Next.js

Style Guide

Conventions for writing, structuring, formatting and shipping Next.js 16 apps with the App Router, strict TypeScript, Cache Components and StyleX.

```
async function Invoices({ userId }: InvoicesProps) {
  'use cache';
  cacheLife('hours');
  // Clear beats clever. Measure, then optimize.
}
```

21 chapters · appendix of drop-in configs

Next.js 16.4 | React 19.3 | StyleX

CONTENTS

Inside this guide

01	Purpose and scope	2
02	Baseline and toolchain	3
03	Project layout	5
04	Naming and file conventions	7
05	Formatting and linting	9
06	TypeScript	11
07	Server and Client Components	13
08	Routing and layouts	15
09	Data fetching and caching	17
10	Mutations: Server Actions and Route Handlers	19
11	Components and styling	21
12	State management	24
13	Forms	25
14	Errors and loading states	27
15	Security	28
16	Performance	30
17	Accessibility	32
18	Testing	33
19	Build, configuration and deployment	35
20	Comments and documentation	37
21	Git and review	38
A	Appendix	40

Rule levels MUST MUST NOT SHOULD SHOULD NOT MAY AVOID

01

Purpose and scope

Every Next.js project that adopts this guide uses the App Router on Next.js 16.4 or later, TypeScript in strict mode, and the Cache Components model. The goal is that any two projects read as if one person wrote them.

The guide covers layout, naming, formatting, TypeScript, the server/client split, routing, data and caching, mutations, styling, state, forms, security, performance, accessibility, testing, configuration and commits. It does not cover per-project product or architecture decisions; each project documents those in its own `docs/` folder.

Rule levels. Each rule carries one of these words:

LEVEL	MEANING	BREAKING IT
MUST MUST NOT	Non-negotiable. Tooling or review enforces it.	Needs a written reason in the code and the commit.
SHOULD	The default. Expected in nearly all code.	Fine with a short comment saying why.
MAY	Allowed when it helps; neither encouraged nor discouraged.	Nothing to break.
AVOID	Allowed but a smell.	Fine when the alternative is clearly worse.

Examples. Code samples use a placeholder app called Project with an `invoices` feature. Substitute your own names.

Scope. The Pages Router is out of scope. Existing Pages Router code is migrated route by route, not extended.

When the guide is silent, match the surrounding code, then the official Next.js documentation and React documentation, in that order.

02

Baseline and toolchain

The baseline is Next.js 16.4 with React 19.3, Turbopack, TypeScript in strict mode, pnpm and an LTS release of Node.js. As of 16.4 the Next.js team recommends Cache Components for every app, and they become the default in Next.js 17 (Next.js 16.4 release post).

TOOL	VERSION POLICY
Next.js	16.4 or later, App Router only
React / React DOM	The version Next.js ships with (19.3 for 16.4)
Node.js	Current Active LTS; never below 20.9, the Next.js 16 minimum
TypeScript	Pinned in `package.json`; never below 5.1, the Next.js 16 minimum
Package manager	pnpm, pinned with the `packageManager` field
Bundler	Turbopack (the default since Next.js 16); webpack only with a written reason
Lint / format	ESLint (flat config) and Prettier; `next lint` no longer exists

Rules

- MUST NEXTJS-02.1 enable Cache Components and Partial Prefetching (`cacheComponents: true`, `partialPrefetching: true`) in new apps. Existing apps migrate to them as one tracked task.
- MUST NEXTJS-02.2 pin Node.js in `.nvmrc` and the `engines` field, and pnpm in `packageManager`, so local and CI match.
- MUST NEXTJS-02.3 commit `pnpm-lock.yaml` and install with `pnpm install --frozen-lockfile` in CI.
- MUST NOT NEXTJS-02.4 use features flagged `experimental` in `next.config.ts` in production apps without an ADR naming the flag and the exit plan.
- SHOULD NEXTJS-02.5 upgrade minor Next.js releases within a month and apply security releases within days. `next upgrade` and the codemods do most of the work.
- SHOULD NEXTJS-02.6 enable the React Compiler (`reactCompiler: true`) and drop manual `useMemo` / `useCallback` / `memo` except where a profile shows they still help.

Changes in Next.js 16 this guide assumes (Next.js 16 release post)

CHANGE	WHAT IT MEANS FOR CODE
`params`, `searchParams`, `cookies()`, `headers()`, `draftMode()` are async	Always `await` them
`middleware.ts` renamed to `proxy.ts` (Node.js runtime)	Request interception lives in `proxy.ts` exporting `proxy`
Caching is opt-in with `'use cache'`	Nothing is cached unless marked
`revalidateTag(tag)` needs a profile; `updateTag()` and `refresh()` added	Use `updateTag` in actions for read-your-writes
Parallel route slots need `default.tsx`	Builds fail without one
`next lint` removed	Run ESLint directly
AMP, `serverRuntimeConfig`, `publicRuntimeConfig` removed	Use environment variables

03

Project layout

Routes live in `src/app` and stay thin; the code they render lives in feature folders under `src/features`. A route file composes features, it does not contain business logic.

```
project/
├── next.config.ts
├── package.json                    # engines, packageManager, scripts
├── eslint.config.mjs
├── prettier.config.mjs
├── tsconfig.json
├── .env.example                    # every variable, no real values
├── public/                         # static files served as-is
├── e2e/                            # Playwright end-to-end tests
└── src/
    ├── proxy.ts                    # request interception (auth redirects, rewrites)
    ├── app/
    │   ├── layout.tsx              # root layout: html, body, fonts, providers
    │   ├── not-found.tsx
    │   ├── global-error.tsx
    │   ├── (marketing)/            # route group: public pages, own layout
    │   │   ├── layout.tsx
    │   │   └── page.tsx
    │   ├── (app)/                  # route group: signed-in app shell
    │   │   ├── layout.tsx
    │   │   └── invoices/
    │   │       ├── page.tsx
    │   │       ├── loading.tsx
    │   │       ├── error.tsx
    │   │       └── [invoiceId]/
    │   │           └── page.tsx
    │   └── api/
    │       └── webhooks/stripe/route.ts
    ├── features/
    │   └── invoices/
    │       ├── components/         # invoice-table.tsx, invoice-status-badge.tsx
    │       ├── actions.ts          # 'use server' mutations
    │       ├── queries.ts          # server-only reads, 'use cache' where safe
    │       ├── schemas.ts          # zod schemas and inferred types
    │       └── invoice-table.test.tsx
    ├── components/ui/              # shared design-system primitives
    ├── lib/                        # db client, auth, env, logger
    └── styles/globals.css
```

- MUST NEXTJS-03.1 use the `src/` directory and the App Router. Route files (`page.tsx`, `layout.tsx` and friends) live only under `src/app`.
- SHOULD NEXTJS-03.2 keep route files short: read params, call a feature query, render feature components. Logic moves to `src/features/<feature>/`.
- SHOULD NEXTJS-03.3 use route groups `(name)` to give sections different layouts without changing URLs.
- SHOULD NEXTJS-03.4 colocate everything a feature needs (components, actions, queries, schemas, tests) in its feature folder.
- MUST NEXTJS-03.5 put tests next to the file they test, named `<name>.test.ts` or `<name>.test.tsx`. End-to-end tests live in `e2e/`.
- SHOULD NEXTJS-03.6 keep `src/components/ui` for generic, design-system primitives (button, dialog, input) that know nothing about features.
- AVOID NEXTJS-03.7 `utils`, `helpers` or `common` folders. Code belongs to the feature that uses it; promote it to `src/lib` only when three features need it.
- MUST NEXTJS-03.8 import across the tree with the `@/` alias (`@/features/invoices/queries`), never long relative paths like `../../../lib/db`.
- SHOULD NEXTJS-03.9 prefix folders inside `src/app` that are not routes with `_` (`_components`) when something must be colocated there, so they are never routable.

04

Naming and file conventions

Files and folders are kebab-case, React components are PascalCase, and everything else follows standard TypeScript casing. Next.js special file names are reserved and used only for their purpose.

