Lacspace Packages

Developer Handbook

How to use every Lacspace package — a which-package-for-what guide, examples, tips and warnings for each.

61 packages · 13 kits · generated 2026-10-05

AT A GLANCE

- Every package is zero-dependency (or React-peer only), isomorphic and fully typed.
- Start instantly with `npm create lacspace-app@latest` — a finished Next.js app.
- This handbook has a getting-started guide, a which-package-for-what map, and a documented page for every package.
- Tap any entry in the Contents to jump straight to it. The always-current version lives at lacspace.com/docs.

LACSPACE
DEVELOPER DOCS

Contents

Getting started 5
Conventions & guarantees 5
Which package for what 6

Core SDK 10
@lacspace/sdk 10
@lacspace/api 10
@lacspace/auth 11
@lacspace/analytics 11

StockKit — for stock-market apps 12
@lacspace/indicators 12
@lacspace/market 12
@lacspace/market-clock 13
@lacspace/paper-trade 13

MailKit — email for backends 15
@lacspace/mailer 15
@lacspace/email-templates 16
@lacspace/email-validate 16
@lacspace/email-verify 16

SEO Kit — everything crawlers & AI read 18
@lacspace/seo 18
@lacspace/sitemap 19
@lacspace/robots 20
@lacspace/llms-txt 20
@lacspace/site-verify 21
@lacspace/rss 21
@lacspace/slugify 22

Security Kit — the seven layers, as code 23
@lacspace/crypto 23
@lacspace/password 24
@lacspace/jwt 25
@lacspace/apikey 25
@lacspace/otp 26
@lacspace/webauthn 27
@lacspace/mfa 27
@lacspace/lock 28

LACSPACE DEVELOPER DOCS

@lacspace/headers 28
@lacspace/redact 29

Backend Kit — server-side essentials 30
@lacspace/signed-url 30
@lacspace/pdf 30
@lacspace/webhooks 31
@lacspace/idempotency 32

Data Kit — export & parse 33
@lacspace/xlsx 33
@lacspace/csv 33

DX Kit — everyday utilities 35
@lacspace/humanize 35
@lacspace/color 35
@lacspace/money 36

Utils Kit — the missing standard library 38
@lacspace/id 38
@lacspace/retry 39
@lacspace/case 39
@lacspace/cache 40

App Kit — ship a polished app 42
@lacspace/validate 42
@lacspace/form 43
@lacspace/og 44
@lacspace/ui 45
@lacspace/markdown 46
@lacspace/analytics-lite 47

WebKit — for web apps & backends 49
@lacspace/flags 49
@lacspace/env 49
@lacspace/rate-limit 50
@lacspace/next 51

Nepal toolkit 52
@lacspace/nepali-date 52
@lacspace/nepali-utils 52

React Kit — hooks, state, data & UX 54

LACSPACE DEVELOPER DOCS

@lacspace/hooks 54
@lacspace/store 55
@lacspace/query 56
@lacspace/theme 57
@lacspace/hotkeys 58
@lacspace/virtual 60
@lacspace/react 61

LACSPACE
DEVELOPER DOCS

Getting started

Every Lacspace package is **zero-dependency** (or React-peer only), **isomorphic** (servers, browsers, the edge, React Native), ships **dual ESM + CJS** with full TypeScript types, and is free under the Lacspace Free Licence.

Install a package

```
npm i @lacspace/seo
# pnpm add @lacspace/seo · yarn add @lacspace/seo · bun add @lacspace/seo
```

Or scaffold a whole app (fastest)

create-lacspace-app generates a finished-looking Next.js app with SEO, security headers, dynamic OG images, a working contact form, a ?K command palette and a CI SEO gate already wired in — plus blog and docs templates.

```
npm create lacspace-app@latest my-app
cd my-app && npm run dev
```

> **TIP**
> Every generated app ships with a `.github/workflows/seo.yml` that audits your pages on each push and fails the build below grade A.

Conventions & guarantees

The whole ecosystem follows the same rules, so once you learn one package the rest feel familiar.

- **Zero-dependency core** — built on the platform (`fetch`, Web Crypto). Your `node_modules` and supply chain stay small.
- **Isomorphic** — the same code runs on Node, browsers, the edge and React Native, unless a package is explicitly Node-only (e.g. `@lacspace/mailer`).
- **Dual ESM + CJS** with a proper `exports` map and full `.d.ts` types.
- **Tree-shakeable** — import only what you use.
- **Free licence** — the Lacspace Free Licence is permissive — free to use, modify and sell — for the public packages.

> **NOTE**
> Node-only packages (raw SMTP, filesystem) say so in their Runtime line — don't import those on the edge runtime.

Which package for what

Start from what you're building; each goal maps to the packages that solve it, with a snippet to copy.

Validate a form or API body

Type-safe schemas with coercion, then handle the submission with spam protection.

Reach for: @lacspace/validate, @lacspace/form

```
import { v } from "@lacspace/validate";
import { createForm } from "@lacspace/form";

const contact = createForm({
  schema: v.object({ email: v.string().email(), message: v.string().min(10) }),
  honeypot: "company",
});
```

Rank on Google (SEO + JSON-LD)

Configure your site once; metadata, canonical, Open Graph, Twitter and JSON-LD flow everywhere. Gate it in CI.

Reach for: @lacspace/seo, @lacspace/sitemap, @lacspace/robots

```
import { defineSite } from "@lacspace/seo";
export const site = defineSite({ name: "Acme", url: "https://acme.com" });
// site.meta({ title, path }) -> full Next.js Metadata
// npx @lacspace/seo crawl https://acme.com --min-grade A
```

Beautiful social share images

A dynamic OG image per page — real PNG via next/og, or a zero-dep SVG anywhere.

Reach for: @lacspace/og

```
import { ImageResponse } from "next/og";
import { ogCard } from "@lacspace/og";
export const runtime = "edge";
export const GET = (req) => new ImageResponse(
  ogCard({ title: "Hello", from: "#22d3ee", to: "#6366f1" }), { width: 1200, height: 630
  });
```

Make the UI feel alive

Scroll reveals, count-ups, gradient text, tilt cards and a ?K command palette — no animation library.

Reach for: @lacspace/ui

```
import { Reveal, Counter, CommandPalette } from "@lacspace/ui";
<Reveal><Counter value={12480} suffix="+" /></Reveal>
```

Cache slow calls / add SWR

Bound memory (LRU), expire (TTL) and hide latency (stale-while-revalidate); concurrent callers share one fetch.

Reach for: @lacspace/cache

```
import { createCache } from "@lacspace/cache";
const cache = createCache({ max: 500, ttl: 60_000 });
await cache.wrap(`user:${id}`, () => db.users.find(id), { staleWhileRevalidate: 30_000
});
```

Render Markdown (blog / docs)

Markdown to safe HTML with headings, tables, code and a table-of-contents extractor.

Reach for: @lacspace/markdown

```
import { markdownToHtml, extractHeadings } from "@lacspace/markdown";
const html = markdownToHtml(post);
const toc = extractHeadings(post);
```

Handle money correctly

Integer minor-units (no float bugs), currency-safe math, split without losing a cent, Intl formatting.

Reach for: @lacspace/money

```
import { money } from "@lacspace/money";
money(19.99, "USD").multiply(3).format(); // "$59.97"
money(10, "USD").allocate([1, 1, 1]);     // 3.34, 3.33, 3.33
```

Privacy-first analytics

Cookieless page views and events to your own endpoint — no consent banner, respects Do-Not-Track.

Reach for: @lacspace/analytics-lite

```
import { createAnalytics } from "@lacspace/analytics-lite";
const a = createAnalytics({ endpoint: "/api/collect", siteId: "acme" });
a.autoTrack();
```

Authentication & sessions

OTP/TOTP 2FA, password hashing, JWTs, API keys, WebAuthn and account lockout — all Web-Crypto based.

Reach for: @lacspace/password, @lacspace/jwt, @lacspace/otp, @lacspace/apikey

```
import { hashPassword, verifyPassword } from "@lacspace/password";
const hash = await hashPassword(input);
```

Resilient network calls

Retry with backoff + jitter, per-call timeouts, and a circuit breaker; cache the result.

Reach for: @lacspace/retry, @lacspace/cache

```
import { retry } from "@lacspace/retry";
await retry(() => fetch(url), { retries: 4 });
```

Send transactional email

Zero-dependency SMTP, bulletproof responsive HTML templates, and address validation/verification.

Reach for: @lacspace/mailer, @lacspace/email-templates, @lacspace/email-validate

```
import { mailerFromEnv } from "@lacspace/mailer";
await mailerFromEnv().send({ to, subject, html });
```

Rate-limit & secure an API

Token-bucket / sliding-window limits, hardened security headers, idempotency keys and redaction.

Reach for: @lacspace/rate-limit, @lacspace/headers, @lacspace/idempotency

```
import { rateLimit } from "@lacspace/rate-limit";
const { allowed } = rateLimit(ip, { limit: 100, window: 60_000 });
```

Export data (Excel / CSV / PDF)

Real .xlsx and RFC-4180 CSV with no heavy libraries, plus watermarked PDFs with no headless browser.

