URL Shortner

URL Shortener with Analytics — Project Specification

1. Overview

This project is a **simple, single-service Node.js application** that provides:

- Shortening of long URLs into short codes
- Redirect handling
- Click tracking with basic analytics:
  - Total clicks per link
  - Referrers
  - Browser & OS
  - Device type (desktop / mobile / tablet)
  - Clicks over time
- A **simple dashboard** to visualize analytics with charts

The project should be built as a **monolithic backend + simple frontend** (no microservices, no NATS, no Redis).

---

2. Tech Stack (Preferred)

Backend:

- Node.js (LTS)
- TypeScript
- Express.js (or similar HTTP framework)
- MongoDB (using Mongoose or any ORM/ODM)
- JWT-based auth (optional but recommended if users exist)

Frontend:

- Next.js or React (SPA)
- Tailwind CSS or simple CSS framework
- Chart library (e.g., Chart.js, Recharts, or similar)

Other:

- User-Agent parsing library (e.g., `ua-parser-js`)
- Simple logging (console + middleware)
- Environment-based configuration using `.env`

> Note: The AI can replace libraries/frameworks if needed, but must preserve all features and logic.

---

3. Core Concepts & Glossary

- **Short URL**: A short code that maps to a long destination URL.
- **Slug**: The unique short code (e.g., `abc123`) appended to the base domain (e.g., `https://sho.rt/abc123`).
- **Visit / Click**: A single access to a short URL which is recorded for analytics.
- **Referrer**: The HTTP referrer header indicating where the user came from (may be `null`).
- **User-Agent**: String to determine browser, OS, and device type.
- **User (Optional)**: If implemented, a user owns multiple short URLs.

---

4. Functional Requirements

4.1 URL Shortening

- User can submit a **long URL** and get a **short URL**.
- Optional fields when creating a short URL:
    - `customSlug` (string, optional, must be unique)

    - `title` (string, optional, for display in dashboard)
    - `expiresAt` (optional expiration datetime)
- System generates a **unique slug** if `customSlug` is not provided.
- Validation rules:
    - Long URL must be a valid URL with `http` or `https`.
    - `customSlug` must:
        - Be alphanumeric + or `_` only
        - Have length between 4 and 32 characters
    - Reject duplicate `customSlug`.

4.2 Redirect

- When user visits `/{slug}`, the system:
    1. Looks up the slug.
    2. Checks if:
        - Link exists
        - Link is not expired
        - Link is not disabled
    3. Records a **Visit** entry containing:
        - Timestamp
        - Referrer
        - IP address (mask or partial store)
        - Parsed browser
        - Parsed OS
        - Device type (mobile / desktop / tablet / unknown)
    4. Redirects user to the target URL using HTTP 302 or 307.

4.3 Analytics

For each short URL, system should provide:

- **Summary:**
    - Total clicks
    - Unique visitors (approximate: by unique IP or userAgent+IP combo)
    - Creation date
- **Time series:**
    - Clicks per day (for the last N days, e.g., 30)
- **Referrers:**
    - Top referrers with counts
- **Devices:**
    - Device type distribution: mobile, desktop, tablet, other
- **Browsers:**
    - Browser distribution: Chrome, Safari, Firefox, Edge, Other
- **Operating systems:**
    - OS distribution: Windows, macOS, Android, iOS, Linux, Other

4.4 Dashboard (Web UI)

- **Home / Links List:**
    - List all created short URLs (for that user or globally if no auth)
    - Columns:
        - Title (or long URL hostname)
        - Short URL
        - Total clicks
        - Created at
    - Actions:
        - View analytics

        - Copy short URL
        - Disable / enable link (optional)
- **Create Link Page:**
    - Form fields:
        - Long URL (required)
        - Custom slug (optional)
        - Title (optional)
        - Expiration date/time (optional)
    - Shows the generated short URL after creation.
- **Analytics Page (per link):**
    - Basic info: title, long URL, short URL, created date, total clicks.
    - Charts:
        - Line chart of clicks over time (daily)
        - Pie chart of devices
        - Pie or bar chart of browsers
        - Table of top referrers

---

5. Non-Functional Requirements

- Single Node.js process (no microservices).
- Clean separation of layers:
    - Routes / controllers
    - Services (business logic)
    - Models (database)
    - Utility helpers (e.g., slug generator, UA parser, analytics aggregation)
- Error handling middleware with consistent JSON error responses.
- Basic input validation for all endpoints.

- Code should be production-ready, readable, and well-structured.

---

6. Data Model Design

Below are **logical schemas** (MongoDB style). AI can adjust to specific ORM/ODM.

