Frontend folder structure — Headly (Next.js `pages` router, ISR + Preview)

দারুণ — নিচে পুরো **frontend-**ইর ফোল্ডার স্ট্রাকচার দিলাম, ফাইল/ফোল্ডারগুলোর উদ্দেশ্য-সহ। এটি `pages` রাউটার ভিত্তিক Next.js scaffold (তুমি চাইলে পরে `app/` রাউটারে কনভার্ট করতেও পারো)। আমি practical ডিরেক্টরি বিভাজন ও responsibilities অনুসারে সাজিয়েছি, যাতে বড় হয়ে গেলে maintain করা সহজ থাকে।

```
frontend/
├─ .env.example
├─ .eslintrc.js
├─ .prettierrc
├─ package.json
├─ next.config.js
├─ tailwind.config.js
├─ postcss.config.js
├─ public/
│  ├─ favicon.ico
│  ├─ manifest.json
│  └─ static/                # favicon, svgs, robots.txt, default images
├─ src/
│  ├─ pages/
│  │  ├─ api/
│  │  │  ├─ revalidate.js     # secure ISR revalidate endpoint
│  │  │  └─ preview.js        # preview mode endpoint for editors
│  │  ├─ _app.js              # global providers (theme, auth, swr)
│  │  ├─ _document.js         # custom Document (fonts, lang, meta baseline)
│  │  ├─ index.js             # homepage / posts list
│  │  ├─ [slug].js            # content detail page (getStaticPaths/Props)
│  │  └─ 404.js
│  ├─ components/
│  │  ├─ layout/
│  │  │  ├─ Header.js
│  │  │  ├─ Footer.js
│  │  │  └─ Container.js
│  │  ├─ content/
│  │  │  ├─ ContentRenderer.js # render editor JSON -> React nodes / HTML
│  │  │  ├─ RichText.js
```

```
│   │   │   └── ContentCard.js
│   │   ├── ui/
│   │   │   ├── Button.js
│   │   │   ├── Icon.js
│   │   │   └── Loader.js
│   │   └── seo/
│   │       └── SEO.js
│   ├── libs/
│   │   ├── api.js              # axios/fetch wrapper (baseURL, auth)
│   │   ├── cms.js              # convenience functions: fetchPost, fetchList, previewLink
│   │   └── formats.js          # date/slug helpers
│   ├── hooks/
│   │   ├── usePreview.js
│   │   └── useSWRWithAuth.js
│   ├── context/
│   │   └── AuthContext.js
│   ├── styles/
│   │   ├── globals.css
│   │   └── tailwind.css
│   ├── utils/
│   │   ├── renderTiptap.js     # server/client renderer helpers for Tiptap/Editor JSON
│   │   ├── image.js            # helpers for Cloudinary/Next/Image src generation
│   │   └── seo.js
│   ├── config/
│   │   └── index.js            # client-side config (api base, env flags)
│   └── types/                  # (optional) TypeScript types if using TS
│       └── content.d.ts
├── public/                     # (duplicate top-level public for clarity)
├── tests/
│   ├── __mocks__/
│   └── integration/
│       └── pages.home.test.js
└── README.md
```

প্রতিটি অংশের ব্যাখ্যা (কী রাখবে, কেন)

`pages/`

- `index.js` — হোম পেজ / পোস্ট লিস্ট; SSG/ISR (`getStaticProps`) দিয়ে সার্ভ করুন।

- `[slug].js` — কন্টেন্ট পেজ; `getStaticPaths` + `getStaticProps` ব্যবহার করে ISR। `preview` context দেখতে সক্ষম হবে যাতে ড্রাফট দেখানো যায়।
- `api/revalidate.js` — অতি-গুরুত্বপূর্ণ: CMS থেকে Pubish webhook এই এন্ডপয়েন্টে POST করবে (সিক্রেট চেক করে) → `res.revalidate('/slug')` কল করে Next.js page revalidate করে।
- `api/preview.js` — preview mode সেট করে (preview token দিয়ে) ও editor-কে draft link দেয়।
- `_app.js` — global providers (SWRConfig, ThemeProvider, AuthProvider)।
- `_document.js` — font preloads, lang attribute, meta baseline.

