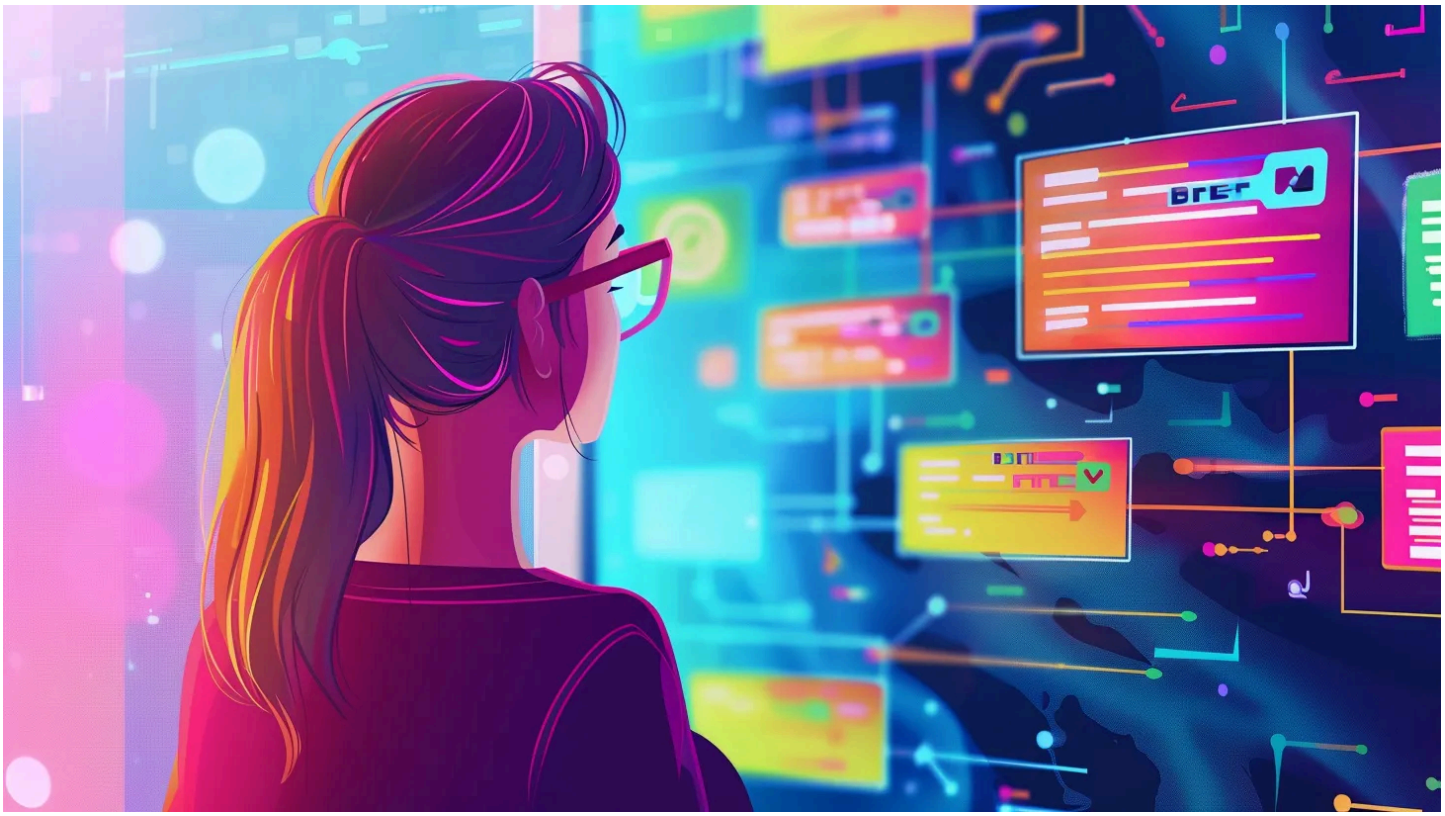
Next.js App Router Routing patterns you should know

Alon Valadji · Follow
Published in Israeli Tech Radar
8 min read · Mar 12, 2024

Listen Share



> *TL;DR grab the code from the GitHub repo: https://github.com/alonronin/nextjs-israel-routing-pattenrs*

Defining a Route

The simplest pattern is just to create a directory inside the app folder with the route name and at that directory create a `page.tsx` file.

