощает создание маршрутов и управление страницами, одновременно предоставляя мощные возможности для оптимизации и расширения вашего приложения.



Ресурсы

Код урока: репозиторий - ТЫК | деплой - ТЫК

React Wouter - ТЫК

React Router DOM v.5 - официальная документация - ТЫК

React Router DOM v.5 - документация на русском - ТЫК

React Router DOM v.6 - официальная документация - ТЫК

React Router DOM v.6 - документация на русском - ТЫК

Next.js - ТЫК