Reach for: @lacspace/xlsx, @lacspace/csv, @lacspace/pdf

```
import { jsonToXlsx } from "@lacspace/xlsx";
const bytes = jsonToXlsx(rows); // Uint8Array (.xlsx)
```

Common React needs (storage, debounce, clicks)

The everyday hooks — persisted state, debounced values, click-outside, media queries, clipboard — SSR-safe and typed.

Reach for: @lacspace/hooks

```
import { useLocalStorage, useDebounce, useMediaQuery } from "@lacspace/hooks";

const [q, setQ] = useLocalStorage("q", "");
const debounced = useDebounce(q, 300);
const isMobile = useMediaQuery("(max-width: 640px)");
```

Global state without the boilerplate

Create a store, use selectors, no provider — with a persist middleware for localStorage.

Reach for: @lacspace/store

```
import { create, persist } from "@lacspace/store";

const useCart = create(persist((set) => ({
  items: [] as string[],
  add: (id: string) => set((s) => ({ items: [...s.items, id] })),
}), { name: "cart" }));
```

Fetch, cache & revalidate server data

useQuery shares a cache across components, de-dupes requests and revalidates on focus; useMutation writes back.

Reach for: @lacspace/query

```
import { useQuery, mutate } from "@lacspace/query";

const { data, isLoading } = useQuery(["user", id], () => fetchUser(id));
// after an update:
mutate(["user", id]);
```

Dark mode & a ?K command palette

SSR-safe theming with no flash, plus keyboard shortcuts with combos and sequences.

Reach for: @lacspace/theme, @lacspace/hotkeys, @lacspace/ui

```
import { useTheme } from "@lacspace/theme";
import { useHotkeys } from "@lacspace/hotkeys";

const { setTheme } = useTheme();
useHotkeys("mod+k", () => setOpen(true));
```

Render huge lists smoothly

Only render the rows in view — fixed or dynamically-measured heights, overscan and scroll-to-index.

Reach for: @lacspace/virtual

```
import { useVirtualizer } from "@lacspace/virtual";

const v = useVirtualizer({ count: 100000, getScrollElement: () => ref.current,
estimateSize: () => 48 });
v.getVirtualItems().map((row) => /* absolutely-positioned row at row.start */);
```

Start a whole app

Scaffold a finished-looking Next.js app with every relevant package pre-wired.

Reach for: create-lacspace-app

```
npm create lacspace-app@latest my-app
```

Core SDK

Talk to the Lacspace platform from any JavaScript runtime.

In this kit

- **@lacspace/sdk** — The all-in-one client
- **@lacspace/api** — Zero-dependency HTTP client
- **@lacspace/auth** — Authentication flows
- **@lacspace/analytics** — Event tracking

@lacspace/sdk

Bundles api + auth + analytics behind one client that shares a single connection — a login token instantly applies everywhere — plus e-commerce helpers. Start here.

Install: npm i @lacspace/sdk

Runtime: api · auth · analytics

Example

```
lac.ecommerce.getProducts()
// -> Product[]
```

Links: npm https://www.npmjs.com/package/@lacspace/sdk · source https://github.com/lacspace/npm-packages/tree/main/sdk

@lacspace/api

The foundation everything is built on. Typed requests, predictable errors, automatic retries with backoff, request/response interceptors, query params, in-flight de-duping, response caching and cursor pagination — built on the platform fetch. A tiny axios / ky alternative that runs on Node, browsers, edge and React Native.

Install: npm i @lacspace/api

Runtime: zero dependencies

Example

```
api.get("users/me")
// -> { id, name, … }
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/api · source https://github.com/lacspace/npm-packages/tree/main/api

@lacspace/auth

Login, register, current user, logout and token refresh — with the bearer token applied automatically after sign-in. Plus auto-refresh (401 -> refresh -> retry, deduped), pluggable token storage (memory / localStorage) and auth-change subscriptions.

Install: npm i @lacspace/auth

Runtime: built on @lacspace/api

Example

```
auth.login({ email, password })
// -> { token, user }
```

Links: npm https://www.npmjs.com/package/@lacspace/auth · source https://github.com/lacspace/npm-packages/tree/main/auth

@lacspace/analytics

Track events one at a time, or queue and flush them in a single batch — handy for offline-first apps and reducing request volume.

Install: npm i @lacspace/analytics

Runtime: built on @lacspace/api

Example

```
analytics.track("viewed")
```

Links: npm https://www.npmjs.com/package/@lacspace/analytics · source https://github.com/lacspace/npm-packages/tree/main/analytics

StockKit — for stock-market apps

The suite that powers StockYatra. Everything a trading, charting or algo app re-implements — done once, done right.

In this kit

- **@lacspace/indicators** — Streaming technical indicators
- **@lacspace/market** — Money & mechanics
- **@lacspace/market-clock** — Trading hours & holidays
- **@lacspace/paper-trade** — Paper-trading engine

@lacspace/indicators

RSI, MACD, EMA, Bollinger, ATR, VWAP, Stochastic, Supertrend, ADX — with an incremental O(1) API. Push one live LTP tick and the indicator updates without recomputing the whole series. Plus a tick->OHLC candle aggregator and candlestick pattern detection (doji, hammer, engulfing, harami…).

Install: npm i @lacspace/indicators

Runtime: zero dependencies

Example

```
rsi.next(ltp)
// -> 70.46
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/indicators · source https://github.com/lacspace/npm-packages/tree/main/indicators

@lacspace/market

P&L, returns, CAGR, XIRR, position sizing, tick-size rounding and circuit limits — plus a real Indian brokerage & charges calculator (STT, GST, SEBI, stamp) with discount-broker presets. Now with Black-Scholes options pricing & greeks, implied volatility, and portfolio analytics (Sharpe, Sortino, max drawdown).

Install: npm i @lacspace/market

Runtime: zero dependencies

Example

```
charges({ segment: "intraday", buy: 100, sell: 102, qty: 500 }).netPnl
// -> 946.34
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/market · source https://github.com/lacspace/npm-packages/tree/main/market

@lacspace/market-clock

Is the market open right now? When does it next open or close? A holiday-aware, timezone-correct trading clock with NSE/BSE presets — or bring your own exchange spec.

Install: npm i @lacspace/market-clock

Runtime: zero dependencies

Example

```
new MarketClock(NSE).status()
// -> "open"
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/market-clock · source https://github.com/lacspace/npm-packages/tree/main/market-clock

@lacspace/paper-trade

The simulator core behind StockYatra. A virtual wallet, market/limit/stop orders that fill against live prices, positions, holdings and live mark-to-market P&L. Now with a per-fill charges hook (pair it with @lacspace/market for net P&L) and backtest-style stats — win rate, profit factor, avg win/loss. Drop it into any app.

Install: npm i @lacspace/paper-trade

Runtime: zero dependencies

Example

```
acct.buy("RELIANCE", { qty: 10 })
// -> Order { status: "FILLED" }
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/paper-trade · source https://github.com/lacspace/npm-packages/tree/main/paper-trade

LACSPACE DEVELOPER DOCS

MailKit — email for backends

Everything a backend needs to send, compose and validate email. Send with one line; compose bulletproof HTML; catch bad addresses before they cost you.

In this kit

- **@lacspace/mailer** — Zero-dependency SMTP client
- **@lacspace/email-templates** — Responsive HTML emails
- **@lacspace/email-validate** — Validation beyond regex
- **@lacspace/email-verify** — Deliverability checks

@lacspace/mailer

Send email over raw Node net/tls — STARTTLS, AUTH, MIME with attachments — with no npm dependencies. Provider presets make Hostinger, Gmail, Outlook, Zoho & more a one-liner. Node only.

> **NOTE**
> When to use — Sending transactional email over raw SMTP with zero npm dependencies (Node only).

Install: npm i @lacspace/mailer

Runtime: zero dependencies · Node

Example

```
createMailer(presets.hostinger({ user, pass })).send({ to, subject, html })
// -> { messageId, accepted }
```

> **TIP**
> `mailerFromEnv()` reads `SMTP_*` vars; presets exist for Hostinger, Gmail, Outlook, Zoho and more.

> **TIP**
> Compose bodies with @lacspace/email-templates for bulletproof, dark-mode-safe HTML.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> This is Node-only (uses `node:net`/`node:tls`), so it won't run on the edge runtime.

Links: npm https://www.npmjs.com/package/@lacspace/mailer · source https://github.com/lacspace/npm-packages/tree/main/mailer

@lacspace/email-templates

Compose bulletproof, dark-mode-aware HTML emails from simple blocks (button, OTP code, invoice table) — no more <table> soup. Ships OTP/welcome/alert/invoice templates. A tiny MJML alternative.

Install: npm i @lacspace/email-templates

Runtime: zero dependencies

Example

```
otpEmail({ code: "482913", brandName: "Lacspace" })
// -> <!DOCTYPE html>…
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/email-templates · source https://github.com/lacspace/npm-packages/tree/main/email-templates

@lacspace/email-validate

Syntax + disposable/temp-mail detection, role & free-provider flags, Gmail normalization, and 'did you mean?' typo suggestions (gmial.com -> gmail.com). Store the normalized form to de-dupe users.

