**<https://github.com/michey85/next-blog-app>**

Серверные компоненты появились в 18 реакте, и активно использует Next JS.

Серверные и клиентские компоненты могут біть вложенны друг в друга, но по определенным правилам.

Основую базовую логику реакт выполняет только на сервере.

А на клиент поставляет результат своей работы. Результат конкретного рендеринга.

Чтобы снизить размер бандла серверный компонент хорошо для этого потходит.

### [Правила вложения](https://github.com/michey85/next-blog-app#%D0%BF%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D0%B0-%D0%B2%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F)

* нельзя импортировать серверный компонент внутри клиентского
* можно пробрасывать серверные компоненты в клиентские как children

Используйте клиентские компоненты когда:

* необходимо использовать хуки
* когда необходимы обработчики событий на пользовательские действия
* при использовании браузерного API
* когда используется классовый компонент

Используйте серверные компоненты, когда:

* вы получаете данные через серверное API
* когда нужен прямой доступ к ресурсам бэкенда
* когда используется sensetive информация (ключи API, токены и пр.)
* когда используются тяжелые зависимости

<https://www.youtube.com/watch?v=htcsvLIC1xI&t=43s>

Подсветим пункты меню.

Создадим отдельный компонент для навигации.

Он будет клиентским компонентом.

**Header**

import Link from 'next/link';

import Navigartion from './Navigartion';

const navItems = [

  { label: 'Home', href: '/' },

  { label: 'Blog', href: '/blog' },

  { label: 'About', href: '/about' },

];

export default function TheHeader() {

  return (

    <header>

      <Navigation navLinks={navItems} />

    </header>

  );

}



'use client';

import Link from 'next/link';

import { usePathname } from 'next/navigation'; // хук от NextJS

// по нему мы можем сранивать активное текущее положение

type NavLink = {

  label: string;

  href: string;

};

type Props = {

  // propsom принимаем navLinks

  navLinks: NavLink[];

};

function Navigation({ navLinks }: Props) {

  // console.log(navLinks);

  const pathname = usePathname();

  //console.log(pathname);

  return (

    <>

      {navLinks.map((link) => {

        const isActive = pathname === link.href;

        return (

          <Link

            key={link.label}

            href={link.href}

            className={isActive ? 'active' : ''}

          >

            {link.label}

          </Link>

        );

      })}

    </>

  );

}

export default Navigation;

**Добавим поиск на странице блога**

**1) Вариант**.

Сделаем страницу блона клиентской.

Теперь мы здесь не можем использовать асинхронные функции.

'use client';

import type { Metadata } from 'next';

import Link from 'next/link';

import { useEffect, useState } from 'react';

import { getAllPosts } from '../services/getPosts';

/\*

const a = (info: any) => {

  console.log(info);

};

getAllPosts().then(a);

\*/

export const metadata: Metadata = {

  title: 'Blog | Next App',

  description: 'Blog page',

};

export default function Blog() {

  const [posts, setPosts] = useState<any[]>([]);

  const [loading, setLoading] = useState(true);

  useEffect(() => {

    getAllPosts()

      .then(setPosts)

      .finally(() => setLoading(false));

  }, []);

  return (

    <>

      <h1>Blog page</h1>

      {loading ? (

        <h3>Loading...</h3>

      ) : (

        <ul>

          {posts.map((post: any) => (

            <li key={post.id}>

              <Link href={`blog/${post.id}`}>{post.title}</Link>

            </li>

          ))}

        </ul>

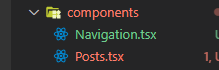
      )}

    </>

  );

}

Вынесем в одельный компонент.



**page.tsxt**

export default function Blog() {

  const [posts, setPosts] = useState<any[]>([]);

  const [loading, setLoading] = useState(true);

  useEffect(() => {

    getAllPosts()

      .then(setPosts)

      .finally(() => setLoading(false));

  }, []);

  return (

    <>

      <h1>Blog page</h1>

      {loading ? <h3>Loading...</h3> : <Posts posts={posts} />}

    </>

  );

}

Posts.tsx

Делать его клиентским ненужно. Пусть будет серверным.

import Link from 'next/link';

type Props = {

  posts: any[];

};

function Posts({ posts }: Props) {

  return (

    <ul>

      {posts.map((post: any) => (

        <li key={post.id}>

          <Link href={`blog/${post.id}`}>{post.title}</Link>

        </li>

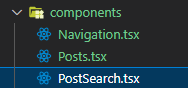
      ))}

    </ul>

  );

