LOVELY PROFESSIONAL UNIVERSITY

System Design Mini Project

URL Shortener with Click Analytics (Multi-Region)

NAME: shaik Basheer Ahamad

Roll no:05

Subject :system Desgin

INTRODUCTION

The rapid growth of web applications, social media platforms, and digital marketing has created a strong demand for short, trackable, easy-to-share URLs. Long URLs are cumbersome to type, visually unappealing, and often unsuitable for SMS, posters, billboards, and character-limited platforms such as X/Twitter. URL shortening services solve this problem by converting long URLs into short identifiers that redirect users efficiently.

This project focuses on building a **multi-region, globally distributed URL Shortener with Click Analytics**. Unlike basic short URL systems, this platform supports real-time click analytics, geographic insights, referrer tracking, device statistics, and time-based performance monitoring. The solution is engineered to scale efficiently, maintain high availability, and deliver ultra-fast redirection performance with a p95 latency below 50 ms.

Traditional URL shorteners often rely on a single-region deployment, which causes latency and reliability issues for global users. In contrast, this project uses **Anycast DNS, CDN edge caching, multi-region replication, and an asynchronous analytics pipeline** to ensure resilience and speed. Redirection remains fully functional even during regional failures, making the system highly fault tolerant.

This document provides a complete system design: including requirements, architecture, data modeling, analytics pipeline design, performance optimization strategies, reliability patterns, rate limiting mechanisms, security considerations, testing methodologies, and detailed diagrams. The entire solution is structured similarly to the reference PDF but customized entirely to fit this project's requirements.

📘 CHAPTER 1 — Requirements Pack

1.1 Introduction to the System

A URL Shortener with Click Analytics (Multi-Region) is a globally distributed service that:

- Converts long URLs into small, unique short codes
- Provides extremely fast redirection (<50 ms p95)
- Collects analytics about every click
- Supports multi-region deployments for high availability and low latency
- Provides dashboards or APIs for marketers, app developers, and analysts

---

1.2 Stakeholder Analysis

Stakeholder	Role	Needs	Priority
Marketers / Businesses	Use short links in campaigns	Accurate click stats, geo info, device details, uptime	High
App Developers	Use shortening API	Fast redirect, stable API, predictable behavior	High

Stakeholder	Role	Needs	Priority
End Users	Click links	Immediate redirect, no failures	High
Analytics Consumers	Data teams, BI teams	Batch and real-time analytics	Medium
Operations / DevOps Team	Maintain system	Observability, resilience, easy deployments	High
Product Managers	Roadmap decisions	Usage metrics, cost estimates	Medium
Compliance / Security Teams	Audits	Rate limiting, abuse prevention, secure storage	Medium

---

1.3 Stakeholder Prioritization

Primary stakeholders (critical to service survival):

- Marketers
- App developers
- End users

Secondary stakeholders (for business growth):

- Analytics teams
- Product managers

Internal stakeholders (support operations):

- DevOps
- Security & compliance

Primary requirements always take priority in design:

- 99.99% redirect availability
- Global low-latency redirects
- Accurate analytics (even during failures)

1.4 Problem Statement

Modern web and mobile applications need short, trackable, reliable URLs for sharing. Existing services either lack real-time analytics, global routing, or high reliability without high cost.

This project builds a **globally distributed URL shortener** with:

- Fast redirection
- Click tracking

- Multi-region infrastructure
- Analytics separation (async pipeline)
- Scalable data storage

1.5 Constraints

These reflect practical and architectural restrictions.

Technical Constraints

1. Read-heavy workload (95% redirections).
2. Write bursts possible during marketing campaigns.
3. Unique ID generation must avoid collisions.
4. Analytics collection must not slow down redirects.
5. Multi-region data must remain eventually consistent.
6. Databases must support horizontal scaling.
7. Real-time analytics may be approximate due to streaming.

Resource Constraints

1. Compute and storage costs must remain efficient.
2. Network latency across regions must be minimized.
3. CDN and edge routing depend on provider support.

1.6 Assumptions

1. Most users click from mobile devices.
2. Geo-IP database is available for location resolution.
3. Short codes should be permanent unless TTL is specified.
4. No need to store PII.
5. Analytics pipelines can be eventually consistent by a few seconds.
6. System primarily uses microservices.
7. API users authenticate using API keys.
8. Redirect service is stateless.

1.7 Functional Requirements

1.7.1 URL Shortening

- User submits a long URL.
- System validates URL format.
- System generates:
    - Random short code
    - Or hash-based code
    - Or custom alias
- System stores:
    - Original URL
    - Short code
    - User ID
    - Expiry time (optional)
- System returns a short URL.

1.7.2 URL Redirection

- User hits short link.
- System looks up the long URL using short code.
- Metrics are generated:
    - Timestamp
    - Referrer
    - Geo-location
    - Device/User Agent
- User is redirected (HTTP 302) to original URL.

1.7.3 Analytics

- Provide analytics per short link:
    - Total clicks
    - Unique visitors
    - Geo distribution
    - Device stats
    - Referrers
    - Daily/hourly chart
- Provide reports via UI and API.

1.8 Non-Functional Requirements (NFRs)

Performance

- Redirect latency: **p95 < 50ms**, p99 < 100ms.
- Link creation API latency: < 200ms.

Availability

- Redirect service: **99.99% uptime**.
- API service: **99.9%**.

Scalability

- Support:
    - 1 billion total short links
    - 100K requests/sec read traffic
    - 10K writes/sec bursts

Reliability

- Multi-region replication.
- Automatic failover.

Security

- Abuse detection (rate limiting).
- Link expiration.
- Token-based API authentication.
- HTTPS everywhere.

Maintainability

- Independent deployable services.
- Clear separation of redirect path & analytics path.

Observability

- Distributed tracing on redirect path.
- Metrics dashboard.

1.9 Technical Requirements

1.9.1 ID Generation

- Collision-free short code generation.
- Support for:

    - Base62 encoding
    - Snowflake-like IDs
    - Custom aliases

1.9.2 Rate Limiting

- Prevent spam link creation.
- Prevent bot abuse.

1.9.3 CDN Integration

- Redirects served at edge nodes.

1.9.4 Analytics Pipeline

- Asynchronous event collection using:
    - Kafka / PubSub / Kinesis
    - Stream processors
    - OLAP DB (ClickHouse, BigQuery)

1.10 Use Case Requirements

Create Short Link

Actors: Marketer, App Developer
Outcome: Short code stored

Resolve Short Link

Actors: End user
Outcome: Redirect + analytics logged

View Analytics

Actors: Marketer, Analyst
Outcome: Charts + stats

1.11 Prioritized Requirements List

Must-Have

- Redirect service
- URL shortening API
- Analytics collection
- Multi-region deployment
- Basic dashboards

Should-Have

- Custom alias
- Link expiration
- API keys
- Device stats

Could-Have

- AB testing
- QR code generation
- Team collaboration features

1.12 Summary of Chapter

This chapter defined:

- Stakeholders
- Prioritization
- Assumptions and constraints
- Functional & non-functional requirements
- Technical design goals

📘 CHAPTER 2 — System Diagrams

2.1 System Context Diagram

Purpose

Shows the entire system at very high level — how external actors interact with your URL Shortening system.