KIND	STYLE	EXAMPLE
File and folder	kebab-case	`invoice-table.tsx`, `use-debounce.ts`
Route segment	kebab-case; dynamic in brackets, camelCase inside	`invoices/[invoiceId]`
Route group	kebab-case in parentheses	`(marketing)`
React component	PascalCase noun	`InvoiceTable`, `StatusBadge`
Hook	`use` + PascalCase	`useDebouncedValue`
Function, variable	camelCase	`formatCurrency`, `invoiceCount`
Server Action	camelCase verb	`createInvoice`, `markInvoicePaid`
Query function	`get` / `list` + noun	`getInvoice`, `listOverdueInvoices`
Type, interface	PascalCase, no `I` or `T` prefix	`Invoice`, `InvoiceTableProps`
Zod schema	camelCase + `Schema`	`createInvoiceSchema`
Module-level constant	UPPER_SNAKE_CASE for true constants	`MAX_PAGE_SIZE`
Environment variable	UPPER_SNAKE_CASE; `NEXT_PUBLIC_` only if browser-visible	`DATABASE_URL`, `NEXT_PUBLIC_SITE_URL`
Boolean	`is` / `has` / `can` / `should` prefix	`isPaid`, `hasAccess`
Test file	source name + `.test`	`invoice-table.test.tsx`

Next.js special files (reserved names inside `src/app`)

FILE	PURPOSE
`page.tsx`	Makes a segment routable; renders the UI
`layout.tsx`	Shared UI that persists across child routes
`loading.tsx`	Suspense fallback for the segment
`error.tsx`	Error boundary for the segment (Client Component)
`not-found.tsx`	UI for `notFound()`
`global-error.tsx`	Root error boundary, replaces the root layout on error
`route.ts`	Route Handler (HTTP endpoint); never beside a `page.tsx`
`default.tsx`	Fallback for a parallel route slot (required in Next.js 16)
`template.tsx`	Like a layout but re-mounts on navigation

Rules

- MUST NEXTJS-04.1 name a component file after its main component in kebab-case: `InvoiceTable` lives in `invoice-table.tsx`.
- SHOULD NEXTJS-04.2 export one main component per file. Small private sub-components may stay in the same file.
- MUST NEXTJS-04.3 use named exports everywhere except where Next.js requires a default export (pages, layouts, special files, `next.config.ts`).
- MUST NEXTJS-04.4 name props types `<Component>Props`.
- MUST NEXTJS-04.5 write acronyms as one capitalized word: `getHttpClient`, `InvoicePdf`, not `getHTTPClient`.
- SHOULD NEXTJS-04.6 make a name as descriptive as its scope is wide, and never abbreviate by deleting letters (`customer`, not `cstmr`).

05

Formatting and linting

Prettier decides formatting and ESLint catches mistakes; both run in the editor on save and in CI, and nobody argues about either in review. The configs are in the appendix.

TOPIC	RULE
Indentation	2 spaces, never tabs
Line width	100 characters (`printWidth: 100`)
Quotes	Single quotes in TypeScript, double quotes in JSX attributes
Semicolons	Always
Trailing commas	Everywhere they are valid (`trailingComma: 'all'`)
Braces	Same line; always braced, even for one-line `if` bodies
StyleX style keys	Sorted by the `@stylexjs/sort-keys` lint rule
Imports	Grouped and sorted automatically (see below)

Imports

Imports are sorted into four groups separated by blank lines: Node built-ins, external packages (with `react` and `next` first), `@/` aliases, then relative imports. Type-only imports use `import type`.

```
import { Suspense } from 'react';
import Link from 'next/link';
import { notFound } from 'next/navigation';
import { z } from 'zod';

import { InvoiceTable } from '@/features/invoices/components/invoice-table';
import { listInvoices } from '@/features/invoices/queries';
import { Button } from '@/components/ui/button';

import type { InvoiceFilter } from './types';
```

Rules

- MUST NEXTJS-05.1 run `prettier --check` and `eslint --max-warnings 0` in CI. A warning that cannot be fixed is disabled on the narrowest line with a comment saying why.
- MUST NEXTJS-05.2 use the ESLint flat config (`eslint.config.mjs`) with the Next.js plugin's `core-web-vitals` rules, `typescript-eslint` type-aware rules, and the React Hooks rules.
- MUST NOT NEXTJS-05.3 disable a lint rule for a whole file or project to make one warning go away.
- MUST NOT NEXTJS-05.4 use default exports outside the places Next.js requires them; the lint config enforces this.
- MUST NEXTJS-05.5 use `import type` for imports used only as types, so they are erased from the bundle.
- MUST NOT NEXTJS-05.6 use barrel files (`index.ts` that re-exports a folder) inside the app; they slow builds and drag unused code into bundles. Import from the file that defines the thing.

06

TypeScript

TypeScript runs in strict mode with the extra safety flags on, `any` never appears in app code, and every value that crosses a trust boundary (form data, request bodies, search params, environment variables, third-party responses) is parsed with a Zod schema before use.

Compiler settings

- MUST NEXTJS-06.1 enable `strict`, `noUncheckedIndexedAccess`, `noImplicitOverride`, `exactOptionalPropertyTypes` and `verbatimModuleSyntax` (full `tsconfig.json` in the appendix).
- MUST NEXTJS-06.2 keep `pnpm typecheck` (`tsc --noEmit`) clean in CI.

Types

- MUST NOT NEXTJS-06.3 use `any`. Use `unknown` and narrow it, or write the real type.
- MUST NOT NEXTJS-06.4 use non-null assertions (`value!`) or `as` casts to silence the compiler. The exceptions are `as const` and narrowing a value you have just validated.
- SHOULD NEXTJS-06.5 use `type` for unions, props and most shapes, and `interface` only when declaration merging or `extends` chains are needed. Be consistent within a feature.
- SHOULD NEXTJS-06.6 derive types instead of repeating them: `z.infer<typeof schema>`, `ReturnType<typeof fn>`, `Awaited<...>`, `ComponentProps<typeof Button>`.
- SHOULD NEXTJS-06.7 use `satisfies` to check an object against a type while keeping its literal types.
- SHOULD NEXTJS-06.8 model states as discriminated unions (`{ status: 'idle' } | { status: 'error'; message: string }`) rather than several optional flags.
- MUST NEXTJS-06.9 handle every member of a union in a `switch`, ending with an exhaustiveness check.
- AVOID NEXTJS-06.10 `enum`; use a union of string literals or an `as const` object.

Validation at boundaries

- MUST NEXTJS-06.11 parse untrusted input with a Zod schema at the boundary: Server Action arguments, Route Handler bodies, `searchParams`, webhook payloads, and responses from external APIs.
- MUST NEXTJS-06.12 validate environment variables once at startup in `src/lib/env.ts` and import them from there, never from `process.env` directly.
- SHOULD NEXTJS-06.13 keep each schema next to the feature that owns it (`features/invoices/schemas.ts`) and export the inferred type beside it.

```typescript
// src/features/invoices/schemas.ts
import { z } from 'zod';

export const createInvoiceSchema = z.object({
  customerId: z.uuid(),
  dueDate: z.coerce.date(),
  lines: z
    .array(z.object({ description: z.string().min(1), amountCents: z.number().int().positive() }))
    .min(1, 'An invoice needs at least one line'),
});

export type CreateInvoiceInput = z.infer<typeof createInvoiceSchema>;

// Exhaustiveness check
export function statusLabel(status: InvoiceStatus): string {
  switch (status) {
    case 'draft':
      return 'Draft';
    case 'issued':
      return 'Awaiting payment';
    case 'paid':
      return 'Paid';
    default: {
      const unreachable: never = status;
      throw new Error(`Unknown status: ${String(unreachable)}`);
    }
  }
}
```

07

Server and Client Components

Components are Server Components by default, and `'use client'` goes only on the small interactive leaves that need state, effects or browser APIs. The less JavaScript sent to the browser, the faster the page.