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               └── posts
│                   └── page.tsx
└── libste
```

Here is our code to fetch posts and display them:

```
import { ContentWrapper, Title } from '@nrp/components/server';
import Link from 'next/link';

export default async function Page() {
  const posts = await fetch('https://jsonplaceholder.typicode.com/posts').then(
    (res) => res.json(),
  );

  return (
    <ContentWrapper>
      <Title>Posts</Title>

      <ul className="flex flex-col gap-2">
        {posts.map((post: { id: string; title: string }) => (
          <li key={post.id}>
            <Link
              href={`/posts/${post.id}`}
              className="capitalize hover:underline"
            >
              {post.title}
            </Link>
          </li>
        ))}
      </ul>
    </ContentWrapper>
  );
}
```

Dynamic Routes

We want to navigate to a post by its id, for this, we will need to create a dynamic route. For that just create a directory with the square brackets and the name of the param inside and a `page.tsx` file inside that directory as follows:

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               └── posts
│                   ├── [postId]
│                   │   └── page.tsx
│                   └── page.tsx
└── libs
```

Here is the code for our post:

```tsx
import { ContentWrapper, Paragraph, Title } from '@nrp/components/server';

export default async function Page({ params }: { params: { postId: string } })
  await new Promise((resolve) => setTimeout(resolve, 1000));
  const post = await fetch(
    `https://jsonplaceholder.typicode.com/posts/${params.postId}`,
  ).then((res) => res.json());

  return (
    <ContentWrapper>
      <Title className="capitalize">{post.title}</Title>

      <Paragraph className="capitalize">{post.body}</Paragraph>
    </ContentWrapper>
  );
}
```

Catch All and Optional Catch All Routes

To catch all routes from a directory except for the root of that directory's route, we can use the Catch All pattern. We will add a directory with the `[...slug]` bracket and 3 dot annotation, the slug will be our route param in the `params` props, and we'll add our `page.tsx` file to that directory:

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               └── catch-all
│                   └── [...slug]
│                       └── page.tsx
└── libs
```

Here is the code:

```
import { ContentWrapper, Title } from '@nrp/components/server';

export default function Page({ params }: { params: { slug: string[] } }) {
  return (
    <ContentWrapper>
      <Title>From Catch All</Title>
      <pre>{JSON.stringify(params.slug, null, 2)}</pre>
    </ContentWrapper>
  );
}
```

When we navigate to `/catch-all` we'll get a `404` page, however, to `/catch-all/next/page/etc` you'll get the page rendered as expected with array of `params` in the json.

The second pattern allows us to catch the root's directory also. So we'll create a directory with the `[[...slug]]` double brackets and 3 dots annotation, adding `page.tsx` to that folder:

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               └── optional-catch-all
│                   └── [[...slug]]
```

```
│                       └── page.tsx
└── libs
```

And the `page.tsx` code:

```tsx
import { ContentWrapper, Title } from '@nrp/components/server';

export default function Page({ params }: { params: { slug: string[] } }) {
  return (
    <ContentWrapper>
      <Title>From Optional Catch All</Title>

      <pre>{JSON.stringify(params.slug, null, 2)}</pre>
    </ContentWrapper>
  );
}
```

Now the root directory will not return a `404` page but the page's title and empty `params` array. Navigating further will result in the same behavior as previously illustrated.

Nested Layout

We can nest our routing layouts by adding a `layout.tsx` file to our new route. This will nest the current layout inside the parent layout file and display the current page inside the new layout `children`. It is a great pattern to create tabbed navigation, for example.

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               └── nested
│                   ├── password
│                   │   └── page.tsx
│                   ├── layout.tsx
│                   ├── page.tsx
│                   └── tabs.tsx
└── libs
```

Here is the `layout.tsx` code:

```
import { ContentWrapper, Title } from '@nrp/components/server';
import { NavigationTabs } from '@nrp/components';

export default function Layout({ children }: { children: React.ReactNode }) {
  return (
    <ContentWrapper>
      <Title>I am nested layout</Title>
      <NavigationTabs
        items={[
          { title: 'Account', url: '/nested' },
          { title: 'Password', url: '/nested/password' },
        ]}
      />

      {children}
    </ContentWrapper>
  );
}
```

Now our page at the `nested` route will be rendered at the layout's children and also the `password` route. What is great about that is that the layout would not cause a re-render and speed up performance navigating our spa.

If you need to re-render the layout page, consider using the `template` file convention: https://nextjs.org/docs/app/api-reference/file-conventions/template

Here is the `page.tsx` code:

```
import { ContentWrapper, Paragraph, Title } from '@nrp/components/server';

export default function Page() {
  return (
    <ContentWrapper>
      <Title>Account</Title>

      <Paragraph>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. A architecto,
        corporis eos eum facilis incidunt libero perspiciatis provident quae
        quod. Aliquid animi at culpa, hic illo reiciendis similique? Molestiae,
        repudiandae.
      </Paragraph>
```

```
      </ContentWrapper>
    );
}
```

And the password's `page.tsx` code:

```
import { ContentWrapper, Paragraph, Title } from '@nrp/components/server';

export default function Page() {
  return (
    <ContentWrapper>
      <Title>Password</Title>

      <Paragraph>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. sed do eiusmoc
      </Paragraph>
    </ContentWrapper>
  );
}
```

Parallel Routes

This pattern is great to display two, or more, different pages side by side and even create a unique navigation for each page inside the joined parent page.

To do so we need to create a slot first. A slot is a directory with the `@` sign and the name of the slot ie: `@albums` . Inside we'll create again a `page.tsx` file. Let's create the same for `@users` page.

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               └── parallel
│                   ├── @albums
│                   │   └── page.tsx
│                   └── @users
│                       └── page.tsx
└── libs
```

Here is the code for the albums page:

```
import { ContentWrapper, Title } from '@nrp/components/server';

export default async function Page() {
  const albums = await fetch(
    `https://jsonplaceholder.typicode.com/albums`,
  ).then((res) => res.json());

  return (
    <ContentWrapper>
      <Title size="small">Albums</Title>

      <ul>
        {albums.map((album: { id: string; title: string }) => (
          <li key={album.id}>{album.title}</li>
        ))}
      </ul>
    </ContentWrapper>
  );
}
```

And for the users page:

```
import { ContentWrapper, Title } from '@nrp/components/server';
import { Avatar, AvatarFallback, AvatarImage } from '@nrp/components';
import Link from 'next/link';

export default async function Page() {
  const users = await fetch('https://jsonplaceholder.typicode.com/users').then(
    (res) => res.json(),
  );

  return (
    <ContentWrapper>
      <Title size="small">Users</Title>

      <ul className="flex flex-col gap-4">
        {users.map(
          (user: {
            id: string;
            name: string;
            username: string;
            email: string;
          }) => (
            <li key={user.id} className="flex items-center gap-4">
```

```
                <Avatar>
                  <AvatarImage
                    className="bg-foreground"
                    src={`https://robohash.org/${user.username}`}
                    alt="@shadcn"
                  />
                </Avatar>
                <div>
                  <p className="text-sm font-medium leading-none">{user.name}</p>
                  <p className="text-sm text-muted-foreground">{user.email}</p>
                </div>
              </li>
            ),
          )}
        </ul>
      </ContentWrapper>
    );
}
```

Next we'll need to add the slot in our layout. We can add a nested layout inside our `parallel` route or add it to the root layout as well. We create the `layout.tsx` file and a `page.tsx` for our `parallel` page.

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               ├── parallel
│               │   ├── @albums
│               │   │   └── page.tsx
│               │   └── @users
│               │       └── page.tsx
│               ├── layout.tsx
│               └── page.tsx
└── libs
```

Here is the code for the layout:

```
import { Title } from '@nrp/components/server';

export default function Layout({
  children,
```

```
  users,
  albums,
}: {
  children: React.ReactNode;
  users: React.ReactNode;
  albums: React.ReactNode;
}) {
  return (
    <div>
      <Title>Parallel Layout</Title>

      {children}

      <div className="flex gap-4 p-4 justify-around">
        {users}
        {albums}
      </div>
    </div>
  );
}
```

And our `parallel` route page:

```
import { Title } from '@nrp/components/server';

export default function Page() {
  return <Title size="medium">Parallel Routes</Title>;
}
```

Intercepting Routes

Sometimes we want to do a soft route on a page to just peek at it, let's say in a Modal, and have the original route intact (for direct access, full refresh or sharing the links). So this pattern is great just for that.

Having an photos gallery and a photo by id routes we want to open the photo in a Modal dialog at the client but load the photo page at full reload and direct link sharing.

This will be the directory structure for our photo gallery:

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               ├── intercepted
│               │   └── [photoId]
│               │       └── page.tsx
│               └── page.tsx
└── libs
```

Now we want to intercept the `/intercetped/[photoId]` route. Todo so we'll need to create a slot directory and add it to the layout, in the slot directory we'll need to create a directory with the `(.)` prefix that will represent the intercepted route. `(.)` is for current scope, `(..)` for parent scope, `(../..)` for parent's parent scope and `(...)` for root scope.