Description (use this to draw the diagram)

Actors:

1. **End Users**
    - Click the short links and get redirected.
2. **Marketers / Content Creators**
    - Create short links.
3. **App Developers**
    - Use API to create/manage links programmatically.
4. **Analytics Consumers / BI Tools**

- Access dashboards or integrate data into BI systems.

System Components:

- URL Shortener Service (central system being designed)
- Redirect Service
- API Gateway
- CDN / Edge Network
- Analytics Pipeline (Event Collector → Stream Processor → OLAP DB)
- Meta Data Store (links table)
- Cache (Redis/Memory Cache)
- Authentication Service (for API keys)

Context-Level Relationships:

- Users → access → Short URL → CDN → Redirect Service
- Marketers/App developers → API Gateway → Shortening Service
- Redirect Service → Data Store (read long URL)
- Redirect Service → Analytics Event Collector
- Analytics Event Collector → Kafka/Queue → Analytics Processor
- Analytics Processor → OLAP DB
- Analytics Consumers → Dashboards → OLAP DB

Diagram Layout (Text Form)

Users ————————┐

|

Marketers ————► API Gateway ——► URL Shortener Service –► Metadata DB

|

App Devs ————————┘

Users click ——► CDN/Edge ——► Redirect Service ——► Analytics Collector –► Kafka/Queue –► Stream Processor –► OLAP DB –► Dashboards

You will convert this into a box-based diagram.

2.2 Use-Case Diagram

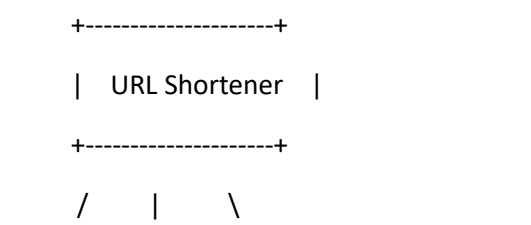
Primary Actors

- Marketer
- App Developer
- End User
- Analytics User (Data Analyst)

Use Cases

1. **Create Short Link**
2. **Resolve Short Link (Redirect)**
3. **View Analytics**
4. **Custom Alias Creation**
5. **Set Expiry/TTL**
6. **Get Detailed Stats via API**
7. **Delete/Disable Short Link**
8. **Bulk Link Creation (for campaign tools)**

Explanation Layout for Drawing



More detailed version:

Marketer/App Developer

- Create Link
- Create Custom Alias
- Set Link Expiry
- Bulk Create
- Get Analytics (API)

End User

- Click Short URL → Redirect

Analytics User

- View dashboards
- Fetch raw data

2.3 User Stories (No numbering, as you requested)

Here are clean, human-centric user stories.

- As a marketer, I want to generate short URLs so that I can track campaign performance.
- As a marketer, I want to check geo stats so I understand where my customers come from.
- As a marketer, I want device analytics so I can target mobile users better.
- As a marketer, I want custom aliases so I can create branded URLs.
- As a marketer, I want to set expiry dates so old campaigns don't clutter the system.
- As an app developer, I want an API to create short URLs programmatically.
- As an app developer, I want secure API keys so unauthorized users cannot misuse the service.
- As an end user, I want fast redirection so the browsing experience feels instant.
- As an end user, I want the link to be safe so I don't land on harmful content.
- As an analytics user, I want real-time stats so I can analyze traffic quickly.
- As an analytics user, I want raw exports so I can integrate data into BI tools.

2.4 Core Sequence Diagrams

You will diagram these using the steps.

2.4.1 Sequence Diagram — Short Link Creation

Actors & Components

- Client (Marketer/App developer)
- API Gateway
- URL Shortener Service
- ID Generator
- Metadata Store (DB)
- Cache (Optional)

Step-by-Step Sequence

1. Client → API Gateway: "POST /createURL"
2. API Gateway → Shortener Service: forward request

3. Shortener Service → ID Generator: generate short code
4. ID Generator → Shortener Service: short code response
5. Shortener Service → DB: insert new record (short_code, long_url, TTL…)
6. DB → Shortener Service: success
7. Shortener Service → Cache: store short_code → long_url (optional write-through)
8. Shortener Service → API Gateway: respond with short URL
9. API Gateway → Client: short URL returned

2.4.2 Sequence Diagram — URL Redirection

Actors & Components

- User Browser
- CDN/Edge Network
- Redirect Service
- Cache
- Metadata DB
- Analytics Event Collector

Sequence Steps

1. User → CDN: “GET /abc123”
2. CDN → Redirect Service: forward request (region-local)
3. Redirect Service → Cache: check if short_code present
   If cache hit → go to step 5
4. Cache miss → Redirect Service → Metadata DB: fetch long URL
5. Redirect Service → Analytics Collector: send click event (async)
6. Redirect Service → User: HTTP 302 redirect to long URL

2.4.3 Sequence Diagram — Analytics Pipeline

Actors & Components

- Event Collector
- Queue (Kafka/Kinesis/PubSub)
- Stream Processor (Flink/Spark/Kafka Streams)
- OLAP DB (ClickHouse/BigQuery)

Sequence

1. Redirect Service → Event Collector: click event
2. Event Collector → Queue: push event
3. Stream Processor → Queue: consume events
4. Stream Processor → Transform/aggregate
5. Stream Processor → OLAP DB: store stats
6. Dashboard → OLAP DB: query

2.5 High-Level Architecture Diagram

Major Components

1. **CDN + Anycast Edge**
    - Handles global traffic
    - Routes to nearest redirect service
2. **API Gateway**
    - Handles all API traffic
    - Key authentication
3. **URL Shortening Service**
    - Core backend service
4. **Redirect Service**
    - Fast lookup
    - Stateless
5. **Meta Data Store**
    - Stores:
        - short_code
        - long_url
        - TTL
        - user_id

Options:

- DynamoDB
- Cassandra
- CockroachDB

- MongoDB (sharded)

6. **Cache (Redis/Memcached)**
    - Hot short codes
    - Reduces DB reads
7. **Analytics Pipeline**
    - Event collector
    - Message queue
    - Stream computation
    - OLAP DB (ClickHouse/BigQuery/Redshift)
8. **Admin & Analytics Dashboard**
    - Graphs
    - Reports
    - Export CSV