Install: npm i @lacspace/email-validate

Runtime: zero dependencies

Example

```
validateEmail("me@gmial.com").suggestion
// -> "me@gmail.com"
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/email-validate · source https://github.com/lacspace/npm-packages/tree/main/email-validate

@lacspace/email-verify

Resolve a domain's MX records and optionally run a real SMTP RCPT probe to check a mailbox exists — without sending anything. Best-effort (servers greylist/catch-all), great for catching dead domains. Node

only.

Install: npm i @lacspace/email-verify

Runtime: MX + SMTP · Node

Example

```
await verifyEmail(email, { checkSmtp: false })
// -> { mxFound: true }
```

Links: npm https://www.npmjs.com/package/@lacspace/email-verify · source https://github.com/lacspace/npm-packages/tree/main/email-verify

LACSPACE DEVELOPER DOCS

SEO Kit — everything crawlers & AI read

Generate every file and tag search engines and AI crawlers look for — metadata, JSON-LD, sitemaps, robots.txt, llms.txt, feeds and more. All typed and dependency-free.

In this kit

- **@lacspace/seo** — SEO Autopilot & CI gate
- **@lacspace/sitemap** — sitemap.xml + index
- **@lacspace/robots** — robots.txt + AI blocking
- **@lacspace/llms-txt** — llms.txt / llms-full.txt
- **@lacspace/site-verify** — Search-engine verification
- **@lacspace/rss** — RSS / Atom / JSON feeds
- **@lacspace/slugify** — SEO URL slugs

@lacspace/seo

SEO Autopilot: defineSite() once, then site.meta()/article()/product()/faq() auto-fill Metadata AND JSON-LD — title template, canonical, OpenGraph, Twitter, auto description and dynamic OG image. Plus 19 schema.org builders, a @graph composer, content helpers (excerpt, reading time), and a built-in auditor with a CI sitemap crawler — `npx @lacspace/seo crawl <site> --min-grade A` fails your build the moment SEO regresses. Stop writing SEO by hand.

> **NOTE**
> When to use — Any Next.js (or framework-agnostic) site that needs correct metadata, Open Graph, Twitter cards and schema.org JSON-LD without hand-writing fragile objects on every page.

Install: npm i @lacspace/seo

Runtime: zero dependencies

Example

```
site.meta({ title: "Pricing", path: "/pricing" })
// -> full Metadata + OG + Twitter
```

Key API

defineSite(config) The Autopilot engine — returns a `site` with meta()/article()/product()/faq()/…

seoMetadata(input) Framework-agnostic -> Next.js `Metadata`.

auditHtml(html) Grade a page's on-page SEO 0–100 with per-check results.

CLI: audit / crawl `npx @lacspace/seo audit <url>` · `crawl <site> --min-grade A`.

> **TIP**
> Call `defineSite()` once in `lib/site.ts`; then `site.meta({ title, path })` returns a full Next.js `Metadata` object with canonical, OG and Twitter already filled.

> **TIP**
> Use `site.article()`, `site.product()`, `site.faq()` etc. to get metadata AND a matching JSON-LD `@graph` in one call — great for rich results.

> **TIP**
> Wire the CI gate: `npx @lacspace/seo crawl https://yoursite.com --min-grade A` fails your build if any page's on-page SEO regresses.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> Set `NEXT_PUBLIC_SITE_URL` in production, or canonical/OG URLs fall back to your placeholder domain.

> **WARNING**
> Don't hardcode a canonical in the root layout — every child page would inherit it. Let each page set its own `path`.

Links: npm https://www.npmjs.com/package/@lacspace/seo · source https://github.com/lacspace/npm-packages/tree/main/seo

@lacspace/sitemap

URL entries with lastmod/changefreq/priority, image/video/news extensions and hreflang alternates. Auto-splits past 50,000 URLs into an index. Emits XML and Next.js sitemap.ts objects.

> **NOTE**
> When to use — Generating `sitemap.xml` from your routes without repeating your domain on every row.

Install: npm i @lacspace/sitemap

Runtime: zero dependencies

Example

```
sitemap([{ loc, priority: 1.0, changefreq: "daily" }])
// -> <?xml … urlset>
```

> **TIP**
> `sitemapForSite(site.config, paths)` derives absolute URLs from your `defineSite()` config.

> **TIP**
> For a blog/docs, map your content list into the paths array so every page is indexed.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/sitemap · source https://github.com/lacspace/npm-packages/tree/main/sitemap

@lacspace/robots

Typed per-user-agent rules, sitemap refs, a parser, framework presets (Next.js, WordPress, Shopify) and an isAllowed() crawlability matcher — plus a one-liner to block AI crawlers (GPTBot, ClaudeBot, CCBot, Google-Extended and 14 more).

> **NOTE**
> When to use — Generating `robots.txt` from config, including AI-crawler controls.

Install: npm i @lacspace/robots

Runtime: zero dependencies

Example

```
blockAiBots({ sitemap: "/sitemap.xml" })
// -> robots.txt
```

> **TIP**
> `robotsForSite(site.config)` keeps robots in sync with your canonical host and sitemap URL.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/robots · source https://github.com/lacspace/npm-packages/tree/main/robots

@lacspace/llms-txt

Generate and parse llms.txt and llms-full.txt — the emerging llmstxt.org standard that gives LLMs a curated, Markdown map of your site — or build one straight from your sitemap entries. The SEO layer for the AI era.

Install: npm i @lacspace/llms-txt

Runtime: zero dependencies

Example

```
llmsTxt({ title, summary, sections })
// -> # Site\n> summary…
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/llms-txt · source https://github.com/lacspace/npm-packages/tree/main/llms-txt

@lacspace/site-verify

Verification meta tags, Next.js verification metadata and file tokens for Google Search Console, Bing Webmaster, Yandex, Baidu, Pinterest, Ahrefs, Facebook and more — no more hunting for the right meta name.

Install: npm i @lacspace/site-verify

Runtime: zero dependencies

Example

```
verificationMeta({ google, bing, yandex })
// -> meta[]
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/site-verify · source https://github.com/lacspace/npm-packages/tree/main/site-verify

@lacspace/rss

Define your feed and items once, emit valid RSS 2.0, Atom 1.0 or JSON Feed 1.1 — with CDATA content, categories, authors and podcast enclosures. Great for blogs, changelogs and news.

Install: npm i @lacspace/rss

Runtime: zero dependencies

Example

```
rss(feed, items)
// -> <?xml … rss>
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/rss · source https://github.com/lacspace/npm-packages/tree/main/rss

@lacspace/slugify

Turn any text into a clean, SEO-friendly URL slug — transliterates diacritics, collapses separators, truncates on word boundaries, and guarantees uniqueness against an existing set.

> **NOTE**
> When to use — Turning titles into URL-safe slugs consistently across your app.

Install: npm i @lacspace/slugify

Runtime: zero dependencies

Example

```
slugify("Héllo, World! — 2026")
// -> "hello-world-2026"
```

> **TIP**
> Use the same slugify on write and on lookup so URLs always match.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/slugify · source https://github.com/lacspace/npm-packages/tree/main/slugify

Security Kit — the seven layers, as code

Real security primitives built on vetted crypto (Web Crypto — never hand-rolled): AES, password hashing, tokens, API keys, 2FA/3FA, passkeys, account lockout, hardened headers and log redaction. For web, backend, mobile, AWS/S3 and Mongo.

In this kit

- **@lacspace/crypto** — AES-256-GCM encryption
- **@lacspace/password** — Password hashing
- **@lacspace/jwt** — JWTs & tokens
- **@lacspace/apikey** — API keys
- **@lacspace/otp** — TOTP / HOTP 2FA
- **@lacspace/webauthn** — Passkeys / biometric
- **@lacspace/mfa** — 2FA / 3FA orchestration
- **@lacspace/lock** — Account lockout
- **@lacspace/headers** — Secure headers + CSP
- **@lacspace/redact** — Log redaction

@lacspace/crypto

Authenticated AES-256-GCM encrypt/decrypt (with AAD), PBKDF2 & HKDF key derivation, SHA-2, HMAC, secure random and constant-time compare — a thin, correct layer over Web Crypto. Plus a versioned Keyring for envelope encryption & key rotation. Encrypt Mongo fields, S3 payloads and cookies.

> **NOTE**
> When to use — Encrypting a field before it lands in your database or object storage, using authenticated AES-256-GCM.

Install: npm i @lacspace/crypto

Runtime: zero dependencies · isomorphic

Example

```
await decrypt(await encrypt(data, key), key)
// -> data
```

> **TIP**
> Built on Web Crypto — the same code works in Node, browsers and the edge.

> **TIP**
> Store the key outside your code (env/secret manager); rotate by versioning ciphertext.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> Never hand-roll crypto or reuse a nonce — the package manages nonces per message; don't override that.

> **WARNING**
> Losing the key means losing the data — back it up securely.

Links: npm https://www.npmjs.com/package/@lacspace/crypto · source https://github.com/lacspace/npm-packages/tree/main/crypto

@lacspace/password

Hash & verify with PBKDF2-HMAC-SHA256 (600k iterations per OWASP) in a portable PHC string, constant-time verify, rehash detection and a strength estimator. Never store plaintext.

> **NOTE**
> When to use — Hashing and verifying user passwords with PBKDF2 (Web Crypto), no native modules.