Server Components

- SHOULD NEXTJS-07.1 fetch data, read cookies and headers, and talk to the database in Server Components, close to where the data is rendered.
- MUST NEXTJS-07.2 mark modules that must never reach the browser (database clients, secrets, server-side SDKs) with `import 'server-only';` at the top, so a mistaken client import fails the build.
- SHOULD NEXTJS-07.3 pass Server Components into Client Components as `children` or props rather than importing them inside client files, which would turn them into client code.

Client Components

- MUST NEXTJS-07.4 add `'use client'` only to files that need `useState`, `useReducer`, effects, event handlers, browser APIs, or client-only libraries.
- SHOULD NEXTJS-07.5 push the boundary as far down the tree as possible: make the button interactive, not the whole page.
- MUST NEXTJS-07.6 pass only serializable props from server to client (plain data, Server Action references, JSX). Never pass class instances, functions other than Server Actions, or database records with methods.
- MUST NOT NEXTJS-07.7 put secrets, internal IDs that grant access, or full database rows into client props. Select only the fields the UI shows; everything passed is visible in the page payload.
- MUST NOT NEXTJS-07.8 fetch data in `useEffect` when a Server Component could fetch it.
- SHOULD NEXTJS-07.9 use `next/dynamic` with `ssr: false` only for components that genuinely cannot render on the server (canvas, map widgets).

```tsx
// src/app/(app)/invoices/[invoiceId]/page.tsx (Server Component)
import { notFound } from 'next/navigation';

import { InvoiceActions } from '@/features/invoices/components/invoice-actions';
import { InvoiceSummary } from '@/features/invoices/components/invoice-summary';
import { getInvoice } from '@/features/invoices/queries';
import { requireUser } from '@/lib/auth';

// A loading.tsx beside this page is the Suspense boundary for the awaited params.
export default async function InvoicePage({ params }: PageProps<'/invoices/[invoiceId]'>) {
  const { invoiceId } = await params;
  const user = await requireUser();
  const invoice = await getInvoice(invoiceId, user.id);
  if (!invoice) {
    notFound();
  }

  return (
    <>
      <InvoiceSummary invoice={invoice} />
      {/* Client leaf: receives only what it needs */}
      <InvoiceActions invoiceId={invoice.id} isPaid={invoice.status === 'paid'} />
    </>
  );
}
```

```tsx
// src/features/invoices/components/invoice-actions.tsx
'use client';

import { useTransition } from 'react';

import { Button } from '@/components/ui/button';

import { markInvoicePaid } from '../actions';

export function InvoiceActions({ invoiceId, isPaid }: InvoiceActionsProps) {
  const [isPending, startTransition] = useTransition();

  return (
    <Button
      disabled={isPaid || isPending}
      onClick={() => startTransition(() => markInvoicePaid(invoiceId))}
    >
      {isPending ? 'Saving…' : 'Mark as paid'}
    </Button>
  );
}

type InvoiceActionsProps = { invoiceId: string; isPaid: boolean };
```

08

Routing and layouts

URLs are designed first and kept stable; layouts hold shared chrome, pages hold content, and every segment that loads data declares its loading and error states.

URLs and segments

- MUST NEXTJS-08.1 use lowercase kebab-case segments that read as nouns: `/invoices/[invoiceId]/edit`, not `/getInvoice?id=`.
- MUST NEXTJS-08.2 `await` `params` and `searchParams` (they are Promises since Next.js 16), and type them with the generated `PageProps<'/route'>` and `LayoutProps<'/route'>` helpers.
- MUST NEXTJS-08.3 validate `searchParams` with a Zod schema before use; they are user input.
- SHOULD NEXTJS-08.4 enable typed routes (`typedRoutes: true`) so `<Link href>` and `router.push` are checked at compile time.
- SHOULD NEXTJS-08.5 implement `generateStaticParams` for dynamic segments whose values are known at build time.

Layouts

- MUST NEXTJS-08.6 keep the root layout minimal: `<html>`, `<body>`, fonts, global providers and global styles.
- SHOULD NEXTJS-08.7 use route-group layouts for each distinct shell (marketing site, signed-in app, admin).
- MUST NOT NEXTJS-08.8 rely on a layout to protect its child pages. Layouts do not re-render on every navigation; check authorization in each page, action and handler (section 15).
- MUST NEXTJS-08.9 export `metadata` or `generateMetadata` from every page that is reachable without signing in, with a unique title and description.

Loading, errors and not found

- SHOULD NEXTJS-08.10 give every segment that fetches slow data a `loading.tsx`, or wrap the slow part in `<Suspense>` with a skeleton that matches the final layout.
- MUST NEXTJS-08.11 give each top-level section an `error.tsx` (a Client Component with a retry button) and the app a `global-error.tsx`.
- MUST NEXTJS-08.12 call `notFound()` when a requested record does not exist, rather than rendering an empty page.

Advanced routing

- MAY NEXTJS-08.13 use parallel routes (`@slot`) and intercepting routes (`(.)photo`) for modals and split views that need their own URLs. Each slot needs a `default.tsx`.
- AVOID NEXTJS-08.14 them for anything a plain component would do; they add files and edge cases.

```
// src/app/(app)/invoices/page.tsx
import { Suspense } from 'react';

import { InvoiceTable } from '@/features/invoices/components/invoice-table';
import { InvoiceTableSkeleton } from '@/features/invoices/components/invoice-table-skeleton';
import { invoiceFilterSchema } from '@/features/invoices/schemas';

export const metadata = { title: 'Invoices' };

// The page shell renders immediately; request data is read inside Suspense.
export default function InvoicesPage({ searchParams }: PageProps<'/invoices'>) {
  return (
    <Suspense fallback={<InvoiceTableSkeleton />}>
      <FilteredInvoices searchParams={searchParams} />
    </Suspense>
  );
}

async function FilteredInvoices({ searchParams }: Pick<PageProps<'/invoices'>, 'searchParams'>) {
  const filter = invoiceFilterSchema.parse(await searchParams);
  return <InvoiceTable filter={filter} />;
}
```

09

Data fetching and caching

With Cache Components, everything runs at request time unless it is explicitly marked `'use cache'`. Data is read on the server in feature query functions, cached deliberately with a lifetime and tags, and streamed behind `<Suspense>` when it is dynamic (caching guide).

Where data is read

- MUST NEXTJS-09.1 read data in Server Components or in query functions in `features/<feature>/queries.ts`, marked `import 'server-only'`.
- MUST NOT NEXTJS-09.2 call your own Route Handlers from Server Components (`fetch('/api/invoices')`); call the query function directly.
- SHOULD NEXTJS-09.3 start independent requests in parallel (`Promise.all`) rather than awaiting them one after another.
- SHOULD NEXTJS-09.4 wrap request-scoped reads in React `cache()` when several components in one render need the same data (for example the current user).

Dynamic data and Suspense

- MUST NEXTJS-09.5 read request-time data (`cookies()`, `headers()`, `params`, `searchParams`, uncached queries) inside a `<Suspense>` boundary or under a `loading.tsx`, so the static shell can be served immediately.
- SHOULD NEXTJS-09.6 place each Suspense boundary around the smallest part that is slow, with a skeleton the same size as the real content.

Caching

- SHOULD NEXTJS-09.7 mark a component or query `'use cache'` when its output depends only on its arguments and changes rarely: product catalogs, published content, reference data.
- MUST NEXTJS-09.8 give every `'use cache'` scope a `cacheLife` profile (`'minutes'`, `'hours'`, `'days'`, `'max'` or a named custom profile) and at least one `cacheTag` for the data it depends on.
- MUST NOT NEXTJS-09.9 read `cookies()`, `headers()` or other request data inside a `'use cache'` scope. Read them outside and pass the values in as arguments; arguments become part of the cache key.
- MUST NOT NEXTJS-09.10 cache per-user data in a shared cache. Pass the user ID as an argument, or do not cache it.