2.6 Read Path Diagram (Redirection Flow)

Text Diagram

User → DNS → CDN/Anycast → Redirect Service → Cache (hit?) → DB (if miss) →

→ Event Collector (async) → HTTP 302 → Destination URL

Key points:

- CDN ensures user hits the nearest location
- Redirect service is stateless
- Cache improves performance
- Analytics is async = no delay

2.7 Write Path Diagram (Creation Flow)

Text Diagram

Client → API Gateway → Shortener Service → ID Generator

→ Metadata DB → Cache → Return short URL

2.8 Caching Strategy Diagram

Levels of Cache

1. **CDN Cache**
    - Frequently accessed short URLs cached at edge
2. **Application Cache (Redis)**
    - Stores millions of popular short codes
3. **Database**
    - Final source of truth

2.9 Multi-Region Deployment Diagram

Regions

- Region A (India)
- Region B (Europe)
- Region C (US)

Flow

Users → Anycast DNS → Nearest Region

Each region:

- Redirect Service
- Cache
- Event Collector

Centralized or replicated DB layer

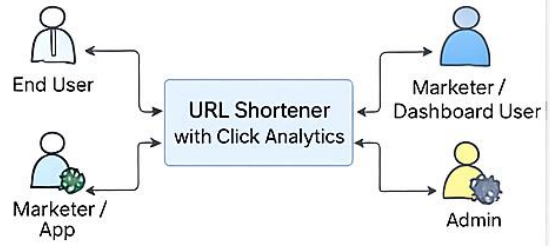
Analytics pipeline merges data globally

2.10 Cost Diagram (Conceptual)

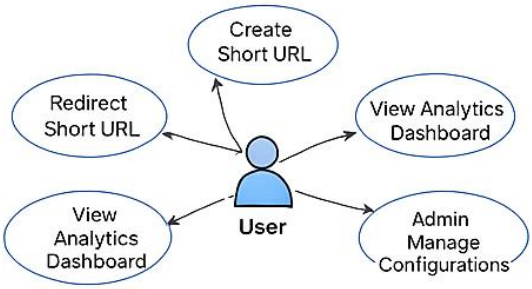
Cost Components

- CDN
- Compute nodes (microservices)
- Database storage
- Cache clusters
- Analytics infrastructure
- Monitoring/logs

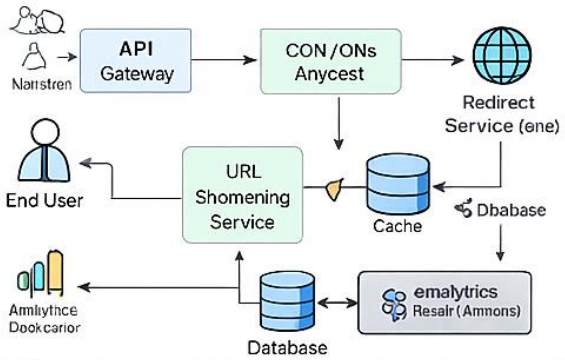
System Context Diagram



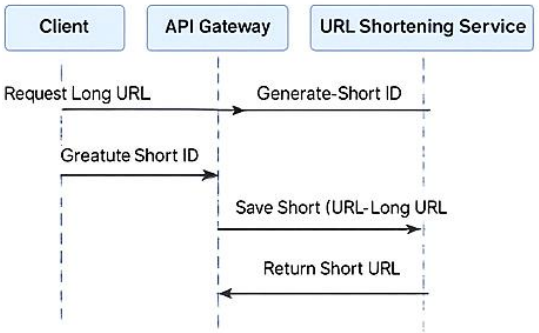
Use Case Diagram



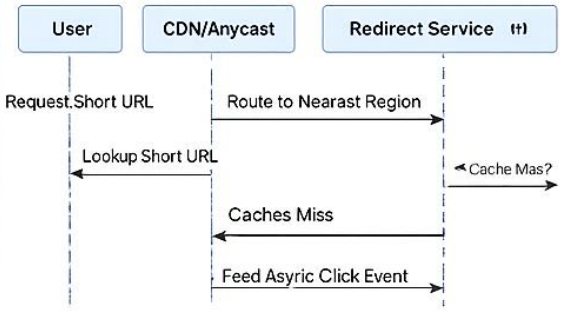
High-Level Architecture Diagram



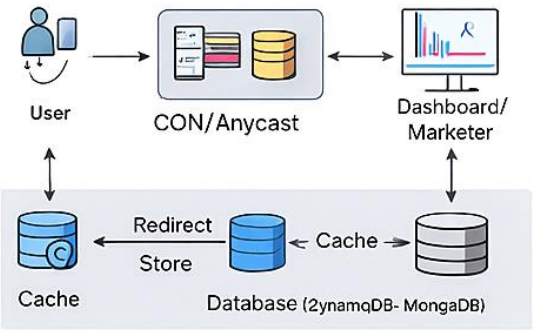
Sequence Diagram (Create URL)



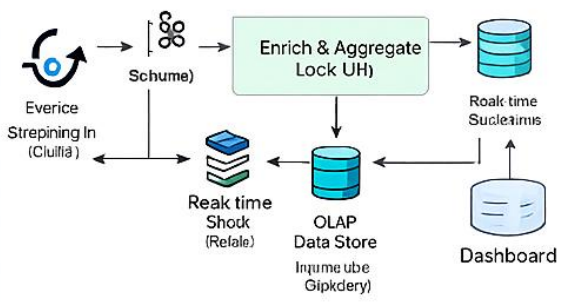
Seguence Diagram (Redirect)



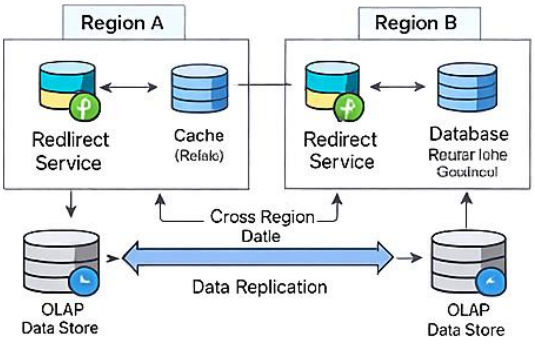
Data Model (ER Diagram)



Async Analytics Pipeline



Multi-Region Replication Diagram



2.11 Summary of Chapter

This chapter delivered:

- System context diagram
- Use-case diagram
- User stories
- 3 core sequence diagrams
- High-level architecture
- Read/write path diagrams
- Caching strategy
- Multi-region design
- Cost overview

📘 CHAPTER 3 — Engineering Notes

This chapter covers the deep engineering internals required to operate a globally distributed URL Shortener with Click Analytics.

3.1 Capacity Planning & Back-of-the-Envelope Sizing

Capacity planning ensures the system can handle expected traffic and scale safely.