Install: npm i @lacspace/password

Runtime: zero dependencies · isomorphic

Example

```
await verify(input, storedHash)
// -> true / false
```

> **TIP**
> `hashPassword()` embeds the salt and parameters in the output string, so `verifyPassword()` needs nothing else.

> **TIP**
> Pair with @lacspace/lock to throttle brute-force attempts.

TIP
Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

WARNING
Verification is intentionally slow (that's the point). Do it off the hot path and rate-limit login endpoints.

Links: npm https://www.npmjs.com/package/@lacspace/password · source https://github.com/lacspace/npm-packages/tree/main/password

@lacspace/jwt

Sign & verify JWTs — symmetric HS256/384/512 and asymmetric RS256/ES256 — with strict expiry/issuer/audience checks over Web Crypto, so it runs on edge and workers. Plus remote JWKS verification, refresh-token rotation with reuse detection, and opaque & CSRF tokens. A tiny jose alternative.

NOTE
When to use — Issuing and verifying JSON Web Tokens (HS/RS/ES) for sessions or service-to-service auth.

Install: npm i @lacspace/jwt

Runtime: zero dependencies · isomorphic

Example

```
await verify(token, secret, { issuer })
// -> payload
```

TIP
Keep access tokens short-lived and rotate refresh tokens; verify `exp`/`nbf` (the package does this).

TIP
For RS/ES, publish a JWKS and verify against it.

TIP
Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

WARNING
Never accept the `alg: none` algorithm, and don't verify HS tokens with a public key — the verifier guards against algorithm confusion.

Links: npm https://www.npmjs.com/package/@lacspace/jwt · source https://github.com/lacspace/npm-packages/tree/main/jwt

@lacspace/apikey

Issue prefixed, high-entropy keys (lac_live_…), store only the SHA-256 hash, show the last 4, and verify in constant time — exactly how Stripe/GitHub-style keys work.

Install: npm i @lacspace/apikey

Runtime: zero dependencies · isomorphic

Example

```
await verifyApiKey(key, storedHash)
// -> true / false
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/apikey · source https://github.com/lacspace/npm-packages/tree/main/apikey

@lacspace/otp

Google Authenticator-compatible two-factor auth over Web Crypto — runs on Node, edge and browser. Generate secrets, verify codes with drift tolerance and replay protection, build otpauth:// URIs, and issue single-use hashed backup/recovery codes. A tiny speakeasy alternative, verified against the RFC test vectors.

> **NOTE**
> When to use — Adding TOTP/HOTP two-factor authentication (Google Authenticator-style) to an app.

Install: npm i @lacspace/otp

Runtime: zero dependencies · isomorphic

Example

```
await verifyTotp(code, secret)
// -> 0 (valid) | null
```

> **TIP**
> `keyuri()` produces the `otpauth://` URL for a QR code; show it once during enrollment.

> **TIP**
> `verifyTotp()` accepts a small time window to tolerate clock drift.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> Store the shared secret encrypted (see @lacspace/crypto) and offer backup codes for recovery.

Links: npm https://www.npmjs.com/package/@lacspace/otp · source https://github.com/lacspace/npm-packages/tree/main/otp

@lacspace/webauthn

FaceID, fingerprint and security keys via WebAuthn — browser ceremony helpers plus server challenge, options and real ES256/RS256 assertion verification (CBOR/COSE parse, DER handling) over Web Crypto.

Install: npm i @lacspace/webauthn

Runtime: zero dependencies · isomorphic

Example

```
await verifyAuthentication({ … })
// -> { verified, newCounter }
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/webauthn · source https://github.com/lacspace/npm-packages/tree/main/webauthn

@lacspace/mfa

Combine password + TOTP + passkey into 2FA/3FA step-up flows with NIST assurance levels (AAL1–3). Track which factors are cleared and decide when a policy is satisfied.

Install: npm i @lacspace/mfa

Runtime: zero dependencies · isomorphic

Example

```
mfaSession(cfg).markVerified("totp").satisfied
// -> true
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/mfa · source https://github.com/lacspace/npm-packages/tree/main/mfa

@lacspace/lock

Server-side brute-force protection: track failed attempts per user/IP, lock after N strikes with exponential backoff, auto-expire the window, reset on success. Pluggable store (Redis-ready).

Install: npm i @lacspace/lock

Runtime: zero dependencies · isomorphic

Example

```
(await guard.check(ip)).locked
// -> false
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/lock · source https://github.com/lacspace/npm-packages/tree/main/lock

@lacspace/headers

A tiny framework-agnostic Helmet: strict security response headers (HSTS, X-Frame-Options, Referrer-Policy, COOP) and a typed Content-Security-Policy builder. Works everywhere + Next.js.

> **NOTE**
> When to use — Applying hardened security headers (HSTS, CSP, X-Frame-Options, etc.) in a Next.js config or any server.

Install: npm i @lacspace/headers

Runtime: zero dependencies · isomorphic

Example

```
securityHeaders({ contentSecurityPolicy })
// -> { HSTS, CSP, … }
```

> **TIP**
> `toNextHeaders()` returns the exact shape `next.config` expects — drop it into `async headers()`.

> **TIP**
> Tighten the CSP allowlist whenever you add a third-party script or font.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> A too-strict CSP can silently break inline scripts/styles — test in report-only mode first.

Links: npm https://www.npmjs.com/package/@lacspace/headers · source https://github.com/lacspace/npm-packages/tree/main/headers

@lacspace/redact

Mask secrets & PII before they reach your logs — by sensitive key name (password, token, authorization…) and by pattern (JWTs, API keys, emails, cards, IPs). Deep, on strings and objects.

Install: npm i @lacspace/redact

Runtime: zero dependencies · isomorphic

Example

```
redact({ password: "x", token: "y" })
// -> { password: "[REDACTED]" }
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/redact · source https://github.com/lacspace/npm-packages/tree/main/redact

Backend Kit — server-side essentials

The things every backend re-implements — signed links and PDF documents — done once, zero-dependency, isomorphic. A new kit, growing.

In this kit

- **@lacspace/signed-url** — Signed & expiring URLs / tokens
- **@lacspace/pdf** — PDF invoices, receipts & docs
- **@lacspace/webhooks** — Sign, verify & deliver webhooks
- **@lacspace/idempotency** — Exactly-once operations

@lacspace/signed-url

HMAC-signed, expiring URLs and tokens over Web Crypto — magic-login links, secure download links, unsubscribe links and one-time actions. Tamper-proof, timing-safe, with magicLink() / signUrl() helpers. Fills a real gap: no self-hosted, zero-dep option existed.

> **NOTE**
> When to use — Creating expiring, tamper-proof links — magic login, password reset, time-limited downloads.

Install: npm i @lacspace/signed-url

Runtime: built on @lacspace/crypto

Example

```
await verifyUrl(request.url, { secret })
// -> { valid: true }
```

> **TIP**
> Keep expiries short and include a purpose/scope in the payload so a link can't be reused elsewhere.

> **WARNING**
> The signing secret must stay server-side — never ship it to the client.

Links: npm https://www.npmjs.com/package/@lacspace/signed-url · source https://github.com/lacspace/npm-packages/tree/main/signed-url

@lacspace/pdf

Generate real PDFs with zero dependencies and no headless browser — pdfkit is heavy and node-only, puppeteer spins up Chromium. Batteries-included invoice() & receipt() plus a document builder (tables, auto page-breaks). Runs on Node, edge and the browser.

> **NOTE**
> When to use — Generating invoices, receipts and simple documents server-side without a headless browser.

Install: npm i @lacspace/pdf

Runtime: zero dependencies · isomorphic

Example

```
invoice({ number, from, to, items })
// -> Uint8Array (PDF)
```

> **TIP**
> It builds the PDF bytes directly, so it's fast and dependency-free — ideal for serverless.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/pdf · source https://github.com/lacspace/npm-packages/tree/main/pdf

@lacspace/webhooks

Both directions: sign & deliver outgoing webhooks with retries + backoff, and verify incoming ones (timing-safe, replay-protected) with ready-made Stripe / GitHub / Shopify presets — plus event ids & idempotency for exactly-once handlers. A zero-dep alternative to hosted svix.

> **NOTE**
> When to use — Signing outgoing webhooks and verifying incoming ones (Stripe/GitHub-style HMAC).

Install: npm i @lacspace/webhooks

Runtime: built on @lacspace/crypto

Example

```
await verify(rawBody, header, { secret })
// -> { valid: true }
```

> **TIP**
> Verify the signature AND the timestamp to prevent replay attacks; the verifier checks both.

> **WARNING**
> Use a constant-time comparison (the package does) — never compare signatures with `===`.

Links: npm https://www.npmjs.com/package/@lacspace/webhooks · source https://github.com/lacspace/npm-packages/tree/main/webhooks

@lacspace/idempotency

The 'don't double-charge, don't send twice' primitive. Run any operation at-most-once per idempotency key and replay the stored result on retries — concurrency-safe (in-flight de-dupe + atomic create), with request fingerprinting and a pluggable store. Framework-agnostic (every other lib is Hono/AWS-locked).

> **NOTE**
> When to use — Making an operation safe to call more than once — payment capture, order creation, webhook handling — under retries or duplicate requests.

Install: npm i @lacspace/idempotency

Runtime: zero dependencies · isomorphic

Example