- SHOULD NEXTJS-09.11 name tags `<entity>` and `<entity>:<id>` (`invoices`, `invoice:42`) so invalidation is predictable.
- MAY NEXTJS-09.12 use `export const ensureStatic = 'navigation'` on marketing, docs and blog layouts to fail the build if dynamic content sneaks in.
- MAY NEXTJS-09.13 use `await navigation()` or `await prefetch()` from `next/cache` to keep expensive parts out of link prefetches.

Invalidation

NEED	API	WHERE
User must see their own change immediately	`updateTag('invoice:42')`	Server Actions only
Content can update in the background (stale-while-revalidate)	`revalidateTag('invoices', 'max')`	Server Actions, Route Handlers
Re-render uncached data on the current page	`refresh()`	Server Actions only
Invalidate a whole path	`revalidatePath('/invoices')`	Use sparingly; prefer tags

```
// src/features/invoices/queries.ts
import 'server-only';

import { cacheLife, cacheTag } from 'next/cache';

import { db } from '@/lib/db';

export async function getInvoice(invoiceId: string, userId: string) {
  'use cache';
  cacheLife('hours');
  cacheTag('invoices', `invoice:${invoiceId}`);

  return db.query.invoices.findFirst({ where: (invoice, { and, eq }) => and(eq(invoice.id,
invoiceId), eq(invoice.ownerId, userId)) });
}
```

10

Mutations: Server Actions and Route Handlers

Server Actions handle mutations from the app's own UI; Route Handlers serve everything else that speaks HTTP: webhooks, third-party callers, file downloads. Both are public endpoints, so every one of them validates input and checks authorization itself.

Server Actions

- MUST NEXTJS-10.1 define actions in `features/<feature>/actions.ts` with `'use server'` at the top of the file, not inline inside components.
- MUST NEXTJS-10.2 treat every action as a public POST endpoint: anyone can call it with any arguments. Authenticate, authorize the specific record, and parse arguments with Zod at the start of every action.
- MUST NEXTJS-10.3 return a typed result (`{ ok: true, data } | { ok: false, error, fieldErrors? }`) for expected failures such as validation errors, rather than throwing. Throw only for unexpected failures.
- MUST NEXTJS-10.4 invalidate what changed before returning: `updateTag` for data the user is looking at, `revalidateTag(tag, 'max')` for data others see.
- SHOULD NEXTJS-10.5 call `redirect()` after a successful create when the user should land on the new record. Call it outside `try` / `catch`; it works by throwing.
- MUST NOT NEXTJS-10.6 return more data than the UI needs; the return value is sent to the browser.

Route Handlers

- SHOULD NEXTJS-10.7 use Route Handlers only for callers that are not the app's own React UI: webhooks, public APIs, OAuth callbacks, file and feed responses.
- MUST NEXTJS-10.8 verify webhook signatures before reading the payload.
- MUST NEXTJS-10.9 parse request bodies and query strings with Zod and return proper status codes (400 invalid input, 401/403 auth, 404, 409 conflict, 422, 500).
- MUST NOT NEXTJS-10.10 place a `route.ts` and a `page.tsx` in the same segment.

```ts
// src/features/invoices/actions.ts
'use server';

import { updateTag } from 'next/cache';
import { redirect } from 'next/navigation';
import { z } from 'zod';

import { requireUser } from '@/lib/auth';
import { db } from '@/lib/db';

import { createInvoiceSchema } from './schemas';

export type ActionResult = { ok: true } | { ok: false; error: string; fieldErrors?: Record<string,
string[]> };

export async function createInvoice(_prev: ActionResult, formData: FormData): Promise<ActionResult>
{
  const user = await requireUser();
  const parsed = createInvoiceSchema.safeParse(Object.fromEntries(formData));
  if (!parsed.success) {
    return { ok: false, error: 'Check the highlighted fields', fieldErrors:
z.flattenError(parsed.error).fieldErrors };
  }

  const invoice = await db.invoices.create({ ...parsed.data, ownerId: user.id });
  updateTag('invoices');
  redirect(`/invoices/${invoice.id}`);
}

export async function markInvoicePaid(invoiceId: string): Promise<ActionResult> {
  const user = await requireUser();
  const invoice = await db.invoices.findOwned(invoiceId, user.id);
  if (!invoice) {
    return { ok: false, error: 'Invoice not found' };
  }

  await db.invoices.markPaid(invoice.id);
  updateTag(`invoice:${invoice.id}`);
  return { ok: true };
}
```

11

Components and styling

Components are small, typed, and composed from design-system primitives; all styling is written in StyleX, which compiles styles to atomic CSS at build time with no runtime cost (StyleX Next.js guide).

Components

- SHOULD NEXTJS-11.1 keep a component under about 150 lines and focused on one job. Split when a component renders unrelated sections or mixes data loading with layout.
- MUST NEXTJS-11.2 type props with a `<Component>Props` type and destructure them in the signature.
- SHOULD NEXTJS-11.3 compose with `children` and slot props rather than adding boolean props for every variant. More than three or four boolean props is a sign the component should split.
- MUST NEXTJS-11.4 give list items a stable `key` from the data, never the array index when the list can reorder.
- SHOULD NEXTJS-11.5 build feature components from `src/components/ui` primitives rather than raw HTML styled from scratch.
- MUST NOT NEXTJS-11.6 define components inside other components; they re-mount on every render.

StyleX setup

- MUST NEXTJS-11.7 compile StyleX with `@stylexjs/babel-plugin` and `@stylexjs/postcss-plugin`, configured in `babel.config.js` and `postcss.config.js` (appendix). This works with Turbopack from Next.js 16.0.3.
- MUST NEXTJS-11.8 keep a single CSS entry (`src/styles/globals.css`) containing `@stylex;` once, imported from the root layout, with `useCSSLayers: true` so StyleX rules layer predictably over resets.
- MUST NEXTJS-11.9 run `@stylexjs/eslint-plugin` with `valid-styles`, `no-unused`, `valid-shorthands` and `sort-keys`. The compiler accepts invalid styles; the linter is what catches them.
- SHOULD NEXTJS-11.10 know that a `babel.config.js` makes Next.js compile app code with Babel instead of SWC. Builds are slower; this is the accepted cost of StyleX (and the React Compiler uses Babel anyway).

Writing styles

- MUST NEXTJS-11.11 define styles with `stylex.create` at module level, below the component that uses them, and apply them with `{...stylex.props(...)}`.

- MUST NOT NEXTJS-11.12 use `className` strings, inline `style` objects for static styles, CSS Modules, or another CSS-in-JS library alongside StyleX. Inline `style` is allowed only for truly dynamic values (a measured width), and dynamic StyleX functions are preferred over it.
- MUST NEXTJS-11.13 use design tokens from `stylex.defineVars` for colors, spacing, radii, type sizes, shadows and z-index. Hard-coded hex values and pixel sizes in feature code are not allowed.
- MUST NEXTJS-11.14 put tokens in files named `*.stylex.ts` (for example `src/styles/tokens.stylex.ts`), exporting only `defineVars` / `defineConsts` results; the compiler requires this.
- SHOULD NEXTJS-11.15 create themes (dark mode, brand variants) with `stylex.createTheme` over the same variables, applied at the layout level.
- SHOULD NEXTJS-11.16 write conditional styles per property: `color: { default: tokens.text, ':hover': tokens.textStrong, '@media (prefers-color-scheme: dark)': tokens.textDark }`.
- SHOULD NEXTJS-11.17 let components accept a `style?: stylex.StyleXStyles` prop for caller overrides instead of exposing many visual props, and merge it last: `stylex.props(styles.base, style)`.
- SHOULD NEXTJS-11.18 name style keys by role (`root`, `header`, `label`, `active`), not by appearance (`blueText`, `bigPadding`).

```
// src/styles/tokens.stylex.ts
import * as stylex from '@stylexjs/stylex';

export const colors = stylex.defineVars({
  surface: '#FFFFFF',
  text: '#0E1116',
  textMuted: '#5B6476',
  accent: '#3D5AFE',
  danger: '#B4232A',
});

export const space = stylex.defineVars({
  xs: '4px',
  sm: '8px',
  md: '16px',
  lg: '24px',
});
```

```
// src/features/invoices/components/invoice-status-badge.tsx
import * as stylex from '@stylexjs/stylex';

import { colors, space } from '@/styles/tokens.stylex';

export function InvoiceStatusBadge({ status, style }: InvoiceStatusBadgeProps) {
  return (
    <span {...stylex.props(styles.root, status === 'overdue' && styles.overdue, style)}>
      {statusLabel(status)}
    </span>
  );
}

type InvoiceStatusBadgeProps = { status: InvoiceStatus; style?: stylex.StyleXStyles };

const styles = stylex.create({
  root: {
    borderRadius: 999,
    color: colors.textMuted,
    fontSize: 12,
    paddingBlock: space.xs,
    paddingInline: space.sm,
  },
  overdue: {
    color: { default: colors.danger, ':hover': colors.text },
  },
});
```

12

State management

Most state is not client state. Data from the database lives on the server and is re-rendered after mutations; state a user should be able to share or bookmark lives in the URL; only short-lived UI state lives in React.