6.1 `Url` Collection

```
Url {
  _id: ObjectId;
  slug: string;            // unique, indexed
  longUrl: string;
  title?: string;
  createdAt: Date;
  updatedAt: Date;
  expiresAt?: Date | null;
  disabled: boolean;       // default false
  // If auth is added:
  // ownerId?: ObjectId;    // reference to User
}
```

Indexes:

- Unique index on `slug`
- Optional index on `createdAt`
- Optional compound index on `ownerId + createdAt`

6.2 `Visit` Collection

```
Visit {
  _id: ObjectId;
  urlId: ObjectId;         // ref to Url
  slug: string;            // denormalized for queries
  createdAt: Date;
  ipAddress?: string;       // optional or truncated for privacy
  referrer?: string | null;

  userAgent?: string | null;
  browser?: string | null;  // e.g., 'Chrome', 'Firefox'
  os?: string | null;       // e.g., 'Windows', 'Android'
  deviceType?: string | null; // 'desktop' | 'mobile' | 'tablet' | 'other';

  // Optional: country, region if GeoIP added later
}
```

Indexes:

- Index on `urlId`
- Index on `slug`
- Index on `createdAt`

6.3 `User` Collection (Optional)

If you want authentication:

```
User {
  _id: ObjectId;
  email: string;        // unique
  passwordHash: string; // if using local auth
  createdAt: Date;
}
```

---

7. API Design

All responses should be JSON.

For errors, use a consistent shape like:

```
{
  "error": {
    "message": "Human readable",
    "code": "SOME_CODE",
    "details": {}
  }
}
```

7.1 Create Short URL

POST `/api/urls`

Request body:

```
{
  "longUrl": "https://example.com/very/long/path",
  "customSlug": "my-custom",       // optional
  "title": "My Example Link",      // optional
  "expiresAt": "2026-01-01T00:00:00.000Z" // optional
}
```

Response 201:

```
{
  "id": "664f9d7d...",
  "slug": "abc123",
  "shortUrl": "https://your-domain.com/abc123",
  "longUrl": "https://example.com/very/long/path",
  "title": "My Example Link",
  "createdAt": "2025-12-03T12:34:56.000Z",
  "expiresAt": null,
  "disabled": false}
```

Error cases:

- Invalid URL
- Slug already in use
- Validation failed on fields

7.2 Get URL Details (Metadata, Not Redirect)

GET `/api/urls/:slug`

Returns URL info without analytics.

Response:

```
{
  "id": "664f9d7d...",
  "slug": "abc123",
  "shortUrl": "https://your-domain.com/abc123",
  "longUrl": "https://example.com/very/long/path",
  "title": "My Example Link",
  "createdAt": "2025-12-03T12:34:56.000Z",
  "expiresAt": null,
  "disabled": false}
```

7.3 List URLs

GET `/api/urls`

Query params:

- `page` (default 1)
- `limit` (default 20)

Response:

```
{
  "items": [
    {
      "id": "664f9d7d...",
      "slug": "abc123",
      "shortUrl": "https://your-domain.com/abc123",
      "longUrl": "https://example.com/very/long/path",
      "title": "My Example Link",
      "createdAt": "2025-12-03T12:34:56.000Z",
      "totalClicks": 120
    }
  ],
  "page": 1,
  "limit": 20,
  "total": 42
}
```

7.4 Disable / Enable URL

PATCH `/api/urls/:slug`

Request body can include:

```
{
  "disabled": true,
  "title": "Updated title",
  "expiresAt": "2026-01-01T00:00:00.000Z"
}
```

7.5 Analytics — Summary

GET `/api/urls/:slug/analytics/summary`

Returns high-level metrics.

Response:

```
{
  "slug": "abc123",
  "totalClicks": 120,
  "uniqueVisitors": 90,
  "createdAt": "2025-12-03T12:34:56.000Z",
  "lastClickAt": "2025-12-10T10:00:00.000Z"
}
```

7.6 Analytics — Time Series

GET `/api/urls/:slug/analytics/timeseries`

Query params:

- `from` (ISO date, optional)
- `to` (ISO date, optional)
- If omitted, default to last 30 days.

Response:

```
{
  "slug": "abc123",
  "granularity": "day",
  "points": [
    { "date": "2025-12-01", "clicks": 10 },
    { "date": "2025-12-02", "clicks": 25 }
  ]
}
```

7.7 Analytics — Referrers

GET `/api/urls/:slug/analytics/referrers`

Response:

```
{
  "slug": "abc123",
  "referrers": [
    { "referrer": "https://twitter.com", "clicks": 40 },
    { "referrer": "https://google.com", "clicks": 20 },
    { "referrer": "(direct)", "clicks": 60 }
  ]
}
```

> (direct) is used when no referrer header is present.