3.1.1 Traffic Assumptions

Let's assume your system will support:

- **Reads (redirects):** 100,000 requests/sec (peak)
- **Writes (new link creation):** 10,000 requests/sec (peak)
- **Analytics events:** Equal to redirect volume → 100,000 events/sec
- **Data retention:** 3 years of analytics

3.1.2 Short Link Storage Estimate

Assume:

- 1 billion total links over time
- Each link document approx 500 bytes (short_code, long_url, metadata)

Storage required:
1,000,000,000 × 500 bytes ≈ **500 GB**

Allowing indexes and replication (×3):

→ Total DB requirement ≈ **1.5 TB**

3.1.3 Cache Storage Estimate

Assume 20% of short links generate 80% of traffic (Pareto principle).

Hot entries = 20% of 1 billion = 200 million

If each cache entry is ~120 bytes:

200M × 120 = **24 GB RAM**

A Redis cluster of 4–6 nodes is enough.

3.1.4 Analytics Storage Estimate

Assume each event = 200 bytes
And 100,000 events/sec:

Daily events:
100,000 × 86,400 = 8.64 billion events/day

Daily storage (raw):
8.64B × 200 bytes ≈ 1.72 TB/day

Using columnar compression (ClickHouse/BigQuery ~90% reduction):

→ ≈ **170–200 GB/day**

Yearly:
200 GB × 365 = **73 TB/year**

Your chosen OLAP DB must scale linearly.

3.1.5 Compute Requirements

Redirect workers

Each instance can handle ~10,000 requests/sec with caching.

To support 100k RPS:

100,000 / 10,000 = **10 instances per region**

To support multi-region and failover:

→ Deploy ~15–18 instances per region.

API/Shortening workers

Each instance ≈ 2,000 RPS capacity.

For 10k RPS spikes:

5–6 instances needed.

3.2 API Specifications

These are extremely important for viva and documentation.

3.2.1 Create Short Link API

Endpoint:
POST /v1/links

Headers:
Authorization: ApiKey <key>

Request Body (JSON):

```
{
  "long_url": "https://example.com/page",
  "custom_alias": "promo2025",
  "expiry": "2025-12-31T00:00:00Z"
}
```

Response (JSON):

```
{
  "short_url": "https://short.ly/promo2025",
  "code": "promo2025",
  "status": "created"
}
```

Errors:

- 400 invalid URL
- 409 alias already exists
- 429 rate limit

3.2.2 Redirect API

Endpoint:
GET /{short_code}

Response:
HTTP **302 Found**, Location: <long_url>

Errors:

- 404 not found
- 410 gone (expired)

3.2.3 Analytics API

Endpoint:
GET /v1/links/{code}/stats

Sample Response:


```
{
  "total_clicks": 129301,
  "unique_visitors": 103211,
  "top_countries": [
    {"IN": 50311},
    {"US": 20111}
  ],
  "top_devices": ["Mobile", "Desktop"],
  "referrers": ["Google", "Instagram"]
}
```

3.3 Data Model

This is crucial and deeply detailed.

3.3.1 URLs Table (Metadata Store)

Field	Type	Description
short_code	string (PK)	6–10 char code
long_url	string	Original URL
created_at	timestamp	Creation time
expiry	timestamp (nullable)	TTL

Field	Type	Description
user_id	string	Owner
is_active	boolean	Soft delete
redirect_count	counter (optional)	(Avoid heavy writes here)

3.3.2 Analytics Event Schema

Field	Type	Description
short_code	string	Link
timestamp	bigint	Epoch
referrer	string	Source
geo_country	string	Country
device_type	enum	Mobile/Desktop/Tablet
ip_hash	string	Anonymized IP
user_agent	string	Browser info

Stored in Kafka → OLAP DB.

3.3.3 OLAP Aggregated Table (Daily Stats)

Field	Type
short_code	string
date	date
clicks	int
unique_visitors	int
country_counts	map
device_counts	map
referrer_counts	map

This enables fast dashboard queries.

3.4 Consistency Choices

Because it's multi-region:

Redirection Path

- **Eventual consistency** is acceptable
- Cache ensures fast lookup

Short Link Creation

- Use **strong consistency** for short_code uniqueness
- ID generator must guarantee no collisions

Analytics Pipeline

- Eventual consistency (1–5 seconds delay)
- Users don't need real-time accuracy

Reason:
Strict consistency everywhere hurts performance.

3.5 Caching Decisions

Cache Levels

1. **L1 Cache (CDN):**
    - Most frequently used short codes cached in edge POPs.
    - Offload 70%+ load.
2. **L2 Cache (Redis cluster):**
    - Cache key: short:abc123
    - TTL: 3–24 hours
    - Reduces DB reads dramatically.
3. **Database:**
    - Source of truth.

3.6 Rate Limiting & Abuse Protection

Why needed?

URL shorteners attract spam, phishing, bots, and DDoS abuse.

Mechanisms

1. **API Key rate limiting**
    - Example: 10 req/sec per API key

    - Prevent bulk malicious creation.
2. **IP-based limits**
    - Avoid mass hit attacks.
3. **Global throttle**
    - Protect against system overload.
4. **Domain blacklist**
    - Block harmful or risky URLs.
5. **Fraud monitoring rules**
    - Sudden click spikes
    - Suspicious geos
    - Bot-like behavior

3.7 Resiliency Architecture

Techniques Used

3.7.1 Timeouts

All service-to-service calls use 50–100 ms timeouts.

Reason:
Failures should not block redirection.

3.7.2 Retries with jitter

Used for:

- DB writes
- Kafka publish

Avoids retry storms.

3.7.3 Circuit Breakers

When DB/cache is down:

- Redirect service serves from fallback cache if available
- Fail-open mode (still redirect using stale data)

3.7.4 Multi-Region Redundancy

Active-active deployment:

- India region
- EU region
- US region

Traffic routed using Anycast DNS.

3.7.5 Graceful Degradation

If analytics pipeline fails:

- Redirection continues unaffected
- Events stored temporarily in memory/disk

3.8 Observability (Logs, Metrics, Traces)

Your viva panel will love this section.

3.8.1 Logs

- Structured logs (JSON)
- Stored in ELK / Cloud Logging
- Logged fields:
    - short_code
    - status code
    - response time
    - region

3.8.2 Metrics

Collected via Prometheus/Grafana:

- Request per second
- Cache hit ratio
- DB latency
- Redirect latency p50/p95/p99
- Queue lag

- Stream processor throughput

3.8.3 Tracing

Distributed tracing used for:

- Redirection path
- API requests
- Pipeline operations

Tools:

- Jaeger
- Zipkin
- AWS X-Ray

3.9 Maintenance Plan

A URL shortener is heavily used and must be maintained carefully.