KIND OF STATE	WHERE IT LIVES	EXAMPLE
Server data	Server Components and `'use cache'` queries	The invoice list
Shareable view state	URL search params	Filters, sort order, page number, selected tab
Form input and pending status	The form and `useActionState`	Create-invoice form
Local UI state	`useState` / `useReducer` in the component	An open dropdown, a hover preview
Cross-component UI state	React context near the components that use it	A multi-step wizard
Global client state	A small store (Zustand) only when the above do not fit	A long-lived editor canvas

- MUST NOT NEXTJS-12.1 copy server data into client state and keep it in sync by hand. Mutate with a Server Action and let the server re-render.
- SHOULD NEXTJS-12.2 keep filters, sorting, pagination and selected tabs in the URL, read on the server through `searchParams` and updated on the client with `useRouter` / `<Link>`.
- SHOULD NEXTJS-12.3 use `useOptimistic` for instant feedback on mutations instead of maintaining a parallel client copy of the data.
- SHOULD NEXTJS-12.4 put context providers as deep in the tree as possible, never wrapping the whole app for state one feature uses.
- AVOID NEXTJS-12.5 global client stores. Adding one needs a reason in an ADR.
- MUST NOT NEXTJS-12.6 use `useEffect` to derive state from props or other state; compute it during render.
- SHOULD NEXTJS-12.7 use `useEffectEvent` for non-reactive logic inside effects instead of disabling the exhaustive-deps lint rule.

13

Forms

Forms post to Server Actions through the `action` prop, so they work before JavaScript loads, and use `useActionState` to show pending state and errors. Validation runs on the server every time; client checks only make feedback faster.

- MUST NEXTJS-13.1 submit forms with `<form action={serverAction}>` and read the result with `useActionState`, not with `onSubmit` handlers that call `fetch`.
- MUST NEXTJS-13.2 validate on the server with the same Zod schema as the action, even when the client also validates. Client validation is a convenience, never a guarantee.
- SHOULD NEXTJS-13.3 use native HTML constraints (`required`, `type="email"`, `min`, `maxLength`) for instant client feedback.
- MUST NEXTJS-13.4 show field errors next to their fields and link them with `aria-describedby`, and show a form-level error for anything else.
- MUST NEXTJS-13.5 disable the submit button and show a pending label while the action runs (`isPending` from `useActionState`, or `useFormStatus` inside a submit-button component).
- SHOULD NEXTJS-13.6 keep the user's input after a failed submit by returning it from the action or using `defaultValue` from the previous state.
- SHOULD NEXTJS-13.7 use a form library (React Hook Form) only for forms with complex client behavior: dynamic field arrays, multi-step flows, live cross-field validation.

```
// src/features/invoices/components/create-invoice-form.tsx
'use client';

import { useActionState } from 'react';
import * as stylex from '@stylexjs/stylex';

import { Button } from '@/components/ui/button';
import { TextField } from '@/components/ui/text-field';
import { space } from '@/styles/tokens.stylex';

import { createInvoice, type ActionResult } from '../actions';

const initialState: ActionResult = { ok: true };

export function CreateInvoiceForm() {
  const [state, formAction, isPending] = useActionState(createInvoice, initialState);
  const fieldErrors = state.ok ? undefined : state.fieldErrors;

  return (
    <form action={formAction} {...stylex.props(styles.form)}>
      <TextField name="customerId" label="Customer" required errors={fieldErrors?.customerId} />
      <TextField name="dueDate" label="Due date" type="date" required errors={fieldErrors?.dueDate}
/>
      {!state.ok && <p role="alert">{state.error}</p>}
      <Button type="submit" disabled={isPending}>
        {isPending ? 'Creating…' : 'Create invoice'}
      </Button>
    </form>
  );
}

const styles = stylex.create({
  form: { display: 'grid', gap: space.md, maxWidth: 480 },
});
```

14

Errors and loading states

Expected failures are values the UI renders; unexpected ones are thrown and caught by the nearest `error.tsx`. Every place that waits shows a skeleton shaped like the content it is waiting for.

Errors

SITUATION	HOW IT IS HANDLED
Record does not exist	`notFound()` → nearest `not-found.tsx`
User may not see this record	`notFound()`, so the record's existence is not revealed
Not signed in	`redirect('/sign-in?next=...')`
Validation failure in an action	Typed result with field errors (section 10)
Unexpected failure while rendering	Thrown; nearest `error.tsx` shows a retry
Failure in the root layout	`global-error.tsx`

- MUST NEXTJS-14.1 give `error.tsx` a short human message, a retry button (`reset`), and no stack trace or raw error text. The error `digest` may be shown so support can find the log entry.
- MUST NEXTJS-14.2 report unexpected errors to the error tracker (Sentry or equivalent) through `instrumentation.ts` / `onRequestError`, not by adding `try` / `catch` to every component.
- MUST NOT NEXTJS-14.3 catch an error only to render nothing. Either handle it with a visible state or let it reach the boundary.
- MUST NEXTJS-14.4 keep error messages shown to users free of internal details: table names, IDs of other users, stack traces.

Loading

- SHOULD NEXTJS-14.5 use skeletons that match the final layout's size, so the page does not jump when data arrives.
- SHOULD NEXTJS-14.6 stream: render the shell immediately and suspend only the parts that wait (section 9).
- MUST NEXTJS-14.7 show pending state on every button that triggers an action, and prevent double submission.
- AVOID NEXTJS-14.8 full-page spinners; they hide content that is already available.

15

Security

Authorization is checked where data is read and written, not only at the edge. `proxy.ts` handles coarse redirects; a data-access layer re-checks the user on every query and action, because pages, actions and handlers can all be reached directly.

Authentication and authorization

- MUST NEXTJS-15.1 route every database read and write through a data-access function that takes the current user and enforces ownership or role (`findOwned(invoiceId, user.id)`).
- MUST NEXTJS-15.2 call `requireUser()` (or equivalent) at the top of every Server Action, Route Handler and page that shows private data. A check in a layout or in `proxy.ts` alone is not enough.
- SHOULD NEXTJS-15.3 use `proxy.ts` only for cheap, coarse decisions: redirect signed-out users away from app routes, set headers, rewrite locales. It must not query the database for authorization.
- MUST NEXTJS-15.4 use an established auth library (Auth.js, Better Auth, Clerk or the platform's identity provider) and never hand-roll session cookies or password hashing.
- MUST NEXTJS-15.5 set session cookies `HttpOnly`, `Secure` and `SameSite=Lax` (or `Strict`).

Secrets and environment variables

- MUST NEXTJS-15.6 keep secrets in server-only variables (no `NEXT_PUBLIC_` prefix). Anything prefixed `NEXT_PUBLIC_` is inlined into the browser bundle.
- MUST NEXTJS-15.7 import environment variables only from `src/lib/env.ts`, which validates them with Zod and is marked `server-only` (with a separate validated module for public variables).
- MUST NEXTJS-15.8 list every variable in `.env.example` with a description and no real value. MUST NOT commit `.env` files with real values.