7.8 Analytics — Devices / Browsers / OS

GET `/api/urls/:slug/analytics/devices`

```
{
  "slug": "abc123",
  "devices": [
    { "deviceType": "desktop", "clicks": 70 },
    { "deviceType": "mobile", "clicks": 40 },
    { "deviceType": "tablet", "clicks": 10 }
  ]
}
```

GET `/api/urls/:slug/analytics/browsers`

```
{
  "slug": "abc123",
  "browsers": [
    { "browser": "Chrome", "clicks": 80 },
    { "browser": "Safari", "clicks": 20 },
    { "browser": "Firefox", "clicks": 10 },
    { "browser": "Other", "clicks": 10 }
  ]
}
```

GET `/api/urls/:slug/analytics/os`

```
{
  "slug": "abc123",
  "os": [
    { "os": "Windows", "clicks": 50 },
    { "os": "Android", "clicks": 40 },
    { "os": "iOS", "clicks": 20 },
    { "os": "Other", "clicks": 10 }
  ]
}
```

7.9 Redirect Endpoint (Public)

GET `/:slug`

- This is not JSON; it performs an HTTP redirect.
- Steps:
  1. Fetch `Url` by slug.
  2. If not found → 404 page.
  3. If disabled or expired → show a friendly error page.
  4. Record `Visit` in DB:
     - Read `req.headers['user-agent']`
     - Read `req.headers['referer']` (note spelling)
     - Read IP from `req.ip` or `x-forwarded-for`
     - Parse UA for browser, OS, deviceType
  5. Redirect to `longUrl`.

Error fallback: if DB write fails, still perform redirect, but log error.

---

8. Business Logic Details

8.1 Slug Generation

- Use a **random alphanumeric** generator by default.
- Slug length: default 6–8 characters.
- Characters: `a-zA-Z0-9`
- Algorithm:
  - Generate random slug
  - Check DB for collision
  - If exists, retry (limit e.g. 5 attempts, then error)

8.2 Expiration Handling

- A URL is considered **expired** if `expiresAt` is not null and `expiresAt < now`.
- Expired URL:
  - Should not redirect to long URL.
  - Should show an expired page.
  - Should not record new visits.

8.3 Unique Visitors (Approximation)

- Approximation approach:
  - For a given URL, unique visitor key can be `IP + UserAgent`.
  - When aggregating:
    - Distinct count of `(ipAddress, userAgent)` pairs in Visit collection.

> This does not need to be perfectly precise, just “good enough”.

8.4 Privacy Considerations

- Optionally store truncated IP:
    - e.g., for IPv4, store only first 3 octets: `192.168.1.xxx`.
- Mark this logic clearly in code for future enhancement.

---

9. Frontend / Dashboard Behaviour

The frontend should be simple but neat.

9.1 Pages

1. **Home / Link List** - `/`
    - Displays table of URLs from `/api/urls`
    - Columns: Short URL, Long URL (trimmed), Total Clicks, Created At
    - “Create New Link” button
    - Each row: “View Analytics” button → `/links/:slug`
2. **Create Link Page** - `/create`
    - Form fields:
        - Long URL (textbox)
        - Custom slug (textbox)
        - Title (textbox)
        - Expiration date/time (date+time input)
    - On submit:
        - POST `/api/urls`
        - On success, show short URL and copy button
3. **Analytics Page** - `/links/:slug`
    - Fetch:
        - `/api/urls/:slug`
        - `/api/urls/:slug/analytics/summary`

    - `/api/urls/:slug/analytics/timeseries`
    - `/api/urls/:slug/analytics/referrers`
    - `/api/urls/:slug/analytics/devices`
    - `/api/urls/:slug/analytics/browsers`
    - `/api/urls/:slug/analytics/os`
- Sections:
    - Card: basic info + total clicks + created date
    - Line chart: clicks per day
    - Pie chart: devices
    - Pie/bar chart: browsers
    - Table: top referrers

---

10. Configuration

Use environment variables:

- `PORT` — port for HTTP server
- `MONGODB_URI` — connection string
- `BASE_URL` — e.g., `https://your-domain.com`
- `NODE_ENV` — `development` or `production`
- `LOG_LEVEL` — optional

---

11. Project Structure (Suggested)

https://github.com/hiteshchoudhary/apihub

---

12. Testing Requirements

- Unit tests for:
    - Slug generator
    - Expiration logic
- Integration tests for:
    - Creating URL
    - Redirect behavior
    - Analytics endpoints (basic assertions)

Frameworks suggested:

- Jest or Vitest for testing
- Supertest for HTTP tests

---

13. Future Enhancements (Optional / Nice-to-Have)

- Authentication (per-user URLs and dashboards)
- Public vs private URLs
- QR code generation for each short URL
- GeoIP-based country analytics
- Dark mode for frontend
- Export analytics as CSV

You can now give this entire spec to an AI (including me) and say:

> “Generate the full Node.js project (backend + minimal frontend) according to this SPEC.md.”

If you want, next step I can write a **README.md** tailored for GitHub and a **short project description** you can paste into your repo.