```
await idempotent(key, () => charge(order))
// -> { value, replayed }
```

> **TIP**
> Register the in-flight promise synchronously (before any await) so concurrent same-key calls share one execution — the package does this for you.

> **TIP**
> Return the same stored result for a repeated key within the TTL window.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> The default store is in-memory (per process). For multi-instance deployments, back it with a shared store (e.g. Redis).

Links: npm https://www.npmjs.com/package/@lacspace/idempotency · source https://github.com/lacspace/npm-packages/tree/main/idempotency

Data Kit — export & parse

Get data in and out — real Excel files and correct CSV — with no heavy libraries and no headless browser.

In this kit

- **@lacspace/xlsx** — Excel export, zero-dep
- **@lacspace/csv** — RFC 4180 CSV done right

@lacspace/xlsx

Write real .xlsx files with zero dependencies and no headless browser — a spreadsheet is just a ZIP of XML, so we build the bytes directly (same trick as @lacspace/pdf). Objects or arrays, correct types incl. real Excel dates, bold headers, column widths and multiple sheets. SheetJS/exceljs are heavy; this is tiny and isomorphic.

> **NOTE**
> When to use — Exporting real `.xlsx` spreadsheets (correct types, dates, multiple sheets) without SheetJS/exceljs.

Install: npm i @lacspace/xlsx

Runtime: zero dependencies · isomorphic

Example

```
jsonToXlsx(rows)
// -> Uint8Array (.xlsx)
```

> **TIP**
> Pass an array of objects; headers, column widths and Excel-native dates are handled for you.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/xlsx · source https://github.com/lacspace/npm-packages/tree/main/xlsx

@lacspace/csv

CSV looks trivial until commas-in-quotes, quotes-in-quotes, newlines-in-cells and CRLF. This handles the RFC 4180 edge cases correctly, returns typed row objects, stringifies with minimal quoting, and auto-detects

comma/tab/semicolon — in a few hundred bytes.

> **NOTE**
> When to use — Parsing and stringifying CSV correctly — quotes, commas-in-cells, CRLF and delimiter detection.

Install: npm i @lacspace/csv

Runtime: zero dependencies · isomorphic

Example

```
parse("name,age\nAda,36")
// -> [{ name, age }]
```

> **TIP**
> It returns typed row objects and auto-detects comma/tab/semicolon delimiters.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/csv · source https://github.com/lacspace/npm-packages/tree/main/csv

LACSPACE DEVELOPER DOCS

DX Kit — everyday utilities

The small helpers every UI rebuilds — human-readable formatting and colour maths — as tiny, typed, dependency-free packages.

In this kit

- **@lacspace/humanize** — Human-readable formatting
- **@lacspace/color** — Colour maths + WCAG contrast
- **@lacspace/money** — Money without float bugs

@lacspace/humanize

Bytes (1.5 KB), durations (1d 1h), relative time (3 hours ago), ordinals (21st), compact numbers (1.2M), pluralize, grammatical lists and truncation — one tiny typed package instead of installing five (pretty-bytes + ms + humanize-duration + …).

> **NOTE**
> When to use — Turning raw numbers, sizes, durations and dates into human-readable strings in the UI.

Install: npm i @lacspace/humanize

Runtime: zero dependencies · isomorphic

Example

```
relativeTime(Date.now() - 3.6e6)
// -> "1 hour ago"
```

> **TIP**
> One package replaces pretty-bytes + ms + humanize-duration + ordinal + pluralize.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/humanize · source https://github.com/lacspace/npm-packages/tree/main/humanize

@lacspace/color

Parse hex/rgb/hsl, convert between them, lighten/darken/mix/alpha/rotate — and, crucially, check WCAG contrast so your UI is actually readable (isReadable, readableTextColor). The colour toolkit every app

rebuilds, dependency-free.

> **NOTE**
> When to use — Colour maths in a design system — convert, mix, and (crucially) check WCAG contrast for readability.

Install: npm i @lacspace/color

Runtime: zero dependencies · isomorphic

Example

```
contrast("#000", "#fff")
// -> 21
```

> **TIP**
> `isReadable()` / `readableTextColor()` help you pick accessible text colours programmatically.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/color · source https://github.com/lacspace/npm-packages/tree/main/color

@lacspace/money

0.1 + 0.2 !== 0.3 — so never store money as a float. Keeps amounts as integer minor units, refuses to add different currencies, splits a bill without losing a cent (allocate), handles zero- and three-decimal currencies, and formats with Intl. Immutable, typed.

> **NOTE**
> When to use — Any prices, totals, or financial math — where floating-point cents bugs and currency mix-ups are unacceptable.

Install: npm i @lacspace/money

Runtime: zero dependencies · isomorphic

Example

```
money(10,"USD").allocate([1,1,1])
// -> [$3.34,$3.33,$3.33]
```

Key API

money(major, ccy) / Money.fromMinor(minor, ccy) Construct.

.add/.subtract/.multiply/.allocate/.split Currency-safe operations.

.format(locale?) / .toMinor() / .toJSON() Output.

> **TIP**
> Store and compute in integer minor units; only format for display with `.format(locale)`.

> **TIP**
> Split bills with `.allocate([ratios])` / `.split(n)` — the remainder is distributed so the total is always exact.

> **TIP**
> Zero-decimal (JPY) and three-decimal (BHD/KWD) currencies are handled automatically.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> Arithmetic across different currencies throws — convert first with your own FX rate.

> **WARNING**
> `.toMajor()` returns a JS number for convenience; prefer `.format()` for display to avoid float rounding.

Links: npm https://www.npmjs.com/package/@lacspace/money · source https://github.com/lacspace/npm-packages/tree/main/money

Utils Kit — the missing standard library

The tiny helpers every project reaches for — unique IDs, resilient calls and string-case conversion — typed and dependency-free.

In this kit

- **@lacspace/id** — UUID v4/v7, Nano-ID & short
- **@lacspace/retry** — Retry, timeout, circuit breaker
- **@lacspace/case** — String-case conversion
- **@lacspace/cache** — LRU + TTL + stale-while-revalidate

@lacspace/id

Every ID kind in one package — classic UUID v4, the time-sortable UUID v7 (a superb index-friendly database key), Nano-ID-style and short URL-safe codes, plus prefixed ids like user_9f8c…. Cryptographically random (Web Crypto), monotonic within a millisecond.

> **NOTE**
> When to use — Generating unique ids — a time-sortable UUID v7 makes an excellent, index-friendly database key.

Install: npm i @lacspace/id

Runtime: zero dependencies · isomorphic

Example

```
uuidv7()
// -> "0192e7a1-…-7abc-…" (sortable)
```

> **TIP**
> Prefer `uuidv7()` for primary keys (sortable by creation time); use short ids for public URLs.

> **TIP**
> Cryptographically random via Web Crypto, monotonic within a millisecond.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/id · source https://github.com/lacspace/npm-packages/tree/main/id

@lacspace/retry

Resilience for flaky calls: retry() with exponential backoff + full jitter (shouldRetry/onRetry/AbortSignal), withTimeout() that cancels slow work, and a CircuitBreaker that fails fast while a dependency is down then recovers. Wrap any fetch / DB / queue call.

> **NOTE**
> When to use — Wrapping flaky network, DB or queue calls so transient failures recover automatically instead of surfacing as errors.

Install: npm i @lacspace/retry

Runtime: zero dependencies · isomorphic

Example

```
retry(() => fetch(url), { retries: 4 })
// -> Promise<T>
```

> **TIP**
> `retry(fn, { retries, factor })` uses exponential backoff with full jitter to avoid thundering-herd retries.

> **TIP**
> Use `shouldRetry` to retry only on transient errors (e.g. 5xx / network), not on 4xx.

> **TIP**
> Wrap slow calls in `withTimeout()` and protect a failing dependency with `CircuitBreaker`.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> Only retry idempotent operations — retrying a non-idempotent POST can double-charge. Pair with @lacspace/idempotency.

Links: npm https://www.npmjs.com/package/@lacspace/retry · source https://github.com/lacspace/npm-packages/tree/main/retry

@lacspace/case

camelCase, PascalCase, snake_case, kebab-case, CONSTANT_CASE, Title Case, Sentence case — correct on the tricky inputs (acronyms like XMLHttpRequest, numbers like v2, mixed separators). One tiny typed package with a shared words() splitter.

Install: npm i @lacspace/case

Runtime: zero dependencies · isomorphic

Example

```
kebabCase("XMLHttpRequest")
// -> "xml-http-request"
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/case · source https://github.com/lacspace/npm-packages/tree/main/case

@lacspace/cache

An in-memory cache that does the three things you actually want — bound the size (LRU), expire entries (TTL) and hide latency (stale-while-revalidate) — plus wrap()/memoize() that cache any async function and de-duplicate concurrent in-flight calls. 100 callers, one fetch.

> **NOTE**
> When to use — Caching expensive async work (DB/API calls) in-process, with size bounds and freshness control — and de-duplicating stampedes.

