Next.js

tsevdos.me / @tsevdos

Rules

Feel free to interrupt me for:

- questions
- relevant comments

Agenda

- next.js 14 (metaframework)
- next.js features
- setup and folder structure
- rendering modes
  - static site generation (SSG)
  - server-side rendering (SSR)
  - client-side rendering (CSR)

Agenda

- routing
  - app router
  - ❌ pages router
- data fetching
- error handling
- API routes (REST API endpoints)
- middleware
- styling
- deploying

Next.js features

- static site generation (SSG)
- server-side rendering (SSR)
- client-Side rendering (CSR) with automatic code splitting
- route handlers (API routes)
- many CSS / theming options (CSS modules, CSS-in-JS, global CSS, etc.)
- data fetching (data at build time or request time)
- image optimization
- built-in CSS and JavaScript bundling
- internationalization (i18n)

Setup

```
npx create-next-app@14

✔ What is your project named? … my-app
✔ Would you like to use TypeScript? … No / ✔ Yes
✔ Would you like to use ESLint? … No / ✔ Yes
✔ Would you like to use Tailwind CSS? … No / ✔ Yes
✔ Would you like to use `src/` directory? … ✔ No / Yes
✔ Would you like to use App Router? (recommended) … No / ✔ Yes
✔ Would you like to customize the default import alias (@/*)? … ✔ No / Yes
Creating a new Next.js app in path/to/my-app.
```

Folder structure (top-level directories)

- [app] ➡️ app router
- [public] ➡️ static assets to be served
- [❌ pages] ➡️ pages router
- [❌ src] ➡️ optional application source folder

Folder structure (top-level files)

- [**package.json**] ➡️ project dependencies and scripts
- [**next.config.js**] ➡️ configuration file for Next.js
- [**tsconfig.json**] ➡️ configuration file for TypeScript
- [**next-env.d.ts**] ➡️ typeScript declaration file for Next.js
- [**.eslintrc.json**] ➡️ configuration file for ESLint
- [**.env, .env.local, .env.production**] ➡️ environment variables for local, production and development environments
- [**.gitignore**] ➡️ git files and folders to ignore
- [**tailwind.config.ts** and **postcss.config.mjs**] ➡️

Folder and file structure

Demo.

Web application fundamentals

- client environment (browser)
- server environment (nodejs server)
- request-response lifecycle
- network boundary

Server components

- static rendering (SSG / build time)
- dynamic rendering (SSR / request time)
- streaming (progressively render UI from the server)
  - (streaming is built into the Next.js app router by default)
- node APIs

Client components

- interactivity
- `"use client"` directive
- browser APIs

How client components work

- full page load
- hydration
- interaction

When to use server and client components

Combining server and client components

- Supported pattern: passing server components to client components as props (and children)
- Unsupported pattern: importing server components into client components

App router conventions - routes and nested routes

- **folder** ➡️ Route segment
- **folder/folder** ➡️ Nested route segment

App router conventions - routing files

- **layout**[.js .jsx .tsx] ➡️ layout
- **page**[.js .jsx .tsx] ➡️ page
- **loading**[.js .jsx .tsx] ➡️ loading UI
- **not-found**[.js .jsx .tsx] ➡️ not found UI
- **error**[.js .jsx .tsx] ➡️ error UI
- **global-error**[.js .jsx .tsx] ➡️ global error UI
- **route**[.js .ts] ➡️ API endpoint
- **template**[.js .jsx .tsx] ➡️ re-rendered layout
- **default**[.js .jsx .tsx] ➡️ parallel route fallback page

App router conventions - dynamic routes

- [folder] ➡️ dynamic route segment
- [...folder] ➡️ catch-all route segment
- [[...folder]] ➡️ optional catch-all route segment

App router conventions - route groups and private folders

- (folder) ➡️ Group routes without affecting routing
- _folder ➡️ Opt folder and all child segments out of routing

App router conventions - parallel and intercepted routes

- @folder ➡️ Named slot
- (.)folder ➡️ Intercept same level
- (..)folder ➡️ Intercept one level above
- (..)(..)folder ➡️Intercept two levels above
- (...)folder ➡️ Intercept from root

