Here are **bullet-point notes** for designing a **URL Shortener** (like Bit.ly), structured by key system design categories:

**🧩 1. Problem Statement**

* Convert long URLs into short, unique aliases.
* Redirect short URLs to the original long URLs.
* Handle high read/write traffic (millions of URLs).
* Support analytics (click count, referrer, etc.).

**📎 2. Requirements**

**✅ Functional**

* Shorten a long URL.
* Redirect a short URL to the original long URL.
* Optional: Set expiration for short links.
* Optional: User accounts, custom aliases, link analytics.

**❌ Non-Functional**

* High availability.
* Low latency redirection.
* Scalability to billions of URLs.
* Fault-tolerance and consistency.

**🔗 3. API Design**

http

CopyEdit

POST /shorten

Request: { "longUrl": "https://example.com/long/path" }

Response: { "shortUrl": "https://short.ly/abc123" }

GET /abc123

→ Redirects to original long URL

**🧠 4. Core Components**

* API Gateway
* URL Generation Service
* URL Mapping Store (DB)
* Redirection Service
* Analytics Service (optional)

**🧬 5. Short URL Generation Techniques**

* **Base62 Encoding** of a unique ID (e.g., auto-increment or UUID)
* **Hashing** (e.g., MD5/SHA-256 + collision resolution)
* **Custom Alias Support**: Allow user-specified short codes

**💾 6. Data Storage**

**Schema**

* short\_code: VARCHAR(10) (PK)
* long\_url: TEXT
* created\_at: TIMESTAMP
* expiration: TIMESTAMP (optional)
* click\_count: INT (optional)

**Storage Options**

* **SQL DB** (e.g., MySQL/Postgres) for consistency.
* **NoSQL** (e.g., DynamoDB, Cassandra) for scalability.

**🚀 7. Redirection Flow**

1. User hits short URL.
2. Lookup short code in DB (use cache).
3. Redirect to long URL (HTTP 301/302).

**⚡ 8. Caching Layer**

* Use **Redis or Memcached** to cache mappings.
* Avoids DB hits for frequently accessed URLs.
* TTL can match URL expiration.

**🔨 9. Scalability Strategies**

* **DB Sharding** by hash of short code.
* **Horizontal scaling** of services.
* **CDN** for redirect pages (if static).
* **Rate limiter** on shortener API.

**🔁 10. Handling Collisions**

* If using hashing, check DB for existing short code.
* Retry or append salt/random bits to resolve.

**🔐 11. Security and Abuse Prevention**

* Validate long URLs (avoid SSRF, XSS).
* Rate-limit API requests per user/IP.
* Monitor for phishing/malicious URLs.

**📊 12. Analytics (Optional)**

* Track clicks, IPs, geolocation, device type.
* Store in time-series DB (e.g., InfluxDB).
* Display on user dashboard.

**💡 13. Additional Features**

* Expiring URLs
* Custom aliases
* Bulk shortening
* QR code generation