If it is a nested route we will add the routing directory structure inside this directory without the `(.)` prefix, only the first folder will have it. Let's also add a `layout.tsx` file to hold our `@modal` slot.

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               ├── intercepted
│               │   ├── @modal
│               │   │   └── (..)intercepted
│               │   │       └── [photoId]
│               │   │           └── page.tsx
│               │   └── [photoId]
│               │       └── page.tsx
│               ├── layout.tsx
│               └── page.tsx
└── libs
```

This is the `layout.tsx` file code:

```
import { ReactNode } from 'react';

export default function Layout({
  children,
  modal,
}: {
  children: ReactNode;
  modal: ReactNode;
}) {
  return (
    <>
      {children} {modal}
    </>
  );
}
```

The `/[photoId]/page.tsx` page code:

```
import { ContentWrapper, Title } from '@nrp/components/server';
import { Photo } from '../../components/photo';

export default async function Page({
  params,
}: {
  params: { imageId: string };
}) {
  return (
    <ContentWrapper>
      <Title>Intercepted Route</Title>

      <div className="w-[600px] self-center">
        <Photo imageId={params.imageId} />
      </div>
    </ContentWrapper>
  );
}
```

The intercepted `/@modal/(..)intercepted/[photoId]/page.tsx` page code:

```
import { Photo } from '../../../../components/photo';
import { Suspense } from 'react';
import { Loader2 } from 'lucide-react';
```

```
import { Modal } from '../../../../components/modal';

export default async function Page({
  params,
}: {
  params: { imageId: string };
}) {
  return (
    <Modal title="Intercepted Route">
      <div className="min-h-[100px] flex items-center justify-center">
        <Suspense fallback={<Loader2 className="animate-spin" />}>
          <Photo imageId={params.imageId} />
        </Suspense>
      </div>
    </Modal>
  );
}
```

Now to make it work as expected we also need to add `default.tsx` files in our directory structure to signal next.js what to render in the slot of the layout if nothing was intercepted:

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               ├── intercepted
│               │   ├── @modal
│               │   │   └── (..)intercepted
│               │   │       ├── [photoId]
│               │   │       │   └── page.tsx
│               │   │       └── default.tsx
│               │   └── [photoId]
│               │       ├── default.tsx
│               │       └── page.tsx
│               ├── default.tsx
│               ├── layout.tsx
│               └── page.tsx
└── libs
```

Dynamic Render

Another pattern I like to use, is a Optional Catch All routes and then conditionally render the page, whether is a `params` presented or not. This offers me a way to

handle multiple scenarios in a single page and also render always the pages as I want, even if I share them with direct link access, for example displaying a photo image in a Modal.

Here is the directory structure:

```
nextjs-routing-patterns
├── apps
│   └── blog
│       └── src
│           └── app
│               └── dynamic-render
│                   └── [[...slug]]
│                       └── page.tsx
└── libs
```

And the `page.tsx` code:

```tsx
import { Photo } from '../../components/photo';
import { Suspense } from 'react';
import { Modal } from '../../components/modal';
import { Loader2 } from 'lucide-react';
import { Photos } from '../../components/photos';
import { ContentWrapper } from '@nrp/components/server';

export default async function Page({ params }: { params: { slug: string[] } })
  const [photoId] = params.slug ?? [];

  if (!photoId)
    return (
      <Suspense fallback={'Loading...'}>
        <Photos title="Dynamic Render" page="dynamic-render" />
      </Suspense>
    );

  return (
    <ContentWrapper>
      <Photos title="Dynamic Render" page="dynamic-render" />

      {photoId && (
        <Modal>
          <div className="min-h-[100px] flex items-center justify-center">
            <Suspense fallback={<Loader2 className="animate-spin" />}>
              <Photo imageId={photoId} />
            </Suspense>
```

```
            </div>
          </Modal>
        )}
      </ContentWrapper>
    );
  }
```

Conclusion

So we see how we can use Next.js App's directory routing patterns to achieve different approaches to our navigation system.

I hope that you found this article useful, let me know in the comments below what did you think, your suggestion and love 😍.