components/

- `layout/` — header, footer, global layout.
- `content/ContentRenderer.js` — সবচেয়ে গুরুত্বপূর্ণ: CMS-এ আপনার rich editor (Tiptap/Editor.js) JSON ফলাফল সংরক্ষণ করলে frontend-এ সেটা রেন্ডার করতে হবে। Renderer দুইটা উপায়ে করা যায়:
    - Backend-এ editor JSON → HTML রেন্ডার করে পাঠানো (সরাসরি `dangerouslySetInnerHTML`) — সহজ ও SEO-friendly।
    - অথবা frontend-এ JSON → React nodes রূপান্তর করা (Tiptap renderer / custom renderer)। `ContentRenderer` এই লজিক রাখবে।
- `seo/SEO.js` — ফেইচুয়াল OG/meta ট্যাগ সেট করা (title, description, image, canonical, hreflang)।

libs/

- কেন্দ্রীয় API লেয়ারের ফাংশন: base URL, token injection, revalidate call helper, preview link generator।

hooks/

- `usePreview` — preview mode handling (toggle preview state + fetch draft).
- `useSWRWithAuth` — SWR + auth token wrapper।

utils/renderTiptap.js

- যদি তুমি Tiptap JSON রাখো, server-side rendering দরকার হলে এখানে helper থাকবে: `tiptapToHtml(json)` (যদি back-end এ render না করে)। বিকল্প: back-end থেকে HTML পাঠালে ফ্রন্টএন্ডে এই অংশ অনেক সরল হবে।

`image.js`

- Cloudinary/S3 URL builder, responsive sizes, placeholder generation, integration with `next/image`.

`config/index.js`

- ক্লায়েন্ট সাইড কনফিগ: `NEXT_PUBLIC_API_URL`, CDN base, analytics key, features flags।

গুরুত্বপূর্ণ implementation notes / recommendations

1. **ISR + Revalidate**
    - `getStaticProps(...){ revalidate: 60 }` ব্যবহার করুন। CMS publish → webhook → frontend `/api/revalidate` → `res.revalidate('/the-slug')` → নতুন কন্টেন্ট live।
    - Always secure revalidate with `REVALIDATE_SECRET` header/token.
2. **Preview mode**
    - CMS should generate preview URL: `${FRONTEND_URL}/api/preview?token=${PREVIEW_TOKEN}&slug=${slug}`
    - `pages/[slug].js` should detect `context.preview` and fetch draft (`/api/contents/:slug?draft=true`) with preview-secret header.
3. **Rendering rich content**
    - Option A (recommended): **Render HTML on backend** (e.g., Tiptap server-side `renderToHTML`) and return `bodyHtml` for frontend. Safer for SSR & SEO.
    - Option B: Render JSON on frontend with a dedicated renderer—works fine but ensure server-side rendering compatibility.

4. **Image optimization**
    - Use Next.js `<Image/>` with remotePatterns configured in `next.config.js` for Cloudinary/S3 host. Or use Cloudinary loader. Provide utility functions to generate `srcset`/sizes.
5. **Design system**
    - Keep small UI primitives in `components/ui/` so admin UI (when you add later) and public site reuse components.
6. **Auth**
    - For editor/admin preview flows, use NextAuth (if you host admin in frontend) or a separate admin app. For now, preview token + CMS auth is enough.
7. **Analytics & SEO**
    - Add Google Analytics / Plausible wrapper in `_app.js` and `SEO.js`. Generate `sitemap.xml` server-side (or at build time) and `robots.txt`.
8. **Testing**
    - Use Jest + React Testing Library for components and Playwright for E2E preview flows (publish → revalidate → page updated).

Recommended npm packages

next react react-dom
swr axios
tailwindcss postcss autoprefixer
next-compose-plugins (if using loaders)
next-images (if needed)
date-fns
jsdom (for server-side rendering helpers)
@tiptap/react (if rendering editor on client)
dompurify (for sanitizing HTML when using dangerouslySetInnerHTML)
next-seo (optional)

(If you use TypeScript, add `typescript`, `@types/*` packages.)

Example env vars (`.env.example`)

```
NEXT_PUBLIC_API_URL=https://your-backend.example.com
NEXT_PUBLIC_CLOUDINARY_BASE=https://res.cloudinary.com/your-cloud-name
REVALIDATE_SECRET=supersecret_revalidate_token
PREVIEW_TOKEN=preview_secret_token
NEXT_PUBLIC_ANALYTICS_ID=G-XXXXXXX
```

Scripts (package.json)

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start",
  "lint": "next lint",
  "format": "prettier --write .",
  "test": "jest"
}
```

Quick checklist before you start coding frontend

- Decide whether editor content will be stored as **HTML** or **Editor JSON**. If JSON: implement server-side renderer or a robust client renderer.
- Configure `next.config.js` remotePatterns or image loader for Cloudinary/S3.
- Implement `api/revalidate.js` and test webhook flow locally (use `ngrok` to receive backend webhook to local Next.js dev).
- Add preview route and verify preview fetches draft content (use header secret).
- Create `ContentRenderer` early — it's the core piece that turns CMS data into page HTML.