Data exposure

- MUST NEXTJS-15.9 select only the fields a component renders before passing data to a Client Component or returning it from an action. Use a data-transfer type, never the raw database row.
- SHOULD NEXTJS-15.10 use React's taint APIs (`experimental_taintObjectReference`) only if enabled for the project; otherwise rely on explicit field selection.

Headers and input

- MUST NEXTJS-15.11 send a Content Security Policy with nonces, plus `Strict-Transport-Security`, `X-Content-Type-Options: nosniff`, `Referrer-Policy` and a restrictive `Permissions-Policy`.

- MUST NOT NEXTJS-15.12 use `dangerouslySetInnerHTML` except with content sanitized by a vetted library (DOMPurify), with a comment.
- MUST NEXTJS-15.13 configure `images.remotePatterns` narrowly (exact hosts and paths), never a wildcard host.
- MUST NEXTJS-15.14 rate-limit sign-in, sign-up, password reset and any action that sends email or costs money.
- SHOULD NEXTJS-15.15 apply Next.js security releases within days of publication and subscribe to the Next.js security advisories.

16

Performance

Performance is judged by Core Web Vitals measured on real users, and most of it comes from sending less JavaScript and streaming early. Measure before optimizing, and record the before and after numbers in the commit.

METRIC	TARGET (75TH PERCENTILE)
Largest Contentful Paint (LCP)	under 2.5 s
Interaction to Next Paint (INP)	under 200 ms
Cumulative Layout Shift (CLS)	under 0.1

JavaScript

- SHOULD NEXTJS-16.1 keep `'use client'` boundaries at the leaves (section 7); every client file adds to the bundle.
- SHOULD NEXTJS-16.2 load heavy, below-the-fold or interaction-only components with `next/dynamic` or a dynamic `import()` on demand.
- MUST NEXTJS-16.3 check the bundle analyzer before adding a dependency over about 20 kB gzipped, and prefer a lighter one or a platform API.
- MUST NOT NEXTJS-16.4 import a whole library for one function (`lodash`, icon packs); import the specific module.
- SHOULD NEXTJS-16.5 rely on the React Compiler for memoization and remove manual `useMemo` / `useCallback` unless a profile shows a gain.

Images, fonts and scripts

- MUST NEXTJS-16.6 render images with `next/image`, with explicit `width` and `height` (or `fill` with a sized parent) and a meaningful `sizes` value for responsive images.
- SHOULD NEXTJS-16.7 set `priority` (or `preload`) only on the image that is the LCP element of the page.
- MUST NEXTJS-16.8 load fonts with `next/font`, self-hosted, with `display: 'swap'`, and only the weights in use.
- MUST NEXTJS-16.9 load third-party scripts with `next/script` and the latest-loading strategy that works (`lazyOnload` for analytics and chat widgets).

Rendering

- SHOULD NEXTJS-16.10 make every page with no per-user content fully static or cached, and keep request-time rendering to the parts that need it.
- SHOULD NEXTJS-16.11 use `<Link>` for internal navigation so routes are prefetched, and `prefetch` explicitly on links users are likely to click next.
- MUST NEXTJS-16.12 reserve space for anything that loads late (images, embeds, ads, skeletons) so the layout does not shift.

17

Accessibility

Every page meets WCAG 2.2 level AA. Semantic HTML does most of the work; ARIA fills the gaps only where no native element fits.

- MUST NEXTJS-17.1 use the native element for the job: `<button>` for actions, `<a>` / `<Link>` for navigation, `<label>` for every form control, headings in order (one `<h1>` per page).
- MUST NOT NEXTJS-17.2 attach click handlers to `<div>` or `<span>`. If something is clickable, it is a button or a link.
- MUST NEXTJS-17.3 give every image meaningful `alt` text, or `alt=""` when it is decorative.
- MUST NEXTJS-17.4 make every interaction work with the keyboard alone, with a visible focus style (never `outline: none` without a replacement).
- MUST NEXTJS-17.5 meet contrast ratios of 4.5:1 for body text and 3:1 for large text and UI components. Check the design tokens once, and every use inherits it.
- MUST NEXTJS-17.6 announce form errors and async results: link field errors with `aria-describedby`, and use `role="alert"` or an `aria-live` region for messages that appear after submit.
- MUST NEXTJS-17.7 build dialogs, menus, comboboxes and tabs from an accessible primitive library (Radix, React Aria, Base UI) or the native `<dialog>`, not from scratch.
- SHOULD NEXTJS-17.8 respect `prefers-reduced-motion` for animations and view transitions.
- MUST NEXTJS-17.9 set `lang` on `<html>` and a unique, descriptive `<title>` per page through metadata.
- SHOULD NEXTJS-17.10 move focus to the main heading or content after client navigations that replace the page, and to the first error after a failed submit.
- MUST NEXTJS-17.11 run automated checks (axe through Playwright) in CI, and test new flows with a screen reader before release.

18

Testing

Unit and component tests run with Vitest and Testing Library next to the code they test; user flows run end to end in Playwright against a production build. Async Server Components are covered by end-to-end tests, because unit test runners cannot render them.

LAYER	TOOL	LIVES IN	COVERS
Unit	Vitest	`<name>.test.ts` beside the source	Schemas, formatters, pure logic, query and action functions with fakes
Component	Vitest + Testing Library (jsdom)	`<name>.test.tsx` beside the component	Client Components and synchronous Server Components
End to end	Playwright	`e2e/`	Pages, async Server Components, forms, auth flows, accessibility (axe)

Rules

- MUST NEXTJS-18.1 put each unit or component test next to the file it tests, named `<name>.test.ts(x)`.
- MUST NEXTJS-18.2 query the DOM the way a user would: by role, label and text (`getByRole('button', { name: 'Create invoice' })`), not by test IDs or class names. Use `data-testid` only when nothing else identifies the element.
- MUST NEXTJS-18.3 test Server Actions as functions: call them with a `FormData` and assert on the result and the data-layer fake, including the unauthorized and invalid-input cases.
- SHOULD NEXTJS-18.4 cover each critical user flow (sign in, create, edit, pay, delete) with one Playwright test that runs against `next build && next start`.
- MUST NEXTJS-18.5 keep tests deterministic: fixed clock, seeded data, no real third-party calls (mock them with MSW at the network layer).
- MUST NEXTJS-18.6 add a failing test that reproduces a bug before fixing it.
- SHOULD NEXTJS-18.7 run Playwright's axe check on every page covered by an end-to-end test.
- AVOID NEXTJS-18.8 snapshot tests of whole components; they break on every change and are approved without reading.

```tsx
// src/features/invoices/components/invoice-status-badge.test.tsx
import { render, screen } from '@testing-library/react';
import { describe, expect, it } from 'vitest';

import { InvoiceStatusBadge } from './invoice-status-badge';

describe('InvoiceStatusBadge', () => {
  it('labels an issued invoice as awaiting payment', () => {
    render(<InvoiceStatusBadge status="issued" />);

    expect(screen.getByText('Awaiting payment')).toBeVisible();
  });
});
```

```tsx
// e2e/create-invoice.spec.ts
import AxeBuilder from '@axe-core/playwright';
import { expect, test } from '@playwright/test';

test('creates an invoice and lands on its page', async ({ page }) => {
  await page.goto('/invoices/new');
  await page.getByLabel('Customer').fill('7f1c2b9e-1a2b-4c3d-9e8f-0a1b2c3d4e5f');
  await page.getByLabel('Due date').fill('2026-11-02');
  await page.getByRole('button', { name: 'Create invoice' }).click();

  await expect(page).toHaveURL(/\/invoices\/[\w-]+$/);
  const results = await new AxeBuilder({ page }).analyze();
  expect(results.violations).toEqual([]);
});
```

19

Build, configuration and deployment

One set of `package.json` scripts runs the same checks locally and in CI, configuration lives in typed files, and every environment is built from the same commit with only environment variables changing.