Nextjs | React | Server Components | Frontend

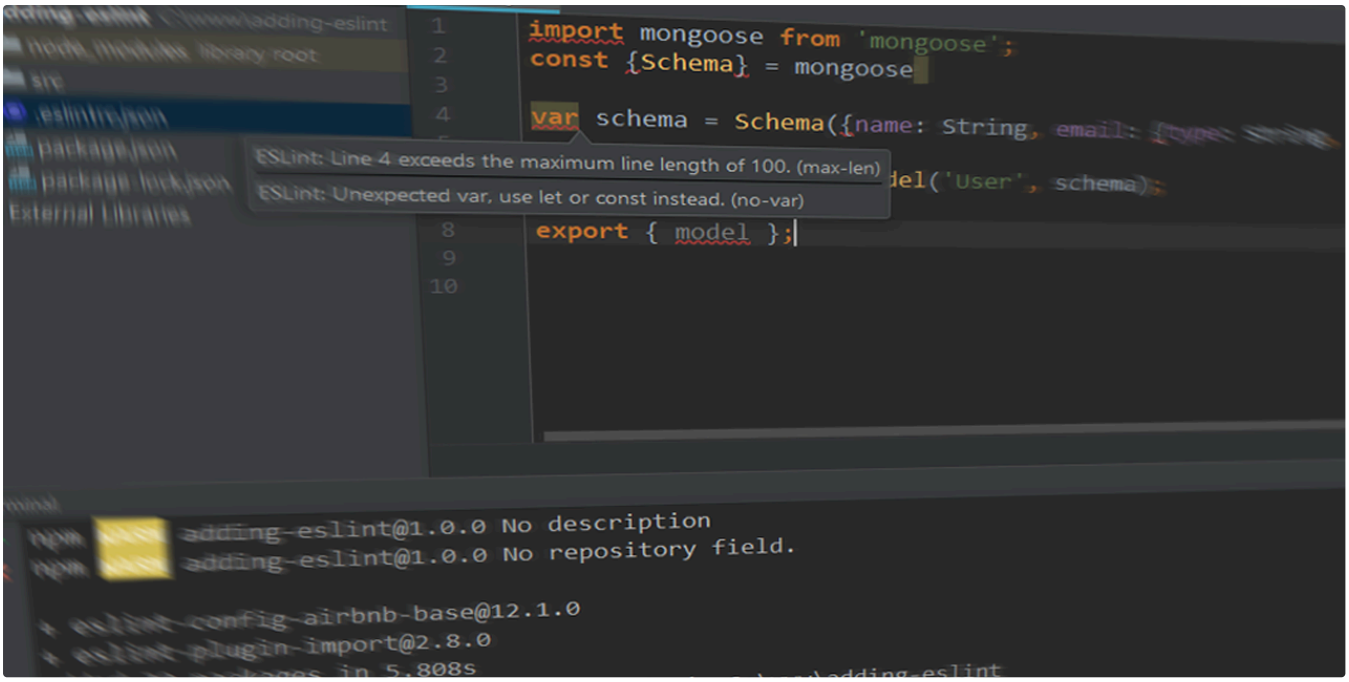
Follow

Written by Alon Valadji

275 Followers · Writer for Israeli Tech Radar

Alon is a leading software architect well-versed in Rozansky and Woods' architectural methodologies. As a full-cycle JavaScript and web architecture torchbearer

More from Alon Valadji and Israeli Tech Radar



Alon Valadji in DailyJS

Adding ESlint to Your Project

In this article I will help you to add ESlint to an existing or a new project with a couple of easy steps using the ESlint CLI tool.

Dec 14, 2017 833 5

Matan Amiel in Israeli Tech Radar

How to create a Monitoring Stack using Kube-Prometheus-Stack (Part 1)

Creating a monitoring stack using Grafana, Prometheus, AlertManager, and integration with Promtail and Loki.