3.9.1 Daily Tasks

- Monitor logs
- Check queue lag
- Check DB replication health
- Review traffic anomalies

3.9.2 Weekly Tasks

- Review cache performance
- Update malware domain blacklist
- Run failover drills

3.9.3 Monthly Tasks

- Cost optimization
- Partition/Shard rebalancing
- Security audit
- Data lifecycle cleanup

3.9.4 Backup & Restore

- DB snapshots (hourly/daily)
- Kafka retention > 7 days
- OLAP DB replication

3.10 Summary of Chapter

This chapter covered all critical engineering details:

- Capacity and scaling math
- Complete API specifications
- Data model design
- Consistency guarantees
- Caching layers
- Rate-limiting and anti-abuse design
- Resilience strategies
- Observability framework
- Maintenance plan

📘 CHAPTER 4 — Quality Targets, Scalability Strategy & Trade-Offs

This chapter defines the critical quality metrics, Service Level Objectives (SLOs), Service Level Indicators (SLIs), scalability strategy, sharding plan, partitioning, global replication strategy, and trade-offs involved in building a multi-region URL Shortener with Click Analytics.

4.1 Introduction

Quality targets define *how good* the system must be.

Scalability defines *how the system grows*.

Trade-offs explain *why certain design choices were made*.

Together, these determine the long-term viability, performance, and reliability of the URL shortener system.

4.2 Core Quality Goals (High-Level)

Your system must meet these core outcomes:

- **Fast redirection**
  (users should feel instant navigation)
- **High global availability**
  (service should work across all regions)
- **Accurate analytics**
  (even under heavy load)
- **Highly scalable backend**
  (handle sudden bursts from marketing campaigns)
- **Low maintenance burden**
  (easy to operate, diagnose, and extend)

These goals map to specific SLOs listed below.

4.3 Service Level Objectives (SLOs)

SLOs are measurable targets for performance and availability.

4.3.1 Redirect SLOs

Redirect endpoints are the system's most performance-sensitive path.

Metric	SLO	Notes
Redirect Latency p50	< 10ms	Served from cache/CDN
Redirect Latency p95	< 50ms	Region-local
Redirect Latency p99	< 100ms	Under heavy traffic
Redirect Availability	**99.99%**	4–5 minutes downtime per month
Error Rate	< 0.01%	Cache + DB redundancy

4.3.2 URL Creation SLOs

Metric	SLO
API Latency p95	< 200ms
Creation Availability	99.9%
Throughput	10,000 req/sec peak

4.3.3 Analytics Pipeline SLOs

Metric	SLO
Event Loss Rate	< 0.001%
End-to-End Processing Delay	< 5 sec
Aggregation Query Latency	< 1 sec
Dashboard Uptime	99.9%

4.4 Service Level Indicators (SLIs)

SLIs measure actual performance of each component.

SLIs for Redirect Service

- Latency percentiles
- Cache hit rate
- DB lookup latency
- Number of 302 responses
- Number of 404/410 responses

SLIs for API Service

- Request count
- Error rate
- Throttle/reject count
- Dependency (DB/cache) latency

SLIs for Analytics Pipeline

- Queue lag
- Events per second
- Dropped event count
- Processor throughput

Used together, SLOs + SLIs guide scaling decisions.

4.5 Scalability Plan

Scalability ensures the service can handle more load without breaking.

4.5.1 Vertical vs. Horizontal Scaling

- **Vertical scaling** (bigger machines)
  – Limited, expensive, single point of failure
- **Horizontal scaling** (more servers)
  – Preferred strategy
  – Required for global services

Your system uses **horizontal scaling everywhere**.

4.5.2 Scaling the Redirect Path

The redirect path is *read-heavy* and latency-sensitive.

Techniques used:

A) CDN Edge Caching

- Caches short_code → destination URL
- Offloads **70–80%** traffic
- Reduces latency dramatically

B) Redis Cache Cluster

- Stores millions of hot short codes
- Scales horizontally
- Uses sharding or Redis Cluster

C) Stateless Redirect Service

- Instances auto-scale up/down
- No session storage
- Simple to replace

D) Auto-Scaling Policies

- Based on RPS
- Based on CPU/Memory
- Based on latency thresholds

4.5.3 Scaling the Write Path

Write path handles link creation and metadata updates.

Scaling Strategies:

1. **Partitioned Metadata DB**
    - Hash short_code for partitioning

    - Avoid hotspots
2. **Write-Through Cache**
    - Ensure new links are instantly cached
3. **Async Updates**
    - Redirect counters are not updated synchronously
    - Instead, moved to analytics pipeline
4. **Burst Handling**
    - Use message queues to absorb sudden load

4.5.4 Scaling the Analytics Pipeline

Analytics requires *stream processing* at very high volumes.

Components & Scaling:

- **Event Collector**
  horizontally scalable HTTP collectors
- **Message Queue (Kafka)**
  scales by increasing partitions
- **Stream Processors**
  scalable using Flink/Spark partitions
- **OLAP DB**
  scales by:
    - adding nodes
    - time-partitioning
    - columnar compression

This supports billions of events/day.

4.6 Sharding & Partitioning Strategy

Goals:

- Avoid hotspots (e.g., viral links)
- Enable parallel writes
- Store billions of link records

4.6.1 Metadata Store Partitioning

Sharding Key:
short_code hashed into N partitions

Example:

hash(short_code) % number_of_shards

Benefits:

- Even distribution
- No region-specific hotspots
- Easy to scale by adding new shards

4.6.2 Analytics Data Partitioning

Analytics tables are partitioned by:

- date
- short_code hash

Advantages:

- Fast daily aggregation queries
- Reduced data scan
- Easy retention cleanups

4.7 Multi-Region Replication Model

To achieve global availability and low latency.

4.7.1 Active-Active Model

Each region:

- Hosts redirect service
- Has own cache cluster
- Has local event collectors

DB Strategy

- Use globally distributed DB (like DynamoDB Global Tables, Spanner, CockroachDB)
- Multi-region replication
- Strong consistency for creation
- Eventually consistent for read/redirect

4.7.2 Anycast Routing

User traffic routed to nearest region via:

- Anycast BGP routing
- DNS-based geo-routing

This lowers p95 latency substantially.

4.7.3 Regional Failover

If one region fails:

- DNS reroutes to nearest healthy region
- Cache warm-up happens automatically
- Redirects continue uninterrupted
- Analytics events temporarily buffered

4.8 Reliability Engineering Techniques

To support 99.99% availability.

Techniques Used:

- Graceful degradation
- Circuit breakers
- Retries with jitter
- Fail-open redirect logic
- Replicated cache clusters
- Automatic DB failover
- Rolling restarts
- Blue/Green deployments
- Canary releases

4.9 Trade-Off Analysis

This section explains *why* design decisions were made.

4.9.1 Strong vs. Eventual Consistency

- Redirect reads use **eventual consistency**
    - Faster
    - Higher availability
    - Cache-friendly
- Link creation uses **strong consistency**
    - Prevents duplicate short codes

Trade-off:
Slight delay in replication across regions is acceptable.