Scripts

SCRIPT	RUNS	IN CI
`pnpm dev`	`next dev` (Turbopack)	No
`pnpm build`	`next build`	Yes
`pnpm lint`	`eslint . --max-warnings 0`	Yes
`pnpm format:check`	`prettier --check .`	Yes
`pnpm typecheck`	`tsc --noEmit`	Yes
`pnpm test`	`vitest run`	Yes
`pnpm e2e`	`playwright test` against `next start`	Yes, on main and release branches

Rules

- MUST NEXTJS-19.1 write the Next.js config as `next.config.ts` typed with `NextConfig`, and keep it small: every option has a comment saying why it is there.
- MUST NEXTJS-19.2 run lint, format check, typecheck, unit tests and build on every pull request; `next build` no longer lints for you.
- MUST NEXTJS-19.3 configure Vitest with the StyleX Babel plugin so components that use StyleX render in tests.
- MUST NEXTJS-19.4 keep all configuration that differs per environment in environment variables, validated at startup (section 15). Removed options like `publicRuntimeConfig` are not used.
- SHOULD NEXTJS-19.5 use `output: 'standalone'` when deploying in a container, with a multi-stage Dockerfile that copies only `.next/standalone`, `.next/static` and `public`.
- SHOULD NEXTJS-19.6 configure a shared cache handler (Redis or the platform's) for `'use cache'` when running more than one instance, so instances do not serve different cached data.

- MUST NEXTJS-19.7 give every deployment a health check route (`/api/health`) that checks dependencies without exposing details.
- SHOULD NEXTJS-19.8 let Renovate or Dependabot propose dependency updates weekly, with Next.js and React updated together.
- MUST NEXTJS-19.9 keep `experimental` flags out of production config unless an ADR names the flag and the plan to remove it.

20

Comments and documentation

Comments explain why, the code explains what. Exported functions, hooks and shared components get a short TSDoc comment; everything else is documented by good names and types.

- SHOULD NEXTJS-20.1 give every exported function, hook, Server Action and `components/ui` primitive a `/** */` TSDoc comment starting with a verb phrase: "Returns the invoices a user can see."
- SHOULD NEXTJS-20.2 document what callers need: required permissions, what happens when nothing is found, caching behavior (`cacheLife`, tags), and side effects (emails sent, cache invalidated).
- MUST NOT NEXTJS-20.3 repeat the type signature in prose (`@param id - the id`). Document a parameter only when its meaning or constraints are not obvious from its name and type.
- MUST NEXTJS-20.4 write implementation comments with `//` and explain why when the code does something surprising: a workaround for a framework bug (with a link), an ordering requirement, a performance trick.
- MUST NEXTJS-20.5 write TODOs as `// TODO: <issue link> - <what needs doing>`.
- AVOID NEXTJS-20.6 commented-out code. Git remembers it.
- MUST NEXTJS-20.7 keep `README.md` current with how to install, run, test and deploy, and the list of environment variables (pointing at `.env.example`).
- SHOULD NEXTJS-20.8 record significant decisions (auth library, database, caching strategy, a global store) as short ADRs in `docs/adr/NNNN-title.md`.

```
/**
 * Returns the invoice if `userId` owns it, or `null`.
 *
 * Cached for hours, keyed by both arguments and tagged `invoice:<id>`;
 * `markInvoicePaid` and `updateInvoice` invalidate it with `updateTag`.
 */
export async function getInvoice(invoiceId: string, userId: string): Promise<InvoiceDetail | null> {
  // ...
}
```

21

Git and review

Commits are small, focused and written in Conventional Commits form. One commit does one thing, passes CI on its own, and can be reverted on its own.

```
<type>(<scope>): <imperative summary, under 72 characters>

<optional body: why the change was needed, what it trades off>
```

TYPE	USE FOR	EXAMPLE
`feat`	New capability	`feat(invoices): add PDF download`
`fix`	Bug fix	`fix(auth): redirect to original page after sign-in`
`perf`	Measured speedup	`perf(invoices): cache list query, LCP 3.1 s to 1.4 s`
`refactor`	No behavior change	`refactor(invoices): move queries out of page files`
`test`	Tests only	`test(invoices): cover unauthorized markInvoicePaid`
`build`	Config, dependencies, CI	`build: upgrade next to 16.4`
`style`	Formatter runs, renames with no logic change	`style: apply prettier`
`docs`	Docs and ADRs	`docs(adr): choose Better Auth`
`chore`	Everything else	`chore: update .env.example`

- MUST NEXTJS-21.1 use the feature folder as the scope (`invoices`, `auth`, `billing`).
- MUST NEXTJS-21.2 keep formatting-only and rename-only changes in their own commits.
- SHOULD NEXTJS-21.3 mark breaking changes with `!` and a `BREAKING CHANGE:` footer.

Review checklist

- [ ] CI is green: lint with zero warnings, format, typecheck, tests, build
- [ ] `'use client'` only on interactive leaves; no server-only code reachable from the client
- [ ] Every action, handler and private page authenticates and authorizes itself
- [ ] All untrusted input parsed with Zod; no `any`, no unchecked casts
- [ ] Client props and action results contain only the fields the UI needs
- [ ] Request data read inside Suspense; `'use cache'` scopes have `cacheLife` and tags
- [ ] Mutations invalidate the right tags
- [ ] Styles use StyleX tokens; no hard-coded colors or spacing
- [ ] Keyboard, labels, alt text and contrast checked
- [ ] New behavior has tests; a bug fix has a regression test
- [ ] Commit messages are Conventional Commits

A

Appendix

Drop-in files for a new project. Install the packages they reference with pnpm, and adjust names to your own.

A. next.config.ts

```ts
import type { NextConfig } from 'next';

// Static security headers for every response (section 15). The Content Security Policy
// needs a fresh nonce per request, so src/proxy.ts sends it instead.
const securityHeaders = [
  { key: 'X-Content-Type-Options', value: 'nosniff' },
  { key: 'Referrer-Policy', value: 'strict-origin-when-cross-origin' },
  {
    key: 'Permissions-Policy',
    value: 'camera=(), microphone=(), geolocation=(), payment=(), usb=(), browsing-topics=()',
  },
  // Production builds only (`next dev` serves plain HTTP). Add `preload` once every
  // subdomain serves HTTPS and the domain is submitted to hstspreload.org.
  ...(process.env.NODE_ENV === 'production'
    ? [{ key: 'Strict-Transport-Security', value: 'max-age=63072000; includeSubDomains' }]
    : []),
];

const nextConfig: NextConfig = {
  // Cache Components programming model (recommended for every app as of 16.4).
  cacheComponents: true,
  partialPrefetching: true,
  // Automatic memoization; replaces most manual useMemo/useCallback.
  reactCompiler: true,
  // Type-checked <Link href> and router.push.
  typedRoutes: true,
  // Container deploys copy only .next/standalone, .next/static and public.
  output: 'standalone',
  // next/image may optimize only these exact remote hosts and paths (section 15).
  images: {
    remotePatterns: [
      { protocol: 'https', hostname: 'images.example.com', pathname: '/uploads/**' },
    ],
  },
  // The static security headers above on every response; src/proxy.ts adds the CSP.
  async headers() {
    return [{ source: '/:path*', headers: securityHeaders }];
  },
};

export default nextConfig;
```

B. tsconfig.json

```json
{
  "compilerOptions": {
    "target": "ES2022",
    "lib": ["dom", "dom.iterable", "esnext"],
    "module": "esnext",
    "moduleResolution": "bundler",
    "jsx": "react-jsx",
    "allowJs": true,
    "esModuleInterop": true,
    "strict": true,
    "noUncheckedIndexedAccess": true,
    "noImplicitOverride": true,
    "exactOptionalPropertyTypes": true,
    "verbatimModuleSyntax": true,
    "isolatedModules": true,
    "resolveJsonModule": true,
    "skipLibCheck": true,
    "noEmit": true,
    "incremental": true,
    "plugins": [{ "name": "next" }],
    "paths": { "@/*": ["./src/*"] }
  },
  "include": [
    "next-env.d.ts",
    "**/*.ts",
    "**/*.tsx",
    ".next/types/**/*.ts",
    ".next/dev/types/**/*.ts"
  ],
  "exclude": ["node_modules"]
}
```