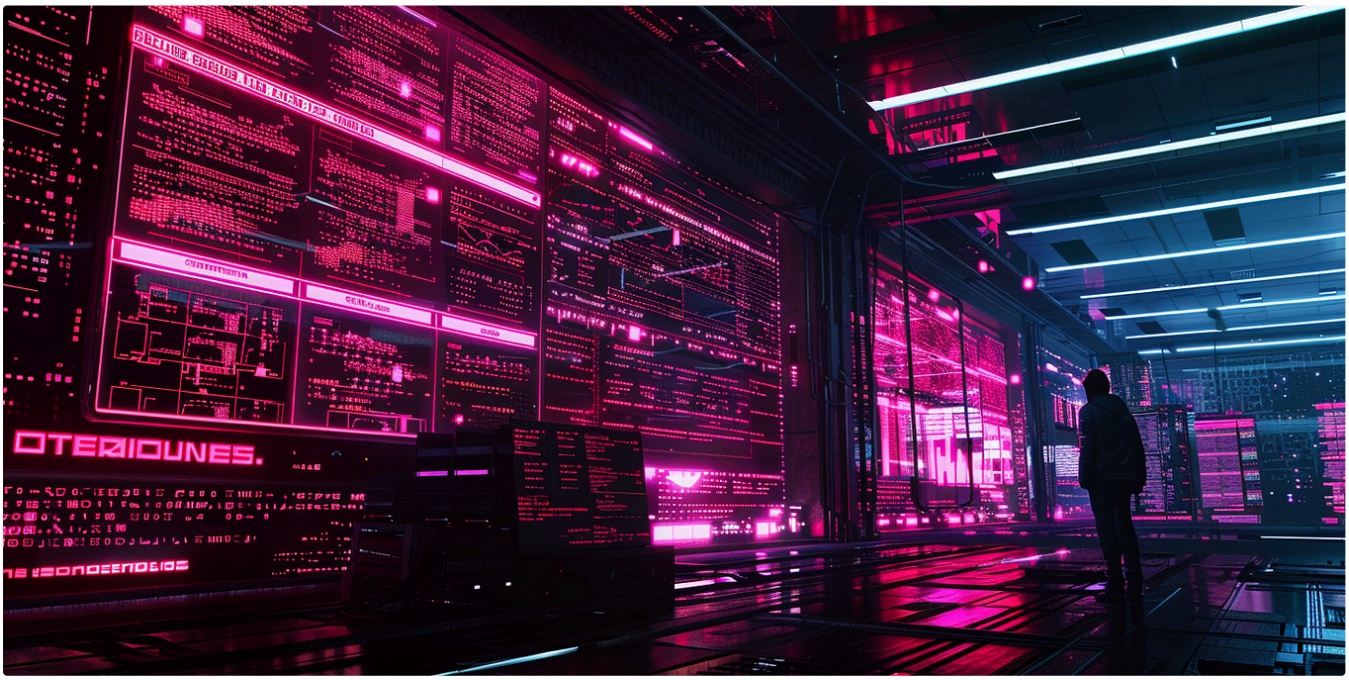
Jun 14, 2023 1.1K 2

Stav Barak in Israeli Tech Radar

OAuth 2.0 and OpenID Connect For Dummies

Many of us struggle with this subject. What's up with that? And what are those, anyway?

Jan 7, 2022 763 1



Alon Valadji in Israeli Tech Radar

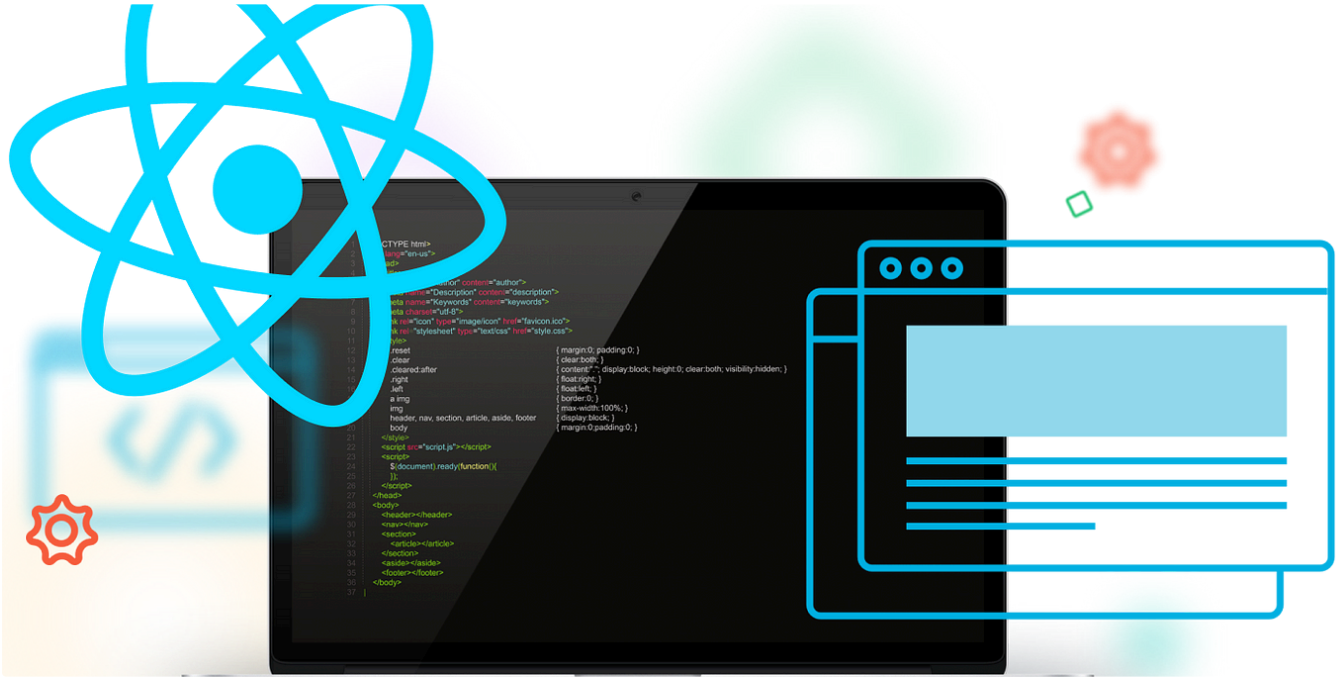
Let's get into the Action, Server Action!