}

export default Posts;



'use client';

import { getPostsBySearch } from '@/app/services/getPosts';

import { FormEventHandler, useState } from 'react';

type Props = {

  onSearch: (value: any[]) => void; // функция ничего возвращать не будет

};

function PostSearch({ onSearch }: Props) {

  const [search, setSearch] = useState('');

  const handleSubmit: FormEventHandler<HTMLFormElement> = async (event) => {

    event.preventDefault();

    const posts = await getPostsBySearch(search);

    onSearch(posts); // отправим посты

  };

  return (

    <form onSubmit={handleSubmit}>

      <input

        type="search"

        placeholder="search"

        value={search}

        onChange={(event) => setSearch(event.target.value)}

      />

      <button type="submit">Search</button>

    </form>

  );

}

export default PostSearch;

**page.tsx**

import PostSearch from '@/components/PostSearch';

export default function Blog() {

  const [posts, setPosts] = useState<any[]>([]);

  const [loading, setLoading] = useState(true);

  useEffect(() => {

    getAllPosts()

      .then(setPosts)

      .finally(() => setLoading(false));

  }, []);

  return (

    <>

      <h1>Blog page</h1>

      <PostSearch onSearch={setPosts} />

      {loading ? <h3>Loading...</h3> : <Posts posts={posts} />}

    </>

  );

}

**Вариант 2.**

Установим:

**npm i zustand**

**Это state менеджер. Хорошая альтернатива редаксу.**



import { getAllPosts, getPostsBySearch } from '../services/getPosts';

import { create } from 'zustand';

type UsePosts = {

  posts: any[];

  loading: boolean;

  getAllPosts: () => Promise<void>;

  getPostsBySearch: (value: string) => Promise<void>;

};

export const usePosts = create<UsePosts>()((set) => ({

  posts: [],

  loading: false,

  getAllPosts: async () => {

    set({ loading: true });

    const posts = await getAllPosts();

    set({ posts, loading: false });

  },

  getPostsBySearch: async (search) => {

    set({ loading: true });

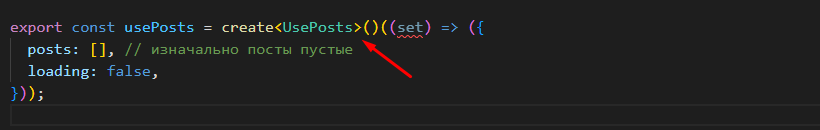
    const posts = await getPostsBySearch(search);

    set({ posts, loading: false });

  },

}));

**Дженерик:**



Правим файл page.tsx:

import type { Metadata } from 'next';

import { Posts } from '@/components/Posts';

import PostSearch from '@/components/PostSearch';

/\*

const a = (info: any) => {

  console.log(info);

};

getAllPosts().then(a);

\*/

export const metadata: Metadata = {

  title: 'Blog | Next App',

  description: 'Blog page',

};

export default function Blog() {

  // const [posts, setPosts] = useState<any[]>([]);

  // const [loading, setLoading] = useState(true);

  // useEffect(() => {

  //   getAllPosts()

  //     .then(setPosts)

  //     .finally(() => setLoading(false));

  // }, []);

  return (

    <>

      <h1>Blog page</h1>

      <PostSearch />

      <Posts />

    </>

  );

}

И изменения в поиске.

Для поиска ненужно пробрасовать функцию:

'use client';

import { usePosts } from '../app/store/index';

import { FormEventHandler, useState } from 'react';

type Props = {

  onSearch: (value: any[]) => void; // функция ничего возвращать не будет

};

function PostSearch() {

  const [search, setSearch] = useState('');

  const getPostsBySearch = usePosts((state) => state.getPostsBySearch);

  const handleSubmit: FormEventHandler<HTMLFormElement> = async (event) => {

    event.preventDefault();

    await getPostsBySearch(search);

  };

  return (

    <form onSubmit={handleSubmit}>

      <input

        type="search"

        placeholder="search"

        value={search}

        onChange={(event) => setSearch(event.target.value)}

      />

      <button type="submit">Search</button>

    </form>

  );

}

export default PostSearch;

**Page.tsx**

Сама страница – это серверный компонент, но внтури два клиентских:

  return (

    <>

      <h1>Blog page</h1>

      <PostSearch />

      <Posts />

    </>

  );

}