C. eslint.config.mjs

```
import stylex from '@stylexjs/eslint-plugin';
import nextVitals from 'eslint-config-next/core-web-vitals';
import nextTs from 'eslint-config-next/typescript';
import { defineConfig, globalIgnores } from 'eslint/config';

export default defineConfig([
  ...nextVitals,
  ...nextTs,
  {
    plugins: { '@stylexjs': stylex },
    rules: {
      '@stylexjs/valid-styles': 'error',
      '@stylexjs/no-unused': 'error',
      '@stylexjs/valid-shorthands': 'warn',
      '@stylexjs/sort-keys': 'warn',
      '@typescript-eslint/no-explicit-any': 'error',
      '@typescript-eslint/consistent-type-imports': 'error',
      'import/no-default-export': 'error',
      curly: ['error', 'all'],
    },
  },
  {
    // Next.js requires default exports from route and config files.
    files: ['src/app/**/*.tsx', 'src/app/**/*.ts', '*.config.{ts,mts,mjs,js}', 'src/proxy.ts'],
    rules: { 'import/no-default-export': 'off' },
  },
  globalIgnores(['.next/**', 'out/**', 'build/**', 'next-env.d.ts']),
]);
```

D. prettier.config.mjs

```
/** @type {import('prettier').Config} */
const config = {
  printWidth: 100,
  singleQuote: true,
  jsxSingleQuote: false,
  semi: true,
  trailingComma: 'all',
  plugins: ['@ianvs/prettier-plugin-sort-imports'],
  importOrder: [
    '<BUILTIN_MODULES>',
    '^(react|next)(/.*)?$',
    '<THIRD_PARTY_MODULES>',
    '',
    '^@/(.*)$',
    '',
    '^[./]',
  ],
};

export default config;
```

E. StyleX: babel.config.js, postcss.config.js, globals.css

From the StyleX Next.js guide, adjusted for the `src/` layout.

```
// babel.config.js
// CommonJS because Next.js and the StyleX PostCSS plugin load this file with require().
// eslint-disable-next-line @typescript-eslint/no-require-imports -- CommonJS config file
const path = require('path');
const dev = process.env.NODE_ENV !== 'production';

module.exports = {
  presets: ['next/babel'],
  plugins: [
    [
      '@stylexjs/babel-plugin',
      {
        dev,
        runtimeInjection: false,
        enableInlinedConditionalMerge: true,
        treeshakeCompensation: true,
        aliases: { '@/*': [path.join(__dirname, 'src', '*')] },
        unstable_moduleResolution: { type: 'commonJS' },
      },
    ],
  ],
};
```

```
// postcss.config.js
// eslint-disable-next-line @typescript-eslint/no-require-imports -- CommonJS config file
const babelConfig = require('./babel.config');

module.exports = {
  plugins: {
    '@stylexjs/postcss-plugin': {
      include: ['src/**/*.{js,jsx,ts,tsx}'],
      babelConfig: {
        babelrc: false,
        parserOpts: { plugins: ['typescript', 'jsx'] },
        plugins: babelConfig.plugins,
      },
      useCSSLayers: true,
    },
    autoprefixer: {},
  },
};
```

```
/* src/styles/globals.css, imported once from src/app/layout.tsx */
@layer resets {
  *,
  *::before,
  *::after {
    box-sizing: border-box;
    margin: 0;
  }
}

@stylex;
```

F. src/lib/env.ts

```
import 'server-only';

import { z } from 'zod';

const serverEnvSchema = z.object({
  DATABASE_URL: z.url(),
  AUTH_SECRET: z.string().min(32),
  NODE_ENV: z.enum(['development', 'test', 'production']),
});

export const env = serverEnvSchema.parse(process.env);
```

G. src/lib/csp.ts

```
// Builds the per-request Content Security Policy that src/proxy.ts sends (section 15).
// Next.js reads the nonce back out of the request's CSP header while rendering and adds
// it to its own scripts, so the policy must stay in this `'nonce-…'` form.

/** Returns a fresh, unguessable nonce. Call it once per request. */
export function createNonce(): string {
  return btoa(crypto.randomUUID());
}

/**
 * Returns the Content-Security-Policy header value for one request.
 *
 * Scripts need the nonce in every environment; `'strict-dynamic'` lets the scripts Next.js
 * loads with that nonce load their own chunks. Development also allows `'unsafe-eval'`
 * (React rebuilds server error stacks with `eval`) and inline styles (the dev overlay and
 * hot reloading inject `<style>` tags). StyleX compiles to a static stylesheet, so
 * production styles need only `'self'` and the nonce.
 */
export function contentSecurityPolicy(nonce: string, isDev: boolean): string {
  const directives = [
    `default-src 'self'`,
    `script-src 'self' 'nonce-${nonce}' 'strict-dynamic'${isDev ? ` 'unsafe-eval'` : ''}`,
    `style-src 'self' ${isDev ? `'unsafe-inline'` : `'nonce-${nonce}'`}`,
    `img-src 'self' blob: data:`,
    `font-src 'self'`,
    `object-src 'none'`,
    `base-uri 'self'`,
    `form-action 'self'`,
    `frame-ancestors 'none'`,
  ];
  if (!isDev) {
    directives.push('upgrade-insecure-requests');
  }
  return directives.join('; ');
}
```

H. src/proxy.ts

```
import { NextResponse, type NextRequest } from 'next/server';

import { contentSecurityPolicy, createNonce } from '@/lib/csp';

const SIGNED_IN_PREFIXES = ['/invoices', '/settings'];

export function proxy(request: NextRequest) {
  // Coarse redirect only. Every page, action and handler still checks the user itself.
  const { pathname } = request.nextUrl;
  const needsSession = SIGNED_IN_PREFIXES.some(
    (prefix) => pathname === prefix || pathname.startsWith(`${prefix}/`),
  );
  if (needsSession && !request.cookies.has('session')) {
    const signIn = new URL('/sign-in', request.url);
    signIn.searchParams.set('next', pathname);
    return NextResponse.redirect(signIn);
  }

  // A fresh nonce per request. Next.js reads it from the request's CSP header and adds it
  // to its scripts; Server Components can read it from `x-nonce` for `<Script nonce>`.
  const nonce = createNonce();
  const csp = contentSecurityPolicy(nonce, process.env.NODE_ENV === 'development');
  const requestHeaders = new Headers(request.headers);
  requestHeaders.set('x-nonce', nonce);
  requestHeaders.set('Content-Security-Policy', csp);

  const response = NextResponse.next({ request: { headers: requestHeaders } });
  response.headers.set('Content-Security-Policy', csp);
  return response;
}

export const config = {
  matcher: [
    {
      // Every page; not Route Handlers, build assets or optimized images. Prefetches are
      // skipped too: they return no HTML, and the page itself checks the user.
      source: '/((?!api|_next/static|_next/image|favicon.ico).*)',
      missing: [
        { type: 'header', key: 'next-router-prefetch' },
        { type: 'header', key: 'purpose', value: 'prefetch' },
      ],
    },
  ],
};
```

I. Starting a new project

1. Run `pnpm create next-app@latest` with TypeScript, ESLint, the App Router and `src/`; decline Tailwind.

2. Copy the files above, install StyleX (`@stylexjs/stylex`, `@stylexjs/babel-plugin`, `@stylexjs/postcss-plugin`, `@stylexjs/eslint-plugin`), Zod, Vitest, Testing Library and Playwright.
3. Pin Node.js in `.nvmrc` and `engines`, and pnpm in `packageManager`.
4. Create `src/styles/tokens.stylex.ts` with the design tokens and `src/lib/env.ts` with the variables, and fill in `.env.example`.
5. Add the scripts from section 19 and a CI workflow that runs them on every pull request.
6. Write `docs/adr/0001-project-scope.md` saying what the app is and is not.