Install: npm i @lacspace/cache

Runtime: zero dependencies · isomorphic

Example

```
cache.wrap(key, fetchUser, { ttl })
// -> de-duped + cached
```

Key API

createCache({ max, ttl }) -> get/set/has/delete/clear/keys/size/wrap.

cache.wrap(key, fn, { ttl, staleWhileRevalidate }) Cached async with de-dup + SWR.

memoize(fn, opts) Memoized function keyed by arguments, with `.cache`.

> **TIP**
> `cache.wrap(key, fn)` de-duplicates concurrent callers — 100 simultaneous misses trigger exactly one `fn()`.

TIP
Add `staleWhileRevalidate` to serve the cached value instantly after expiry while refreshing in the background.

TIP
`memoize(fn, { ttl })` wraps a function and keys by its arguments; use `.cache` for manual control.

TIP
Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

WARNING
This is a per-process, in-memory cache — it doesn't share across serverless instances or survive restarts. For shared state, back it with Redis via your own store.

WARNING
Cache keys are strings; for object args, `memoize` uses `JSON.stringify` by default — pass a custom `key` for stable hashing.

Links: npm https://www.npmjs.com/package/@lacspace/cache · source https://github.com/lacspace/npm-packages/tree/main/cache

App Kit — ship a polished app

The pieces every product app needs — schema validation, spam-proof forms, dynamic social images and a lively UI — powering create-lacspace-app out of the box.

In this kit

- **@lacspace/validate** — Zod's ergonomics, zero deps
- **@lacspace/form** — Typed forms + spam guard
- **@lacspace/og** — Dynamic OG share images
- **@lacspace/ui** — React kit that feels alive
- **@lacspace/markdown** — Markdown -> HTML, safe
- **@lacspace/analytics-lite** — Cookieless analytics

@lacspace/validate

parse / safeParse, objects, arrays, enums, unions and full type inference — plus coercion ("42" -> 42, "true" -> true) so one schema validates a JSON body and an HTML form. Field errors flatten() straight into a form. The validation you reach for, small enough to drop into any edge function.

> **NOTE**
> When to use — Validating anything untrusted — form bodies, API payloads, env vars, query strings — with type inference, in a package small enough for edge functions.

Install: npm i @lacspace/validate

Runtime: zero dependencies · isomorphic

Example

```
User.safeParse(input)
// -> { success, data | error }
```

Key API

v.string()/number()/boolean()/object()/array()/enum()/union() Schema builders.

.optional()/.default()/.refine()/.transform() Modifiers.

parse() / safeParse() / Infer<> Validate and infer types.

> **TIP**
> Use `v.coerce.number()` / `v.coerce.boolean()` for FormData and query strings, which arrive as strings.

> **TIP**
> `schema.safeParse(x)` returns `{ success, data | error }`; call `error.flatten()` to get `{ field: message }` for forms.

> **TIP**
> `Infer<typeof schema>` gives you the exact TypeScript type — one source of truth for runtime and compile time.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> `.strict()` rejects unknown keys — pair it with `@lacspace/form`, which strips internal fields (honeypot, timestamp) before validation.

Links: npm https://www.npmjs.com/package/@lacspace/validate · source https://github.com/lacspace/npm-packages/tree/main/validate

@lacspace/form

Turn a FormData into typed, validated data with a honeypot + timing spam guard, and get back either your data or per-field errors ready to re-render. createForm().action matches Next.js useActionState, so a full contact form is a few lines. Framework-agnostic, zero-dep.

> **NOTE**
> When to use — Handling form submissions on the server (Next.js Server Actions) with validation and spam protection, without a form library.

Install: npm i @lacspace/form

Runtime: zero dependencies · isomorphic

Example

```
contact.action(prev, formData)
// -> { ok, data | errors }
```

Key API

createForm(opts) -> { handle, action } bound to a schema + spam options.

formDataToObject(fd) FormData -> plain object (repeated keys -> arrays).

honeypotProps(name) / timestampValue() Client helpers for spam protection.

> **TIP**
> `createForm({ schema }).action` matches `useActionState`, so a full contact form is a few lines.

> **TIP**
> Add a `honeypot` field and `minSubmitMs` to block bots; internal fields are stripped before validation.

> **TIP**
> The result echoes `values`, so you can re-render the form with what the user typed on error.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> The honeypot input must be visually hidden but present in the DOM — use the provided `honeypotProps()` helper.

Links: npm https://www.npmjs.com/package/@lacspace/form · source https://github.com/lacspace/npm-packages/tree/main/form

@lacspace/og

A share-card design system you configure once and call per page — auto-fitting titles, eyebrows, badges, logos and gradients. Renders two ways from the same options: a next/og element tree (real PNG) and a zero-dependency SVG for previews, emails and icons.

> **NOTE**
> When to use — Generating a share image per page — a real PNG via `next/og`, or a standalone SVG for emails, README banners and previews.

Install: npm i @lacspace/og

Runtime: zero dependencies · isomorphic

Example

```
ogCard({ title, from, to })
// -> 1200×630 social card
```

Key API

ogCard(opts) next/og element tree (title, subtitle, eyebrow, badge, logo, gradient).

ogSvg(opts) / ogSvgDataUri(opts) Zero-dependency SVG string / data URI.

fitFontSize(title) The auto-fit sizing math for your own layouts.

> **TIP**
> `ogCard()` returns a Satori-compatible node — pass it straight to `new ImageResponse(ogCard({...}) as any)`.

> **TIP**
> Titles auto-fit: long headlines shrink to stay on ~3 lines, short ones stay bold. Override with `fitFontSize`.

> **TIP**
> For non-PNG contexts use `ogSvg()` / `ogSvgDataUri()` — no browser, no Satori, fully deterministic.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

> **WARNING**
> `og:image` scrapers want PNG/JPEG — use `ogCard` + `next/og` for the actual social image; keep `ogSvg` for previews and emails.

Links: npm https://www.npmjs.com/package/@lacspace/og · source https://github.com/lacspace/npm-packages/tree/main/og

@lacspace/ui

Scroll reveals, animated counters, gradient text, tilt cards, marquees, a typewriter and a ?K command palette — no animation library, no CSS import. Every component respects prefers-reduced-motion and takes a className, so it drops into any Tailwind + Next.js app. React is the only peer dependency.

> **NOTE**
> When to use — Adding motion and interactive polish to a React/Next.js app without pulling in a heavy animation library or a separate command-palette package.

Install: npm i @lacspace/ui

Runtime: React peer dep · nothing else

Example

```
<Reveal><Counter value={12480} /></Reveal>
// -> animated on view
```

Key API

<Reveal> / <Counter> / <GradientText> Scroll-reveal, count-up and gradient text.

<TiltCard> / <Marquee> / <Typewriter> Micro-interactions and motion.

<CommandPalette> + useInView() ?K palette and the IntersectionObserver hook.

> **TIP**
> The package ships `"use client"`, so you can import it directly into Server Components.

> **TIP**
> Everything honours `prefers-reduced-motion` automatically — no extra work for accessibility.

> **TIP**
> Mount `<CommandPalette items={…} />` once in your layout; it opens on ?K / Ctrl-K.

> **WARNING**
> React is a peer dependency — it uses your app's React, so there's no duplicate-React risk, but React ? 18 is required.

Links: npm https://www.npmjs.com/package/@lacspace/ui · source https://github.com/lacspace/npm-packages/tree/main/ui

@lacspace/markdown

Everything a blog or docs page needs — headings with anchor ids, nested & task lists, fenced code, blockquotes, GFM tables, images and links — rendering to clean HTML with source HTML escaped by default. Includes extractHeadings() for a table of contents. It's what the blog template in create-lacspace-app uses.

> **NOTE**
> When to use — Rendering Markdown to HTML for a blog or docs site, with a table-of-contents extractor, without a heavy parser.

Install: npm i @lacspace/markdown

Runtime: zero dependencies · isomorphic

Example

```
markdownToHtml(post)
// -> "<h1 id=…>…"
```

Key API

markdownToHtml(md, opts) Markdown -> HTML string.

extractHeadings(md) Heading outline for a TOC.

slugify(text) GitHub-style anchor slug.

> **TIP**
> Use `headingOffset: 1` when the page already has an `<h1>`, so document `#` becomes `<h2>`.

> **TIP**
> `extractHeadings(md)` gives you `{ level, text, id }[]` for an on-this-page menu; `slugify()` matches the anchor ids.

TIP
Pair it with the `blog` and `docs` templates in create-lacspace-app for a complete content site.

TIP
Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

WARNING
Raw HTML in the source is **escaped** , not passed through — this is a safety feature. If you need embedded HTML, sanitize and inject it yourself.

Links: npm https://www.npmjs.com/package/@lacspace/markdown · source https://github.com/lacspace/npm-packages/tree/main/markdown

@lacspace/analytics-lite

Privacy-first page views + custom events sent to your own endpoint — no cookies, no localStorage IDs, no cross-site tracking, so no consent banner. Respects Do-Not-Track, auto-tracks SPA navigation, and uses sendBeacon so events survive page unload. Sends only a path, referrer host and an ephemeral per-load id.

NOTE
When to use — Privacy-first product analytics where you own the data and don't want a consent banner or a third-party script.

Install: npm i @lacspace/analytics-lite

Runtime: zero dependencies · isomorphic

Example

```
analytics.track("signup")
// -> -> your /collect endpoint
```

Key API

createAnalytics({ endpoint, siteId }) -> pageview/track/autoTrack/enabled.