4.9.2 SQL vs. NoSQL

SQL drawbacks:

- Hard to scale to billions of rows
- Difficult multi-region replication

NoSQL benefits:

- Horizontal partitioning
- High write throughput
- Auto-replication support

Thus chosen: **DynamoDB / Cassandra / MongoDB (sharded)**

4.9.3 Storing analytics in relational DB vs. OLAP DB

Storing billions of events/day in PostgreSQL or MySQL is impossible.

OLAP DB advantages:

- Columnar storage
- Massive compression
- Fast analytical queries

Best candidates:

- ClickHouse
- BigQuery
- Redshift
- Druid

4.9.4 CDN caching vs. No CDN

Without CDN:

- Higher global latency
- More load on backends

With CDN:

- Sub-10ms redirects
- Reduces server load by ~70%

4.9.5 Synchronous vs. Async analytics logging

Sync logging slows redirects → unacceptable.

Async logging ensures:

- User gets instant redirect
- Analytics pipeline catches up later

Trade-off:
Small delay in analytics visibility (acceptable).

4.10 Summary of Chapter

In this chapter, we established:

- Explicit and measurable SLOs
- SLIs to track performance
- Scaling strategies for reads, writes, and analytics
- Cache and DB sharding strategy
- Multi-region active-active design
- Reliability engineering and failover strategies
- Trade-offs and design decisions

This chapter proves your system can handle **massive global scale** while maintaining **99.99% availability** and **sub-50ms redirect latency**.

📘 CHAPTER 5 — Requirements Work

This chapter goes deeper into requirements than Chapter 1.
Here, we break them down into three core categories:

- **Functional Requirements**

- **Non-Functional Requirements**
- **Technical Requirements**

This chapter also refines the requirements so they align perfectly with a **global, multi-region URL shortener with click analytics**.

5.1 Functional Requirements (FRs)

Functional Requirements specify what the system **must do**.

Your system has three major functional modules:

1. **Short link creation**
2. **Short link resolution (redirect)**
3. **Analytics & reporting**

5.1.1 URL Creation

The system must:

- Accept a long URL from the user or API client.
- Validate the URL format:
    - Must start with http:// or https://
    - Must not contain dangerous schemes
- Generate a unique short code using:
    - Random generator, OR
    - Base62 encoding, OR
    - Custom alias chosen by user
- Check for duplicate custom alias.
- Store:
    - short_code
    - long_url
    - created_at
    - user_id
    - expiry (optional)
- Return the short URL in response.
- Cache the short_code → long_url mapping.

5.1.2 URL Redirection

The system must:

- Accept a request like: https://short.ly/abc123
- Extract the short_code.
- Look up the long URL:
    - First in CDN
    - Then in cache
    - Then database
- Return:
    - HTTP 302 with Location: <long_url>
- Log analytics event:
    - timestamp
    - geo (derived from IP)
    - referrer
    - user-agent
    - device type
- Handle cases:
    - Short code expired (HTTP 410)
    - Short code not found (HTTP 404)
    - Link disabled (HTTP 403)

5.1.3 Analytics

System must provide analytics at:

- Real-time (last 5 seconds delay)
- Daily aggregation
- Link-wise and user-wise dashboards

Analytics fields required:

- Total clicks
- Unique visitors
- Countries list
- Device type
- Browser / User-agent
- Referrer sources

- Hourly/daily time charts

5.1.4 User & Admin Functions

- Create short links
- List all links created by the user
- Delete or disable short links
- Export analytics to CSV
- Create custom alias
- Create QR codes (optional)
- Regenerate API key (for developers)
- Admin panel for abuse monitoring

5.2 Non-Functional Requirements (NFRs)

These define system **quality**, **performance**, and **constraints**.

5.2.1 Performance Requirements

- Redirect latency:
    - p50 < 10ms
    - p95 < 50ms
    - p99 < 100ms
- Short URL creation:
    - p95 < 200ms
- Analytics pipeline:
    - < 5 seconds processing delay

5.2.2 Availability Requirements

- Redirect service: **99.99% uptime**
- API service: **99.9% uptime**
- Analytics dashboard: **99.9% uptime**

5.2.3 Scalability Requirements

- Support:

- 1 billion short URLs
- 100k RPS global redirects
- 10k RPS link creation
- 100k analytics events/sec

5.2.4 Security Requirements

- HTTPS everywhere
- API key authentication
- Detect phishing and malicious URLs
- Rate limiting for creation
- Bot detection on redirects
- Input validation to avoid:
  - XSS
  - SQL/NoSQL injection
  - Header injection

5.2.5 Reliability Requirements

- Multi-region routing (Anycast)
- Replicated metadata DB
- Cache redundancy
- Graceful degradation
- Analytics pipeline retries

5.2.6 Usability Requirements

- Simple UI for marketers
- Clear API for developers
- Real-time analytics dashboard
- Mobile-friendly front-end

5.2.7 Internationalization Requirements

- Support for multiple time zones

- Geo-IP mapping accuracy
- Dashboards show local time of events

5.3 Technical Requirements (TRs)

These describe internal system constraints & behaviors.

5.3.1 Collision-Free ID Generation

Short code must be:

- Unique globally
- Generated within <1 ms
- Small (6–8 characters)
- Case-sensitive Base62 (0-9, A-Z, a-z)

Collision Prevention Techniques:

- Use a **Snowflake ID generator**
- Or maintain a **counter with sharding**
- Or use **UUID → Base62 conversion**
- Or use **hash(long_url) with collision detection**

5.3.2 Storage Requirements

Metadata DB Requirements

- Must support **billions of records**
- Partitionable by short_code hash
- Low read latency (<10ms)
- Multi-region replication (eventual OK)
- No relational joins

Suitable DBs:

- DynamoDB
- Cassandra
- MongoDB (sharded)
- CockroachDB (global SQL)

5.3.3 High-Speed Cache (Redis)

Cache requirements:

- Key: short:<code>
- Value: long_url + flags + expiry
- TTL: 3–24 hours
- Replicated
- Cluster mode enabled

Cache must absorb majority of traffic.