Jun 23 529

See all from Alon Valadji

See all from Israeli Tech Radar

Recommended from Medium



Bryan Aguilar

React Design Patterns

Learn how to apply design patterns in your React applications.

Feb 28 590 3

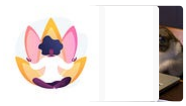
Sviat Kuzhelev

Mastering NextJS Architecture with TypeScript in Mind | Design Abstractions 2024

You'll never know how everything works, but you should understand the system.—by Sviat Kuzhelev.

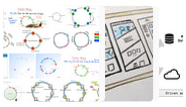
Jun 17 6.8K

Lists



Stories to Help You Grow as a Software Developer

19 stories · 1232 saves

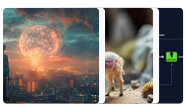


General Coding Knowledge

20 stories · 1430 saves

Medium's Huge List of Publications Accepting Submissions

334 stories · 3171 saves



Natural Language Processing

1604 stories · 1165 saves

Amazon.com Seattle, WA
Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)
- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

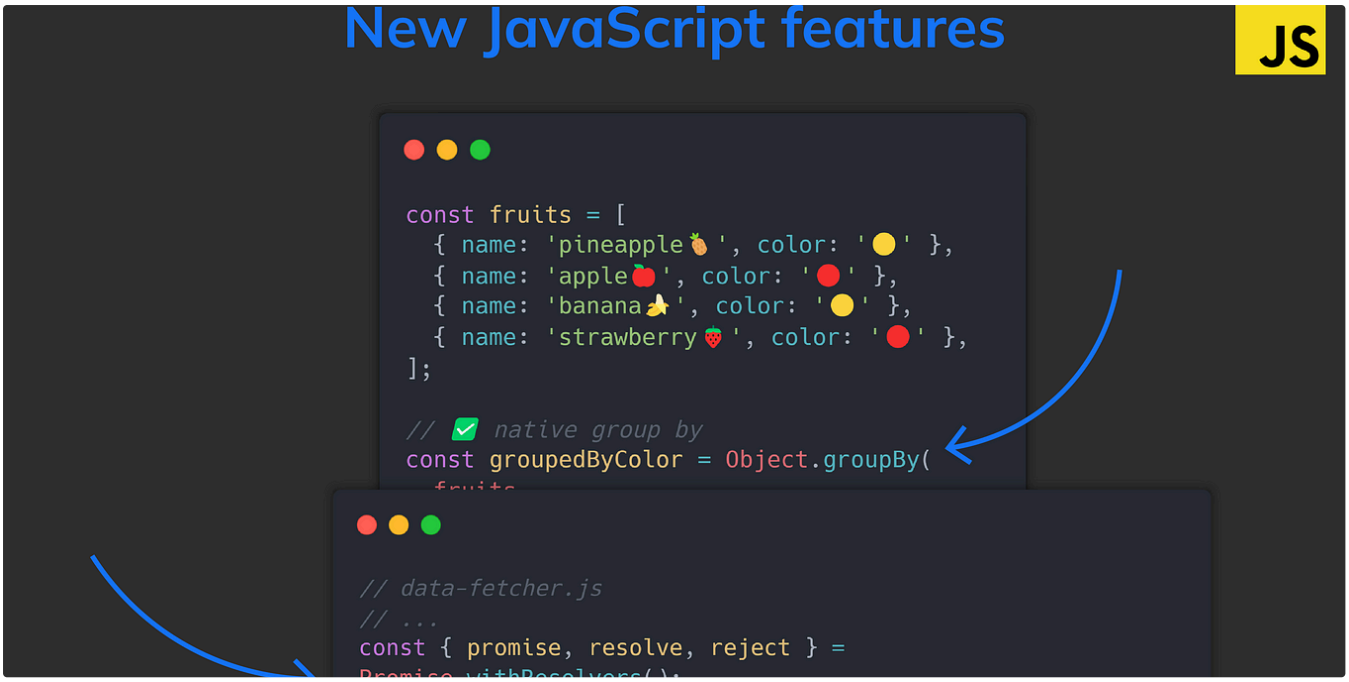
HeatMap (JavaScript)
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