.pageview(path?) / .track(name, props?) Manual events.

.autoTrack() Auto page views on route change; returns a cleanup fn.

TIP
Call `analytics.autoTrack()` once from a client component in your layout to record SPA navigations automatically.

TIP
It sends only a path, referrer host and an ephemeral per-load id — no cookies, no persistent identifier.

TIP
Point `endpoint` at your own collector route; it uses `sendBeacon` so events survive page unload.

TIP
Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

WARNING
It respects `navigator.doNotTrack` by default (`enabled` becomes false). Set `respectDNT: false` only if your policy allows.

WARNING
You still need a backend endpoint to receive and store the events.

Links: npm https://www.npmjs.com/package/@lacspace/analytics-lite · source https://github.com/lacspace/npm-packages/tree/main/analytics-lite

LACSPACE DEVELOPER DOCS

WebKit — for web apps & backends

The utilities every web app and backend re-implements — config, rate limiting, feature flags and Next.js — done once, typed, and dependency-free.

In this kit

- **@lacspace/flags** — Feature flags & A/B — no SaaS
- **@lacspace/env** — Typed env variables
- **@lacspace/rate-limit** — Rate limiting
- **@lacspace/next** — Next.js App Router integration

@lacspace/flags

Feature flags and A/B experiments with no vendor and no infrastructure: you own the config, this evaluates it. Deterministic bucketing (same user always gets the same result), percentage rollouts, targeting rules and weighted variants. Synchronous — evaluate right in render.

> **NOTE**
> When to use — Feature flags, gradual rollouts and A/B tests without a SaaS.

Install: npm i @lacspace/flags

Runtime: zero dependencies · isomorphic

Example

```
flags.isEnabled("new-ui", { key: user.id })
// -> true / false
```

> **TIP**
> A flag with rules but no rollout defaults to off for unmatched users — set an explicit rollout for a percentage.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/flags · source https://github.com/lacspace/npm-packages/tree/main/flags

@lacspace/env

Declare a schema, validate process.env once at boot, and get a typed frozen object. Missing or malformed vars throw one clear error before the app serves traffic. A zero-dep t3-env / envalid alternative.

> **NOTE**
> When to use — Validating and typing environment variables at startup so misconfig fails fast, not at 2 a.m.

Install: npm i @lacspace/env

Runtime: zero dependencies

Example

```
createEnv({ PORT: port({ default: 3000 }) }).PORT
// -> 3000 (number)
```

> **TIP**
> Define the schema once; `createEnv()` returns a typed, frozen object and aggregates all errors.

> **TIP**
> Use it in both server and build steps to catch missing vars early.

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/env · source https://github.com/lacspace/npm-packages/tree/main/env

@lacspace/rate-limit

Fixed-window, sliding-window and token-bucket algorithms over a pluggable store (in-memory built in; bring your own Redis). Standard RateLimit-* headers. Drop into any API route, middleware or edge function.

> **NOTE**
> When to use — Protecting endpoints (login, contact, APIs) from abuse with token-bucket, fixed- or sliding-window limits.

Install: npm i @lacspace/rate-limit

Runtime: zero dependencies

Example

```
await rateLimit({ limit: 10, windowMs: 60000 }).check(ip)
// -> { success, remaining }
```

TIP
Return `rateLimitHeaders()` so clients see `RateLimit-*` and `Retry-After`.

TIP
Key by user id when authenticated, IP otherwise.

TIP
Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

WARNING
The default store is in-memory per instance; use a shared store (Redis) for accurate limits across a fleet.

Links: npm https://www.npmjs.com/package/@lacspace/rate-limit · source https://github.com/lacspace/npm-packages/tree/main/rate-limit

@lacspace/next

The server-side companion to @lacspace/react: an authenticated SDK client from cookies, Route Handler & Server Action wrappers, cookie helpers, a one-line middleware auth guard and double-submit CSRF protection. Next.js 14 & 15.

Install: npm i @lacspace/next

Runtime: built on @lacspace/sdk

Example

```
const lac = await createServerClient()
// -> authed SDK
```

Links: npm https://www.npmjs.com/package/@lacspace/next · source https://github.com/lacspace/npm-packages/tree/main/next

Nepal toolkit

Batteries-included helpers for building Nepal-facing apps.

In this kit

- **@lacspace/nepali-date** — Bikram Sambat ? Gregorian
- **@lacspace/nepali-utils** — Everyday Nepal helpers

@lacspace/nepali-date

Convert between BS and AD dates with Devanagari formatting. Range BS 1970–2086, cross-checked against established datasets across 42,000+ conversions.

Install: npm i @lacspace/nepali-date

Runtime: zero dependencies

Example

```
new NepaliDate().formatNepali()
// -> ???? ??? ?, ??????
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/nepali-date · source https://github.com/lacspace/npm-packages/tree/main/nepali-date

@lacspace/nepali-utils

NPR currency formatting (lakh/crore grouping), Devanagari numerals, amount-in-words for invoices, phone carrier detection & validators, all 77 districts by province, and a land-area converter (ropani-aana-paisa ? bigha-kattha-dhur ? m²/ft²).

Install: npm i @lacspace/nepali-utils

Runtime: zero dependencies

Example

```
formatNPR(1234567.5)
// -> Rs. 12,34,567.50
```

> **TIP**
> Zero-dependency and isomorphic — safe to import on the server, in the browser, on the edge and in React Native. It tree-shakes, so you only ship what you import.

Links: npm https://www.npmjs.com/package/@lacspace/nepali-utils · source https://github.com/lacspace/npm-packages/tree/main/nepali-utils

LACSPACE DEVELOPER DOCS

React Kit — hooks, state, data & UX

A complete React layer that pairs with any app: essential hooks, tiny global state, data fetching, theming, keyboard shortcuts and list virtualization — all zero-dependency and SSR-safe.

In this kit

- **@lacspace/hooks** — 24+ essential hooks
- **@lacspace/store** — Global state in ~1 KB
- **@lacspace/query** — Data fetching & cache
- **@lacspace/theme** — Dark / light / system
- **@lacspace/hotkeys** — Keyboard shortcuts
- **@lacspace/virtual** — List virtualization
- **@lacspace/react** — SDK hooks & provider

@lacspace/hooks

The hooks you reach for on every project — useLocalStorage, useDebounce, useMediaQuery, useOnClickOutside, useCopyToClipboard, useInterval, useIntersectionObserver and 20 more. Every one SSR-safe, fully typed and tree-shakeable.

> **NOTE**
> When to use — Any React or Next.js app that keeps re-implementing the same utility hooks — persisted state, debouncing, media queries, click-outside, clipboard. Import the battle-tested version instead.

Install: npm i @lacspace/hooks

Runtime: zero deps · react 18+

Example

```
const [v, setV] = useLocalStorage("key", 0)
```

Key API

useLocalStorage(key, initial) Persisted, cross-tab state -> [value, setValue, remove].

useDebounce(value, ms) / useThrottle Debounced or throttled value; useDebouncedCallback for actions.

useMediaQuery(q) / useWindowSize SSR-safe responsive state.

useOnClickOutside(ref, fn) / useHover DOM interaction hooks.

useCopyToClipboard() [copied, copy(text)] with a graceful fallback.

useInterval / useTimeout / useIdle Pausable timers and idle detection.

+ 18 more usePrevious, useToggle, useCounter, useDisclosure, useIntersectionObserver, useEventListener,

useOnlineStatus, useLockBodyScroll, useUpdateEffect…

> **TIP**
> Every hook is SSR-safe: on the server it returns a sensible default (e.g. `useMediaQuery` is `false`, storage hooks return your initial value) and hydrates in an effect — no `window is not defined`.

> **TIP**
> `useLocalStorage`/`useSessionStorage` sync across components and browser tabs automatically via the `storage` event; the returned `remove()` clears the key.

> **TIP**
> Reach for `useDebouncedCallback` (not just `useDebounce`) when you need to debounce an action like search-as-you-type — it exposes a `.cancel()`.

> **TIP**
> It tree-shakes: importing one hook ships only that hook.

> **WARNING**
> This is a hooks library with no `"use client"` directive — call the hooks from your own Client Components (files with `"use client"`), never from Server Components.

> **WARNING**
> `useLocalStorage` stores JSON — values must be serializable (no functions, `Map`, `Set` or class instances).

Links: npm https://www.npmjs.com/package/@lacspace/hooks · source https://github.com/lacspace/npm-packages/tree/main/hooks

@lacspace/store

Create a store, use selectors, no provider — built on useSyncExternalStore with a persist middleware and shallow equality. A Zustand-lite that also works outside React.

> **NOTE**
> When to use — Sharing state across components without prop-drilling or a Context provider — and when Redux/MobX feel like too much. Also great for state you need to read or write from outside React.

Install: npm i @lacspace/store

Runtime: zero deps · react 18+

Example

