AffordMed URL Shortener — Professional System Design

Version: 1.0

1. Executive Summary

AffordMed is a lightweight URL shortening solution originally implemented as a frontend-only application (React + Vite + TailwindCSS). This document provides a professional system design for both the existing frontend-only implementation and an extended production-ready architecture that includes a backend API, database, analytics, and deployment recommendations. The goal is to provide a clear blueprint for developers and stakeholders to implement, scale, and operate the service.

2. Goals & Non-Functional Requirements

Functional Goals: shorten long URLs, redirect users, track click events, show statistics. **Availability:** 99.9% uptime for redirects (production architecture). **Latency:** redirect latency < 100ms for cached requests. **Scalability:** support >10M shortened links and 100k redirects/sec with horizontal scaling. **Security:** protect against open redirect misuse, XSS, and injection attacks. **Durability:** data persisted in durable storage with backups.

3. Current (Frontend-only) Architecture

The current implementation is a single-page application (SPA). All logic runs in the browser: - URL validation, short-code generation (nanoid), storage (localStorage), redirect handling via React Router. - Event logging is implemented in sessionStorage for local analytics. Limitations: links remain local to the browser; not shareable across devices; storage clearing removes data.

4. Proposed Production Architecture

To make AffordMed a production-ready, shareable service, introduce a backend API and managed database. Key components: - Load Balancer (e.g., AWS ALB) - API Servers (Node.js + Express) - Primary Database (MongoDB / DynamoDB) - Cache Layer (Redis) for hot redirect keys - Object store for static assets (S3) and image hosting - Analytics pipeline (Kafka -> Clickstore) for click processing - Monitoring & Logging (Prometheus, Grafana, ELK) - CI/CD pipeline (GitHub Actions -> Build -> Deploy)

5. Component Design

Component	Responsibility	Technology
Web Client (SPA)	Collect URLs, show stats, initiate shorten & redirect	React, Vite, TailwindCSS
API Server	Validate input, generate short codes, persist mappings, enforce rate limits	Node.js, Express, TypeScript
Database	Store mappings & metadata (createdAt, expiry, clicks)	MongoDB / DynamoDB
Cache	Serve high-frequency redirect lookups	Redis / ElastiCache
Message Queue	Buffer click events for async processing	Kafka / AWS SQS
Analytics Store	Store aggregated click metrics	Clickhouse / BigQuery

6. Data Model

Primary collection/table: **links** Fields (MongoDB-like): - _id (ObjectId) - shortCode (string, unique, indexed) - longUrl (string) - createdAt (ISODate) - expiresAt (ISODate, optional) - ownerId (string, optional) — for authenticated users - clicksCount (number) - meta { title, tags, description } - disabled (boolean)

Secondary collection: **clicks** - _id - shortCode (string, indexed) - clickedAt (ISODate) - referrer (string) - userAgent (string) - ipAddress (string) - geo { country, region, city }

7. API Design (REST)

1. POST /api/shorten - Request: { longUrl, customAlias (optional), expiryMinutes (optional) } - Response: { shortUrl, shortCode, createdAt, expiresAt } 2. GET /:shortCode - Behavior: lookup shortCode in Redis cache -> DB -> if found increment click (async) and redirect (302) to longUrl. - If not found -> 404 page 3. GET /api/link/:shortCode - Response: link metadata (longUrl, createdAt, expiresAt, clicksCount) 4. GET /api/stats/:shortCode - Response: aggregated click stats (last24h, last7d), top-referrers

8. Sequence Diagrams (High-level)

Shorten flow: User -> Web Client -> POST /api/shorten -> API Server validates -> DB save -> return shortCode -> display to user Redirect flow: User -> GET /:shortCode -> Load Balancer -> API Server checks Redis -> hit -> return longUrl (302) -> async enqueue click event for analytics

9. Scaling & Caching Strategy

- Use Redis as primary cache for redirect lookups. Populate cache on first read and via write-through for new short codes. - Partition database by sharding on shortCode or use consistent hashing for distributed keys. - Auto-scale API servers based on CPU/Request metrics. - Use CDN (CloudFront) for static assets and assets hosting.

10. Security & Abuse Mitigation

- Validate longUrl to prevent SSRF / open redirects to internal networks. - Rate limit endpoint (POST /api/shorten) per IP or per API key. - CAPTCHA for anonymous users after threshold reached. - Store IP-blocklist and detect suspicious patterns (mass creation, repeated redirects). - Sanitize outputs to avoid XSS in any preview fields. - Use HTTPS everywhere and set secure cookies for auth.

11. Monitoring & Observability

- Metrics: request rates, error rates, latency, redirect throughput. - Logs: structured logging for API requests and redirect events. - Traces: distributed tracing (Jaeger) for request flows in production. - Alerts: P95 latency > Xms, error rate > Y% trigger alerts.

