# **Tiny URL - System Design Interview Question URL Shortener**

## ****Overview of the Problem Statement****

A URL shortener service like TinyURL or Bitly provides a way to take long URLs and generate a much shorter version. The primary goal of this system is to allow users to:

* **Convert a long URL into a short URL**.
* **Retrieve the long URL when a short URL is accessed**.
* **Ensure scalability, efficiency, and high availability** to handle billions of URLs.

## ****Functional & Non-Functional Requirements****

### ****Functional Requirements****

1. **Shorten URL** → Given a long URL, return a short URL.
2. **Redirect to Long URL** → When a user accesses the short URL, they should be redirected to the original long URL.

### ****Non-Functional Requirements****

1. **Low Latency** → The system should handle URL redirections quickly.
2. **High Availability** → Ensure service reliability even under heavy load.
3. **Scalability** → Handle billions of URLs and expand as needed.
4. **Security** → Prevent URL hijacking or brute-force guessing of short URLs.

## ****API Design & Endpoints****

A **REST API** is chosen due to its simplicity and wide adoption.

### ****API Endpoints****

* **POST /create** → Accepts a long URL and returns a short URL.
  + Request: { "long\_url": "https://example.com/some-long-url" }
  + Response: { "short\_url": "https://tinyurl.com/abcd123" }
* **GET /{short\_url}** → Redirects users to the long URL.
  + Request: GET /abcd123
  + Response: **301 Permanent Redirect** to the original URL.

A screenshot of a computer

Description automatically generated

## ****Scalability Planning & URL Space Calculation****

### ****Estimating Storage & URL Length****

* The system needs to generate **1,000 URLs per second**.
* Annually, this results in **31 billion URLs**.
* Over 10 years, it requires storage for **310 billion URLs**.
* Short URLs use **7 characters**, selected from **62 characters (A-Z, a-z, 0-9)**, providing **3.5 trillion unique combinations**, ensuring enough unique IDs for future growth.

### ****Encoding Scheme for Short URLs****

* Uses **Base62 encoding (A-Z, a-z, 0-9)** to convert numbers to unique short URLs.
* Example:
  + **ID: 123456789** → Base62 Encoded **"abcd123"**
  + **Ensures short, unique, and URL-friendly identifiers.**

**A screenshot of a computer

Description automatically generated**

## ****System Architecture****

### ****Initial Naïve Solution****

1. **User sends a URL creation request**.
2. **The system assigns an incremented ID** and encodes it in Base62.
3. **Stores the mapping in a single database**.
4. **When retrieving the long URL, it performs a lookup** and redirects the user.

**Issue:**

* **Single Point of Failure** → If the central counter or database fails, the system crashes.
* **Limited Scalability** → Cannot efficiently handle billions of URLs.

### ****Improved Scalable Architecture****

1. **Load Balancer** → Distributes traffic among multiple web servers.
2. **Multiple Web Servers** → Handle URL generation and lookup.
3. **Distributed Database (Cassandra)** → Stores mappings efficiently.
4. **Caching Layer (Redis/Memcached)** → Stores popular URLs for faster lookups.
5. **Zookeeper for Range-Based ID Allocation** → Prevents duplicate IDs across multiple servers.

**Solution Benefits:**

* **Eliminates bottlenecks** and distributes load efficiently.
* **Reduces database dependency** with caching.
* **Ensures unique short URLs** using a distributed ID allocation system.

## ****Database Choice: SQL vs. NoSQL****

### ****SQL Database (PostgreSQL, MySQL)****

**Advantages**:

* ACID compliance ensures **strong data consistency**.
* Well-structured relational storage.

**Disadvantages**:

* Does not scale well for **billions of records**.
* Requires **sharding**, which adds complexity.

### ****NoSQL Database (Cassandra, DynamoDB)****

**Advantages**:

* **Highly scalable** for billions of records.
* Faster lookups due to **optimized distributed storage**.
* Supports **horizontal scaling** across multiple servers.

**Final Choice:** **Cassandra** due to its **efficiency for read-heavy workloads**.

A screenshot of a computer

Description automatically generated

## ****Performance Enhancements****

### ****Caching with Redis/Memcached****

* **Stores recently accessed short URLs in memory**.
* **Reduces database lookups**, improving response time.

### ****Asynchronous Processing****

* **Decouples heavy tasks**, allowing URL creation and logging to be handled efficiently.

### ****Database Sharding & Partitioning****

* Splits data into multiple partitions for **scalability**.
* Prevents single database overload.

## ****Security Enhancements****

* **Rate Limiting** → Prevents abuse and DDoS attacks.
* **Randomized Short URLs** → Prevents attackers from brute-forcing valid short URLs.
* **Encryption for Sensitive Data** → Secures stored URLs against unauthorized access.

## ****Optimized Workflow Example****

### ****URL Creation Process (POST /create)****

1. User **sends a request** with a long URL.
2. Load balancer **distributes request** to a web server.
3. Web server **requests a unique ID** from Zookeeper.
4. The ID is **encoded into Base62** for a short URL.
5. **Mapping is stored** in Cassandra and optionally in Redis.
6. Server **returns the short URL** to the user.

### ****URL Redirection Process (GET /{short\_url})****

1. User **accesses a short URL**.
2. Server **checks Redis cache** for a stored mapping.
3. If not found, **queries Cassandra for the long URL**.
4. Returns an **HTTP 301 redirect** to the original URL.

## ****Insights Based on Numbers****

📌 **1,000 URLs per second** → **31 billion URLs per year**.  
🔠 **Base62 encoding** provides **3.5 trillion unique URLs**.  
🚀 **Caching (Redis/Memcached)** reduces database lookups.  
🔄 **Zookeeper-based ID allocation** prevents duplicate short URLs.

YT Video: <https://www.youtube.com/watch?v=Cg3XIqs_-4c>   
My GPT chat: <https://chatgpt.com/g/g-GvcYCKPIH-video-summarizer/c/67bd5b0b-53f4-800f-9da5-a82611ca3dfb>