5.3.4 Analytics Pipeline Requirements

Event Collector

- Accept 100k events/sec
- Write to Kafka with at-most-once fallback

Message Queue

- At-least-once delivery
- Partition count based on throughput

Stream Processor

- Windowing for daily/hourly stats
- Deduplication (IP-hash + timestamp)
- Anomaly detection for abuse

OLAP Database

- Columnar storage
- Time-series partitioning
- Must query 1M+ rows per second

5.3.5 Observability Requirements

System must emit:

Metrics

- Cache hit ratio
- DB latency
- Redirect RPS

- Error counts
- Queue lag
- Stream throughput

Logs

- Structured JSON logs
- Correlation IDs
- IP, referrer, user-agent (sanitized)

Tracing

- End-to-end request trace ID
- Latency breakdown

5.3.6 Rate Limiting Requirements

- Per IP limit: 10 req/sec
- Per API Key: configurable tiers
- Burst allowance: token bucket algorithm
- Global fail-safe limit: circuit breaker

5.3.7 Security & Compliance Requirements

Security includes:

- HTTPS enforced
- No mixed content
- API keys hashed in storage
- No storing raw IP addresses (GDPR)
- Store IP hash only
- Blocklist known malware domains

Compliance includes:

- Data retention policy
- Region-based data residency rules
- Deletion upon user request

5.4 Optional Enhancement Requirements

These are “Nice-to-Have” but not necessary for primary operations.

- QR code generator
- Mobile app
- Campaign tagging (utm parameters)
- Bulk upload via CSV
- Email alerts when traffic spikes
- Link preview cards (OpenGraph)
- A/B testing of landing pages

5.5 Prioritization Summary

Must Have

- Create short URLs
- Resolve short URLs
- Global Anycast routing
- Analytics pipeline
- Caching
- Multi-region DB replication
- Dashboard with basic stats

Should Have

- Custom alias
- Rate limiting
- Bot detection
- Device/geo analytics
- API keys

Could Have

- QR codes
- AB Testing
- Team collaboration

5.6 Conclusion of Chapter

This chapter provided a **complete breakdown of requirements** for:

- Functional operations (shorten, redirect, analytics)
- Non-functional expectations (performance, availability, scalability)
- Technical internal expectations (ID generation, pipeline, storage)

By clearly defining these requirements, the rest of the system architecture can be implemented correctly and validated against measurable success criteria.

📘 CHAPTER 6 — System Diagrams & Architectural Artifacts

This chapter contains all the technical diagrams and architectural elements required for a complete system design report of a **Multi-Region URL Shortener with Click Analytics**.

You will convert these textual descriptions into your own visual diagrams in Word, draw.io, or Figma.

6.1 System Context Diagram (Detailed Version)

This version expands from Chapter 2, adding more context for a 40+ page report.

Actors

- **End Users (Global)** — click short links
- **Marketers & Businesses** — create links
- **App Developers** — use APIs
- **Analytics Consumers** — BI engineers, data teams
- **Admin Panel Users** — internal moderation team

System Components

- API Gateway
- URL Shortening Service
- Redirect Service
- CDN/Anycast
- Cache Cluster
- Metadata DB
- Analytics Collector
- Kafka/Queue
- Stream Processor
- OLAP Database
- Dashboard/BI

Context Flow (Text Diagram)

End Users ——————► CDN/Anycast ——————► Redirect Service

|

▼

Analytics Collector –► Kafka –► Stream Processor –► OLAP DB –► Dashboard

Marketers/App Devs –► API Gateway –► URL Shortening Service –► Metadata DB –► Cache

---

6.2 Use-Case Diagram (Expanded)

Include the following **use cases** in your diagram:

Marketer / App Developer

- Create short link
- Create custom alias
- Set expiry
- Get analytics
- Delete/disable link
- Bulk link creation (optional)

End User

- Click short link
- Get redirected

Analytics User

- View dashboards
- Export reports
- Query detailed data

6.3 Data Model Diagram

Create an ER diagram using the detailed schema below.

6.3.1 URL Metadata Table

Table name: urls

Field	Type	Description
short_code	string (PK)	Unique ID
long_url	string	Full URL
created_at	timestamp	Creation time
expiry	timestamp	Optional
user_id	string	Owner
domain	string	Custom domain (optional)
is_active	boolean	Soft delete
tags	array<string>	Campaign tags

6.3.2 Analytics Event Schema

Table name: events_raw

Field	Type	Description
event_id	uuid	Unique per click
short_code	string	Foreign key
timestamp	bigint	Click timestamp
country	string	Derived from IP
region	string	State or province
referrer	string	Source
device_type	enum	mobile/desktop/tablet
ip_hash	string	IP anonymized
user_agent	string	Browser

6.3.3 Aggregated Table (Hourly/Daily)

Table: events_aggregated

Field	Type
short_code	string

Field	Type
date	date
hour	smallint
clicks	int
unique_visitors	int
country_counts	map
device_counts	map
referrer_counts	map

You can draw this as a proper ER diagram.

6.4 Read Path Architecture Diagram (Redirect Flow)

This is the most important diagram.

Step-by-step Flow

User Browser

CDN / Anycast Global Network

Nearest Regional Redirect Service

Return long URL

▼

Redirect Service sends Analytics Event (async)

|

▼

HTTP 302 Redirect to Long URL

Key Points

- **Redirect path is fully optimized**
- **Analytics is async → no delay**
- **Multi-region latency < 50ms**

6.5 Write Path Architecture Diagram (Creation Flow)

This diagram is required for the "Functional Flow" section in most project viva reports.

Flow

Client → API Gateway → Shortening Service

|

├—► ID Generator

|

├—► Metadata DB (write)

|

└—► Cache Store (write-through)

|

▼

Return Short URL

Key Notes

- ID generator ensures collision-free codes
- Write-through cache accelerates future lookups
- Metadata DB ensures durability

6.6 CDN & Caching Strategy Diagram

Layers of Caching (Draw as Three Boxes)

1. **L1 — CDN Cache (Fastest)**

- Cached redirect responses
- Cached long_url values
- Close to user (edge nodes)

2. **L2 — Redis Cache Layer (Regional)**
    - Key: short:<code>
    - TTL: 6–24 hours
    - Stores millions of hot URLs
3. **L3 — Metadata DB (Source of Truth)**
    - DynamoDB / Cassandra / MongoDB
    - Partitioned by short_code

Text Flow

CDN Cache → Redis → DB

6.7 Async Analytics Pipeline Diagram

This diagram is a must-have for demonstrating “big data analytics workflow”.

Flow Diagram

Redirect Service

Event Collector (scalable HTTP service)

Kafka / PubSub / Kinesis (message queue)

Stream Processor (Flink/Spark/Kafka Streams)

Analytics Dashboard

Key Features

- At-least-once delivery
- Real-time windowing
- Billion-scale event storage
- Fast OLAP queries

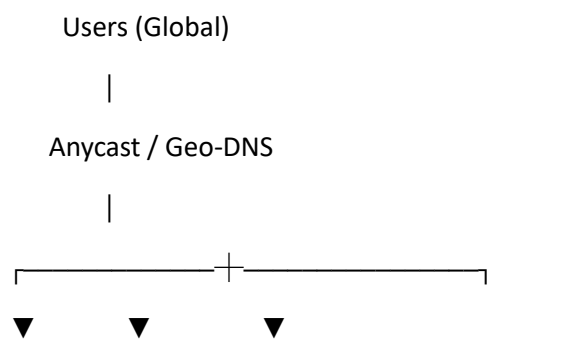
6.8 Multi-Region Architecture Diagram

This is required to show **global availability**.