12. Deployment Recommendations

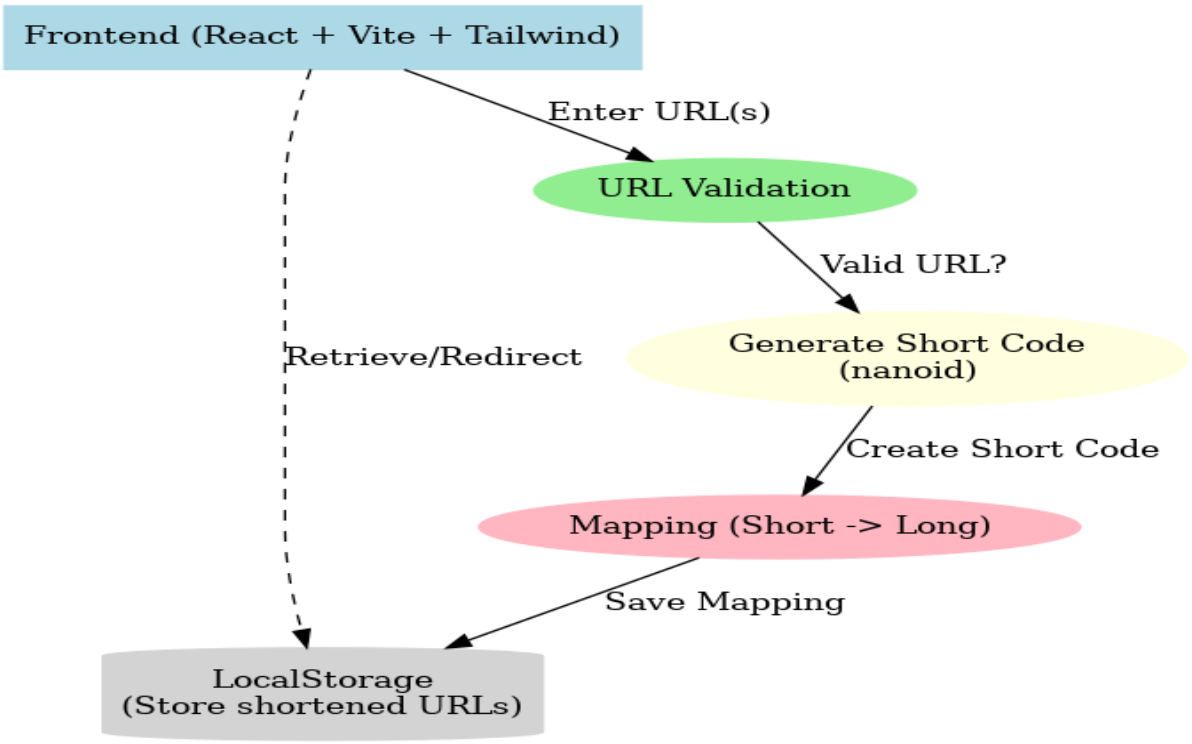
- Use containerized services (Docker). Deploy via Kubernetes or managed services (ECS/EKS). - CI/CD: GitHub Actions -> Build -> Run tests -> Deploy to staging -> Canary -> Production. - Use managed databases (MongoDB Atlas) and managed Redis. - Backups: regular backups for DB; snapshots for persistent volumes.

13. Cost Estimate (Monthly Ballpark)

- Small MVP: $50–$200 (single small VPS + managed DB) - Production (medium): $500–$2,000 (Load balancer, multiple app servers, managed DB, Redis, monitoring) - Large scale: $5k+/month (auto-scaling, analytics, multi-region)

14. Trade-offs & Alternatives

- Using serverless (AWS Lambda + DynamoDB) reduces ops but increases cold-start risk for redirects. - Using SQL DB can provide strong consistency, but NoSQL (Mongo/Dynamo) offers flexible schema for metadata. - Caching is essential for low-latency redirects; consider edge caches (Cloudflare Workers) for global performance.


Frontend (React + Vite + Tailwind)
Enter URL(s)
URL Validation
Valid URL?
Generate Short Code
(nanoid)
Create Short Code
Mapping (Short -> Long)
Save Mapping
Retrieve/Redirect
LocalStorage
(Store shortened URLs)

Appendix A — Data Schema (Example Documents)

Example **links** document: {"shortCode":"abc123", "longUrl":"https://example.com", "createdAt":"2025-08-26T13:00:00Z", "expiresAt":"2025-08-26T13:30:00Z", "clicksCount":0, "ownerId":null} Example **clicks** document: {"shortCode":"abc123", "clickedAt":"2025-08-26T13:05:00Z", "referrer":"https://google.com", "userAgent":"...","ipAddress":"1.2.3.4"}

Document prepared by: AffordMed Engineering