Jun 1 15.1K 231



Tari Ibaba in Coding Beauty

5 amazing new JavaScript features in ES15 (2024)

5 juicy ES15 features with new functionality for cleaner and shorter JavaScript code in 2024.

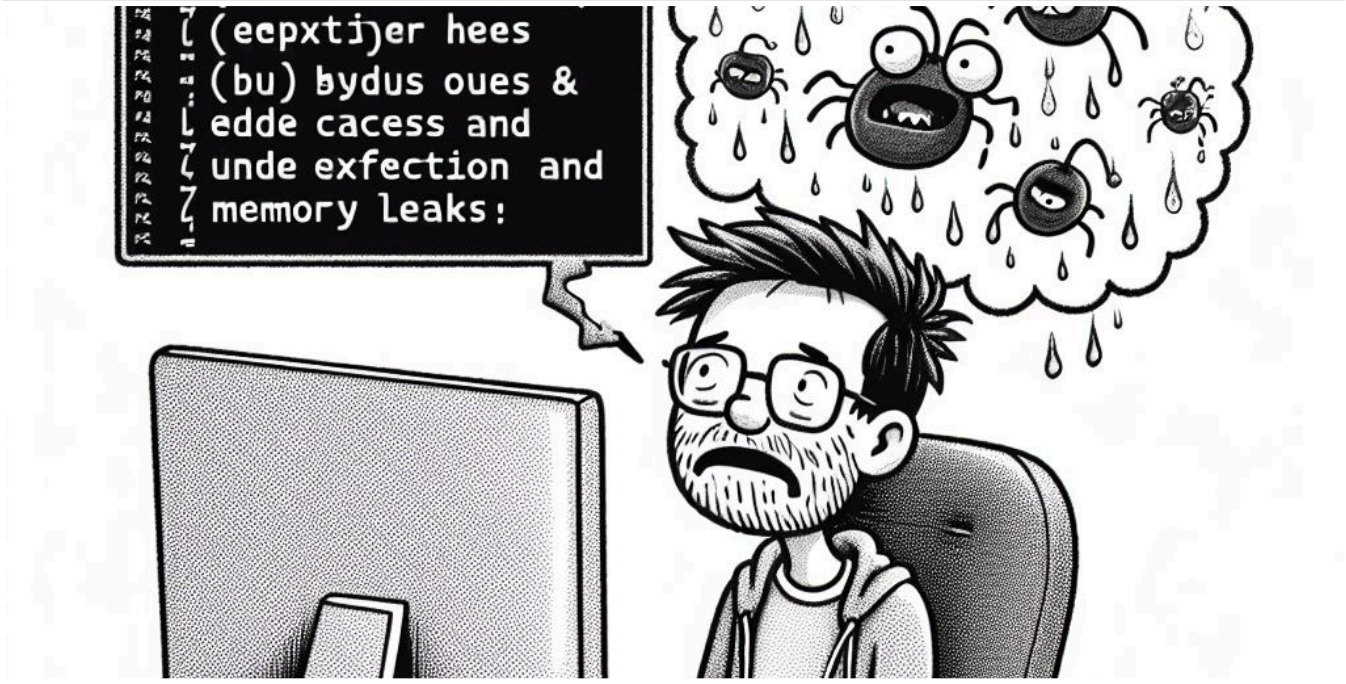
Jun 2 1.93K 10

Mate Marschalko

28 JavaScript One-Liners every Senior Developer Needs to Know

Learn how to implement complex logic with beautifully short and efficient next-level JavaScript syntax.

Mar 1 1.2K 11



Daniel Craciun in Level Up Coding

The Dark Side of useEffect in React

One Mistake away from Disaster

Mar 3 595 9

See more recommendations