Next.js

React apps with the App Router and Server Components.

01 Create an app

```
npx create-next-app@latest my-app
npx create-next-app@latest my-app \
  --typescript --tailwind --src-dir
npx create-next-app@latest my-app --use-pnpm
```

App Router and Turbopack are the defaults in 16. --use-npm, --use-yarn and --use-bun pick other package managers.

02 Special files

page.tsx	Makes the route public
layout.tsx	Wraps pages, keeps state
loading.tsx	Shown while the page loads
error.tsx	Error boundary for the segment
not-found.tsx	404 page
route.ts	API endpoint, no UI
default.tsx	Fallback for a parallel slot

Folders are routes: app/about/page.tsx serves /about.

03 Dynamic routes

```
// app/blog/[slug]/page.tsx
export default async function Page({
  params,
}: { params: Promise<{ slug: string }> }) {
  const { slug } = await params
  return <h1>Post: {slug}</h1>
}
```

params and searchParams are Promises. Next.js 16 removed sync access, so always await them. generateMetadata gets the same params.

04 Route folder names

[...slug]	Catch-all: /shop/a/b
[[...slug]]	Optional: also matches /docs
(marketing)	Group, left out of the URL
@team	Parallel slot, a layout prop
(..)photo	Intercepts a route (modals)

Every @slot needs a default.tsx in 16 or the build fails. (.) (..) and (...) intercept from the same level, one up or the root.

05 Links & navigation

```
import Link from 'next/link'
<Link href={`/blog/${slug}`}>Post</Link>
```

useRouter()	push(), back(), refresh()
usePathname()	Current path, for active links
useSearchParams()	Read the query string

Link prefetches links in the viewport. The hooks come from next/navigation and need 'use client'.

06 Server Components

```
// app/products/page.tsx
export default async function Page() {
  const products = await getProducts()
  return products.map((p: Product) => (
    <p key={p.id}>{p.name}</p>
  ))
}
```

The default: async, can query the database and read secrets, and sends no JavaScript. No hooks or events.

07 Client Components

```
'use client'
export default function Counter() {
  const [count, setCount] = useState(0)
  const add = () => setCount(count + 1)
  return (
    <button onClick={add}>{count}</button>
  )
}
```

For hooks, events and browser APIs; still prerendered on the server. Pass Server Components in as children.

08 Fetching & caching

```
fetch(url)  // uncached by default since 15
fetch(url, { cache: 'force-cache' })
fetch(url, { next: { revalidate: 3600 } })
fetch(url, { next: { tags: ['posts'] } })
```

force-cache keeps a result until revalidated. Requests are deduped per render; use Promise.all for parallel ones.

09 Loading & streaming

```
// app/products/loading.tsx
export default function Loading() {
  return <p>Loading products...</p>
}
<Suspense fallback={<p>Loading reviews</p>}>
  <Reviews />
</Suspense>
```

Suspense streams slow parts in. With cacheComponents on, the static shell is prerendered and the dynamic holes follow (Partial Prerendering).

Next.js

Mutations, endpoints, proxy, images and config.

10 Server Actions

```
'use server'
export async function createPost(
  formData: FormData,
) {
  const title = formData.get('title')
  await db.post.create({ data: { title } })
  revalidatePath('/posts')
  redirect('/posts')
}
```

'use server' on line one makes every export an action; an inline action puts it first in its body. Validate input, for example with zod safeParse.

11 Forms with actions

```
<form action={createPost}>
  <input name="title" required />
  <textarea name="content" />
  <button type="submit">Create</button>
</form>
```

Works before JavaScript loads. The action receives the FormData and reads fields by name.

12 Hooks for actions

useActionState	Returned state, such as errors
useFormStatus	{ pending } in a submit button
useTransition	Pending state for onClick calls
useOptimistic	Show a change before it saves

All from react except useFormStatus (react-dom). useActionState(action, initial) returns [state, formAction].

13 Route handlers

```
// app/api/posts/[id]/route.ts
export async function GET(
  req: Request,
  ctx: { params: Promise<{ id: string }> },
) {
  const { id } = await ctx.params
  return Response.json(await getPost(id))
}
```

Also POST, PUT, PATCH, DELETE. Read a body with await req.json(). GET is uncached by default.

14 proxy.ts (was middleware)

```
export function proxy(req: NextRequest) {
  if (!req.cookies.get('token')) {
    const url = new URL('/login', req.url)
    return NextResponse.redirect(url)
  }
}
export const config = {
  matcher: ['/dashboard/:path*'],
}
```

Lives in the root or src/ and runs before matched requests, on Node.js only. 16 renamed middleware to proxy.

15 Revalidation & after()

```
revalidatePath('/posts')
revalidatePath('/', 'layout') // everything
revalidateTag('posts', 'max')
after(() => sendEmail(post.authorEmail))
```

Call in an action or route handler. revalidateTag needs a profile in 16; after() runs once the response is sent.

16 Images

```
import Image from 'next/image'
import hero from '@/public/hero.jpg'
<>
  <Image src={hero} alt="Hero" preload />
  <Image src="https://example.com/a.jpg"
    alt="Photo" width={800} height={600} />
</>
```

Local imports are sized automatically. Remote images need width, height and an allowed host. preload replaces priority.

17 next.config.ts

```
const nextConfig: NextConfig = {
  cacheComponents: true,
  images: {
    remotePatterns: [
      { hostname: '**.example.com' },
    ],
  },
}
export default nextConfig
```

cacheComponents turns on Partial Prerendering and 'use cache' (it replaced experimental.ppr). Also redirects(), env.

18 Environment variables

```
# .env.local (git ignored)
DATABASE_URL=postgresql://localhost:5432/app
API_SECRET=secret_key_here
NEXT_PUBLIC_API_URL=https://api.example.com
```

Server code reads process.env. Only NEXT_PUBLIC_ values reach the browser; they are inlined at build time. Validate them with zod or t3-env.