```
const useStore = create((set) => ({ n: 0 }))
```

Key API

create(initializer) Returns a `useStore` hook + attached getState/setState/subscribe.

createStore(initializer) The vanilla, framework-agnostic store (no React).

useStore(selector?, equalityFn?) Subscribe to the whole state or a selected slice.

persist(initializer, options) localStorage/sessionStorage middleware with partialize + version.

shallow(a, b) One-level equality for multi-value selectors.

> **TIP**
> `create((set, get) => ({ ... }))` returns a hook that is ALSO the store API — call `useStore.getState()` / `useStore.setState()` / `useStore.subscribe()` anywhere, even outside components.

> **TIP**
> Pass a selector to subscribe to just one slice: `useStore((s) => s.count)` — the component re-renders only when that slice changes.

> **TIP**
> Selecting multiple values? Return an object and pass `shallow` as the equality fn to avoid needless re-renders: `useStore((s) => ({ a: s.a, b: s.b }), shallow)`.

> **TIP**
> Wrap the initializer in `persist(fn, { name })` to save to localStorage; use `partialize` to persist only part of the state.

> **WARNING**
> `setState` shallow-merges by default — pass `replace: true` to swap the whole state object.

> **WARNING**
> `persist` reads storage on creation (SSR-safe); the first server render uses initial state, so guard against layout shift if persisted values change what renders.

Links: npm https://www.npmjs.com/package/@lacspace/store · source https://github.com/lacspace/npm-packages/tree/main/store

@lacspace/query

useQuery + useMutation with a shared cache, request de-duplication, stale-while-revalidate and focus/reconnect revalidation — an SWR-lite in ~2 KB.

> **NOTE**
> When to use — Fetching, caching and revalidating server data in React without hand-rolling loading/error state — a lighter alternative to React Query or SWR when you want zero dependencies.

Install: npm i @lacspace/query

Runtime: zero deps · react 18+

Example

```
const { data } = useQuery("/me", fetchMe)
```

Key API

useQuery(key, fetcher, options?) -> { data, error, isLoading, isFetching, refetch }.

useMutation(fn, options?) -> { mutate, mutateAsync, isPending, data, error, reset }.

mutate(key, data?, opts?) Revalidate or optimistically update a cache entry.

prefetchQuery(key, fetcher) Warm the cache before render.

getQueryData / setQueryData / clearQueryCache Read, write and clear the shared cache.

> **TIP**
> The cache is keyed and shared: two components calling `useQuery` with the same key share one request and re-render together. Use array keys for params: `useQuery(["user", id], …)`.

> **TIP**
> Stale-while-revalidate is the default — cached data shows instantly while a fresh copy loads in the background. Tune with `staleTime`.

> **TIP**
> Disable a query until a dependency is ready by passing a `null`/`false` key: `useQuery(id ? ["user", id] : null, …)`.

> **TIP**
> After a mutation, call `mutate(key)` to revalidate, or `setQueryData(key, data)` for an optimistic update.

> **WARNING**
> The cache lives in module memory — it resets on full page reload and is per-tab. It's a client cache, not persistence.

> **WARNING**
> Fetchers must throw on failure (e.g. non-2xx) for `error`/`isError` to populate — a resolved promise is treated as success.

Links: npm https://www.npmjs.com/package/@lacspace/query · source https://github.com/lacspace/npm-packages/tree/main/query

@lacspace/theme

A tiny ThemeProvider (with a built-in no-flash script), a useTheme hook and getThemeScript(). Persists to storage, follows the OS, toggles a class or data-attribute — a framework-agnostic next-themes-lite.

> **NOTE**
> When to use — Adding dark / light / system theming to any React app — Next.js App Router included — with no flash of the wrong theme on first paint. A framework-agnostic next-themes alternative.

Install: npm i @lacspace/theme

Runtime: zero deps · react 18+

Example

```
const { theme, setTheme } = useTheme()
```

Key API

<ThemeProvider …> defaultTheme, storageKey, themes, attribute, enableSystem, disableTransitionOnChange, enableNoFlashScript (built-in no-flash).

useTheme() -> { theme, setTheme, resolvedTheme, systemTheme, themes }.

getThemeScript(options?) Self-contained no-flash IIFE string (advanced; the provider injects one by default).

> **TIP**
> Just wrap your app in `<ThemeProvider>` — it renders a tiny no-flash script for you, so the correct theme is applied before first paint with nothing else to wire up. Add `suppressHydrationWarning` to your `<html>`.

> **TIP**
> Read/set with `useTheme()` -> `{ theme, setTheme, resolvedTheme }`. `resolvedTheme` turns "system" into the concrete "light"/"dark".

> **TIP**
> Use `attribute="class"` for Tailwind's `dark:` variant, or a `data-theme` attribute for CSS custom-property theming.

> **TIP**
> Set `disableTransitionOnChange` to avoid every transition firing at once when the theme flips.

> **WARNING**
> The package ships `"use client"`, so `getThemeScript()` is a client export — don't call it from a Server Component. You rarely need it: `<ThemeProvider>` already injects the no-flash script. To inject it yourself, set `enableNoFlashScript={false}` and render `getThemeScript()` from a client component or plain HTML.

> **WARNING**
> If you inject the script manually, match its `storageKey`, `attribute` and `defaultTheme` to the provider's, or the pre-paint script and React will disagree.

Links: npm https://www.npmjs.com/package/@lacspace/theme · source https://github.com/lacspace/npm-packages/tree/main/theme

@lacspace/hotkeys

useHotkeys with combos (mod+k), key sequences (g then d), scopes and pretty display formatting (?K). SSR-safe and respects form fields.

> **NOTE**
> When to use — Adding keyboard shortcuts to a React app — a ?K palette, single keys, or Gmail-style `g then i` sequences — with correct cross-platform modifier handling.

Install: npm i @lacspace/hotkeys

Runtime: zero deps · react 18+

Example

```
useHotkeys("mod+k", openPalette)
```

Key API

useHotkeys(keys, handler, options?, deps?) Bind one or many combos/sequences; options for scope, target, form-tags.

parseHotkey(str) / matchesHotkey(e, str) Parse and test combos yourself.

formatHotkey(str, { mac? }) Pretty, platform-aware display string.

enableScope / disableScope / useHotkeysScopes Turn groups of shortcuts on and off.

> **TIP**
> Use `mod` instead of ctrl/cmd: it maps to ? on macOS and Ctrl elsewhere, so one binding works everywhere.

> **TIP**
> Sequences are written `"g then d"` (or `"g d"`) and fire when pressed in order within ~1s.

> **TIP**
> Render the hint with `formatHotkey("mod+k")` — it shows `?K` on Mac and `Ctrl+K` on Windows/Linux automatically.

> **TIP**
> By default shortcuts ignore keystrokes typed into inputs/textareas — set `enableOnFormTags: true` if you need them there.

> **WARNING**
> Browsers reserve some combos (?W, ?T, ?N) — you can't reliably override those; pick app-safe keys.

> **WARNING**
> `preventDefault` is on by default — turn it off for a binding if you still want the browser's native behaviour.

Links: npm https://www.npmjs.com/package/@lacspace/hotkeys · source https://github.com/lacspace/npm-packages/tree/main/hotkeys

@lacspace/virtual

Render 100k rows smoothly — headless windowing with fixed or dynamically-measured sizes, overscan and scroll-to-index. useVirtualizer in ~2 KB.

> **NOTE**
> When to use — Rendering long lists or grids (hundreds to hundreds of thousands of rows) without janking the browser — only the visible rows are mounted. A zero-dependency TanStack-Virtual alternative.

Install: npm i @lacspace/virtual

Runtime: zero deps · react 18+

Example

```
const v = useVirtualizer({ count, ... })
```

Key API

useVirtualizer(options) count, getScrollElement, estimateSize, overscan, horizontal, gap.

getVirtualItems() The rows to render -> { index, start, size, end, key }.

getTotalSize() Total scrollable size for the spacer element.

measureElement(el) / scrollToIndex(i) Dynamic measurement and programmatic scrolling.

> **TIP**
> It's headless: you own the markup. Give the scroll container a fixed height + `overflow:auto`, render a spacer of `getTotalSize()`px, and absolutely-position each item from `getVirtualItems()` at `transform: translateY(item.start)`.

> **TIP**
> For variable row heights, set `data-index={item.index}` and `ref={v.measureElement}` on each row — it measures real heights and corrects the layout.

> **TIP**
> Tune `overscan` (default 5) to render a few extra rows above/below the viewport for smoother fast-scrolling.

> **TIP**
> `scrollToIndex(i, { align })` jumps to any row, even one far outside the current window.

> **WARNING**
> The scroll element must actually scroll (a bounded height with `overflow:auto`); if the page itself scrolls instead, offsets won't track.

> **WARNING**
> `estimateSize` should be close to the real size — a wildly wrong estimate makes the scrollbar jump as rows are measured.

Links: npm https://www.npmjs.com/package/@lacspace/virtual · source https://github.com/lacspace/npm-packages/tree/main/virtual

@lacspace/react

Wrap your app once, then use useAuth, useQuery and useLacspace anywhere — all sharing one authenticated Lacspace SDK client. React 18+.

Install: npm i @lacspace/react

Runtime: built on @lacspace/sdk

Example

```
const { user } = useAuth()
```

Links: npm https://www.npmjs.com/package/@lacspace/react · source https://github.com/lacspace/npm-packages/tree/main/react

LACSPACE DEVELOPER DOCS