Route handlers

- route handlers can be nested inside the app directory, similar to `page[.js .ts]`, but there cannot be a `route[.jsx .tsx]` file and a `page[.js .ts]` at the same route segment
- usually we nest all the route handlers under one `root` directory (ex. api)
- supported HTTP methods: `GET`, `POST`, `PUT`, `PATCH`, `DELETE`, `HEAD`, and `OPTIONS`
- Next.js extends extends the native `Request` and `Response` and provides the `NextRequest` and

Middleware: how to use it

- **file-based configuration**
  - use the file `middleware.ts` (or `.js`) in the root of your project
  - break out middleware functionalities into separate `.ts` or `.js` files and import them into your main `middleware.ts` file
- **conditional execution**
  - use matcher to apply middleware only to specific paths or routes

Middleware: implementation details

- **runs before request handlers**
  - middleware intercepts requests before they reach route handlers
  - ideal for implementing logic like redirects, authentication, or logging
- **request and response manipulation**
  - can rewrite or modify requests and responses dynamically
  - enables functionality like dynamic localization or API routing
- **it's fast**

Middleware: use cases

- **authentication and authorization**
  - validate user sessions or tokens before allowing access to protected pages
  - implement role-based access control efficiently
- **localization**
  - detect user location or preferred language and redirect to the appropriate localized page
- **dynamic rewrites and redirects**
  - alter request URLs dynamically to serve different content without page reloads

Middleware: use cases

- **custom request handling**
  - inject headers, cookies, or metadata into requests for specialized APIs or tracking
- **security enhancements**
  - add security headers, block unwanted bots, or filter malicious requests directly
- **logging and analytics**
  - log request details such as URLs, HTTP methods, user agents, etc.
  - A / B testing, feature flagging, etc.

Metadata file conventions - app icons

- **favicon**[.ico] ➡️ Favicon file
- **icon**[.ico .jpg .jpeg .png .svg] ➡️ App Icon file
- **icon**[.js .ts .tsx] ➡️ Generated App Icon
- **apple-icon**[.jpg .jpeg, .png] ➡️ Apple App Icon file
- **apple-icon**[.js .ts .tsx] ➡️ Generated Apple App Icon

Metadata file conventions - open graph and twitter images

- **opengraph-image**[.jpg .jpeg .png .gif] ➡️Open Graph image file
- **opengraph-image**[.js .ts .tsx] ➡️Generated Open Graph image
- **twitter-image**[.jpg .jpeg .png .gif] ➡️Twitter image file
- **twitter-image**[.js .ts .tsx] ➡️ Generated Twitter image

SEO

- **sitemap**[.xml] ➡ Sitemap file
- **sitemap**[.js .ts] ➡ Generated Sitemap
- **robots**[.txt] ➡ Robots file
- **robots**[.js .ts] ➡ Generated Robots file

CSS styling options

- **global CSS (example)**
  - use `globals.css` for app-wide styles
  - imported in `app/layout.tsx`
- **tailwind CSS (example)**
  - utility-first CSS framework
  - pre-configured support in Next.js
- **CSS in JS (styled components, emotion, etc.)**
  - many libraries with many features
  - dynamic styling

CSS styling options

- **CSS modules (example)**
  - component-scoped styles with `.module.css`
  - automatic class name generation
- **CSS Preprocessors (Sass/SCSS)**
  - write styles using Sass features
  - file extensions: `.module.scss` or `.scss`
- **Vanilla CSS**
  - not very flexible

UI libraries

- simplify UI creation
- pre-built components and design systems
- consistent components and design
- responsive layouts
- accessibility (ARIA-compliant)
- customization (themes, variants, animations, etc.)
- cross-browser compatibility

UI libraries

- Material-UI (MUI)
- Chakra UI
- Ant Design
- Radix UI
- Mantine

Next.js

Demo.

Next.js resources

- react foundations
- learn Next.js
- Next.js documentation

Happy coding!