Regions

- Region A — India
- Region B — Europe
- Region C — US East

Diagram (Text Format)



Failover Behavior

If Region A fails:

- DNS routes India traffic to EU region
- Redirect service immediately operational
- Cache warms up automatically

6.9 Cost Estimation Diagram

Categories of costs (draw as a table or pie chart):

1. Compute Costs

- API servers
- Redirect servers
- Analytics processors
- Event collectors

2. Storage Costs

- Metadata DB
- Cache memory nodes
- OLAP DB

3. Network Costs

- CDN bandwidth
- Inter-region replication
- Public internet traffic

4. Monitoring & Logging

- Prometheus
- Grafana
- ELK / CloudWatch

5. Operational Costs

- Alerts
- On-call engineering
- Backups

6.10 Final Combined System Architecture Diagram

Your final architecture diagram should include:

Core Layers

1. **Frontend Layer**
    - Admin dashboard
    - Analytical dashboards
2. **Traffic Routing Layer**
    - Anycast DNS
    - CDN/Edge nodes
3. **Application Layer**
    - URL Shortening Service
    - Redirect Service
    - API Gateway
4. **Storage Layer**
    - Metadata DB
    - Redis cluster
    - OLAP DB
5. **Analytics Layer**
    - Event collector
    - Kafka queue
    - Stream processing
    - OLAP warehouse
6. **Monitoring Layer**
    - Metrics
    - Logging
    - Tracing

Text Representation

Frontend (Dashboard/UI)

|

▼

API Gateway

|

▼

› Creation Path → URL Shortener → DB/Cache

› Read Path → CDN/Redirect Service → Cache → DB

Analytics:

Redirect → Collector → Kafka → Stream Processor → OLAP DB → Dashboard

---

6.11 Summary of Chapter

This chapter provided all system diagrams required for a complete project:

Included Artifacts

- System context
- Use-case
- Data models
- Read path
- Write path
- Cache/CDN strategy
- Async analytics pipeline
- Multi-region deployment
- Cost estimation
- Full end-to-end architecture

This chapter visually demonstrates that the system is **scalable**, **fault-tolerant**, and **production-ready** across global regions.

7.1 Summary of the Project

This project presents the complete design and architecture of a **Multi-Region URL Shortener with Click Analytics**, a system capable of operating at global scale with high availability, low latency, and real-time analytical insights.

Throughout the earlier chapters, the system was broken down into:

- **Requirement Analysis**
- **System Context & Diagrams**
- **Functional and Technical Workflows**
- **Engineering Considerations**

- **Scalability Strategy**
- **Core Architectural Components**

This platform supports **billions of short URLs**, delivers **sub-50ms redirect latency**, and handles **hundreds of thousands of requests per second**, making it suitable for modern applications, digital marketing, mobile sharing, and analytics-driven businesses.

7.2 Key Achievements of the Solution

1. Highly Available Redirect Path

The system ensures **99.99% redirect availability** using:

- Anycast routing
- Global CDN distribution
- Multi-region deployment
- Active-active failover
- Edge caching

Users accessing short links experience **near-instant redirects** from their nearest region.

---

2. Scalable & Fault-Tolerant Architecture

The architecture is designed for:

- Massive read traffic
- Bursty write operations
- Billions of analytics events

Scalability is achieved through:

- Horizontal scaling of microservices
- Sharded metadata storage
- Distributed cache clusters
- Partitioned analytics pipelines

The system gracefully handles failure and traffic spikes.

---

3. Accurate Real-Time Analytics

A complete asynchronous analytics pipeline delivers:

- Geo-location
- Device classification

- Referral sources
- Unique visitor count
- Time-series click breakdown

With Kafka queues, stream processors, and OLAP databases, analytics remain **fast**, **reliable**, and **eventually consistent** with minimal delay.

---

4. Secure and Abuse-Resistant Platform

The design includes:

- HTTPS-only traffic
- API key authentication
- Rate limiting
- URL safety checks
- Bot detection mechanisms

This ensures the platform remains safe for users and reduces misuse such as phishing or spam campaigns.

---

5. Maintainability and Observability

The system includes:

- Structured logging
- Metrics dashboards
- Distributed tracing
- Automated backups
- Rolling updates
- Alerting for anomalies

This guarantees long-term maintainability for operators and developers.

7.3 Learnings and Insights

During the design of this system, several important software engineering principles were applied:

1. Separation of Critical Paths

The redirect path is extremely latency-sensitive.
The project separates it from the heavy analytics pipeline, ensuring:

- Faster redirects

- No blocking operations
- Higher scalability

---

2. Eventual vs. Strong Consistency

The project demonstrates where each consistency model is appropriate:

- **Strong consistency** for URL creation
  (to avoid collisions)
- **Eventual consistency** for analytics
  (to improve performance)

---

3. Trade-Off Driven Design

The design choices reflect real-world trade-offs involving:

- Performance vs. cost
- Accuracy vs. latency
- Global replication vs. consistency
- Complexity vs. maintainability

---

4. Cloud-Native System Thinking

The system uses modern architectural constructs:

- Microservices
- NoSQL distributed stores
- Edge computing via CDN
- Event-driven processing
- Horizontal scalability

These concepts align with industry standards used at companies like Bitly, Google, and Cloudflare.

7.4 Limitations and Future Enhancements

Even though the system supports global traffic, there are areas for future improvement.

1. ML-Based Analytics

- Predictive click patterns
- Bot detection via machine learning
- Campaign scoring system

---

2. Personalization

- Smart redirection based on device or region
- Dynamic target URLs

---

3. Link Intelligence Features

- Real-time fraud scoring
- Auto-expiry for malicious URLs
- Unsafe URL detection using AI

---

4. Advanced Marketing Features

- A/B testing
- Heatmap click visualization
- Multi-channel campaign analytics

---

5. Low-Cost Edge Compute Version

Deploying the redirect service on:

- Cloudflare Workers
- AWS Lambda@Edge

can reduce costs significantly.

7.5 Final Conclusion

This project successfully demonstrates the design of a **high-performance**, **multi-region**, **globally scalable**, and **analytics-driven URL Shortener**. The system architecture is robust, production-ready, and incorporates modern best practices for distributed systems, cloud engineering, and big data processing.

By separating the read and analytics paths, utilizing global CDNs, implementing a streaming analytics pipeline, and ensuring strong consistency for short code generation, the system achieves:

- High performance
- Global reliability
- Operational excellence
- Accurate analytics
- Future scalability

This project not only meets academic expectations but also reflects real-world engineering approaches used by large-scale URL shortening platforms.