You said:

Please provide a PDF of all of it.

Also, I don't need structure like this.

I want structure like this https://github.com/hiteshchoudhary/apihub

ChatGPT said:

Cool — I can totally help. Here's how we'll proceed given your needs 👇

Important limitation: I — as ChatGPT — cannot directly generate a *downloadable PDF file* for you inside this chat. But I can give you **everything in Markdown (or plain text)**. You can then easily convert that to PDF using tools like md-to-pdf (Node.js CLI), Pandoc, or even copy-paste into a doc and export as PDF.

Because you asked for structure like apihub — I'll adapt the spec accordingly: fewer prose-chapters, more "project-root README + config + code-style" — similar to how apihub organizes things. GitHub+1

🧰 Repository Layout & README (Markdown)

Below is a skeleton of your project's **README + spec**. Save this as `README.md` at repo root. You can then convert it to PDF if needed.

```
# Shortify – URL Shortener with Analytics

> Simple URL shortener service with click tracking & analytics dashboard

## 🚀 What is Shortify

Shortify allows you to:
- Shorten long URLs to compact slugs
- Redirect users using short URLs
- Track each click (time, referrer, browser, OS, device type)
- View analytics per link (click count, referrers, device/browser/OS distribution, time-series)
- Manage links: create, list, optionally disable/expire

## 🧩 Tech Stack
```

```
**Backend**
- Node.js + TypeScript
- Express.js
- MongoDB (Mongoose or similar)
- User-agent parsing library (e.g. `ua-parser-js`)

**Frontend (optional / simple)**
- React or Next.js (optional)
- CSS / Tailwind (optional)
- Chart library (e.g. Chart.js, or similar)

## 📦 Data Models

### Url
| Field       | Type     | Description |
|-------------|----------|-------------|
| slug        | string   | Unique short code (indexed, unique) |
| longUrl     | string   | Original full URL |
| title       | string?  | Optional title / description |
| createdAt   | Date     | When shortened |
| expiresAt   | Date?    | Optional expiration |
| disabled    | boolean  | Whether link is disabled |

Indexes: unique on `slug`.

### Visit
| Field       | Type     | Description |
|-------------|----------|-------------|
| urlId       | ObjectId | Reference to Url |
| slug        | string   | Denormalized slug (for queries) |
| createdAt   | Date     | Time of click |
| ipAddress   | string?  | (optional / truncated for privacy) |
| referrer    | string?  | HTTP Referrer header |
| userAgent   | string?  | original UA string |
| browser     | string?  | parsed browser name |
| os          | string?  | parsed operating system |
| deviceType  | string?  | “desktop” / “mobile” / “tablet” / “other” |

Indexes: on `urlId`, `slug`, `createdAt`.

## 🔌 API Endpoints (HTTP + JSON)

| Method | Path | Description |
|--------|------|-------------|
| POST   | `/api/urls` | Create a new short URL |
| GET    | `/api/urls` | List all URLs (with click counts) |
| GET    | `/api/urls/:slug` | Get metadata for a short URL |
| PATCH  | `/api/urls/:slug` | Update / disable / expire a URL |
| GET    | `/api/urls/:slug/analytics/summary` | Get summary analytics (total clicks, uniques) |
| GET    | `/api/urls/:slug/analytics/timeseries` | Clicks per day (last 30d by default) |
| GET    | `/api/urls/:slug/analytics/referrers` | Top referrers |
| GET    | `/api/urls/:slug/analytics/devices` | Device type distribution |
| GET    | `/api/urls/:slug/analytics/browsers` | Browser distribution |
| GET    | `/api/urls/:slug/analytics/os` | OS distribution |
| GET    | `/:slug` | Public redirect — records visit + redirects to long URL |

## 🔧 Business Logic & Rules

- Slug generation: random alphanumeric (6–8 chars) or custom slug (4–32 chars, alphanumeric + `-`/`_`)
- Validate long URL (must be http/https)
- On redirect: record visit, parse UA, referrer, IP (optional / truncated)
- Expired or disabled links: show friendly error — no redirect, no analytics
- Unique visitors (approx): distinct by IP + userAgent combo

## 🧪 Tests (Optional but recommended)

- Unit tests: slug generator, expiration logic
- Integration tests: API endpoints, redirect + analytics recording
```

````
## 🔮 Future Enhancements (v2+)

- User accounts + login (per-user link dashboard)
- Custom alias + alias reuse prevention
- QR code generation per link
- Export analytics to CSV
- GeoIP analytics (country/region distribution)
- Frontend UI with charts
````

~~~
---

## 📄 How to Generate PDF from README

After you save `README.md`, you can convert it to PDF easily. For example, using `md-to-pdf` (Node.js tool):

```bash
npx md-to-pdf README.md
~~~

This will produce `README.pdf`.