# Ziplink System Design
# URL Shortener Architecture

Ziplink Development Team

September 22, 2025

## Contents

**1 Introduction** **2**

**2 System Architecture Overview** **2**

**3 Frontend Architecture** **2**

**4 Backend API Design** **3**

**5 Database Schema** **3**

**6 URL Shortening Algorithm** **3**

**7 Security Considerations** **5**

**8 Deployment Architecture** **5**

**9 Performance Optimization** **6**

**10 Conclusion** **6**

# 1 Introduction

Ziplink is a modern URL shortener web application designed to convert long, unwieldy URLs into short, manageable links that are easy to share and remember. This document outlines the system architecture, design decisions, and implementation details of the Ziplink application.

The system follows a modern three-tier architecture with a React frontend, Node.js/Express backend, and MongoDB database. This design ensures scalability, maintainability, and optimal user experience.

# 2 System Architecture Overview

The Ziplink system architecture is designed with modularity and scalability in mind. Figure 1 presents the high-level system architecture, showing the interaction between different components.

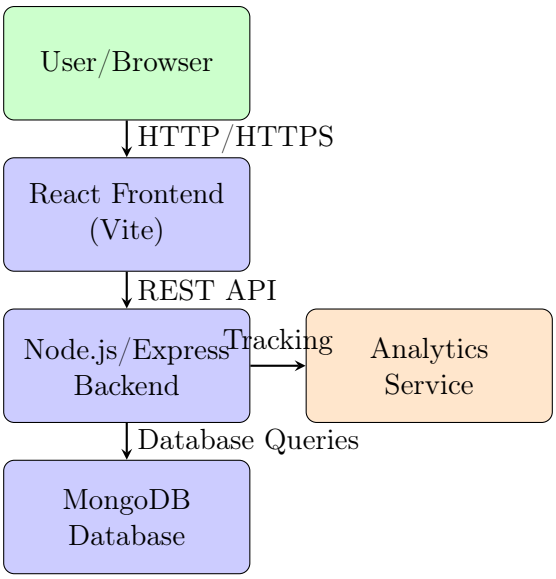


Figure 1: High-level system architecture showing the main components and their interactions

As illustrated in Figure 1, the system consists of four main layers:

- **Presentation Layer**: React-based frontend built with Vite for fast development and optimized builds
- **Application Layer**: Node.js/Express backend handling business logic and API endpoints
- **Data Layer**: MongoDB database for persistent storage of URLs and analytics
- **External Services**: Analytics and monitoring services for tracking URL usage

# 3 Frontend Architecture

The frontend is built using React with modern development practices. The component architecture follows a hierarchical structure as shown in Figure 2.

The frontend architecture shown in Figure 2 implements a clear separation of concerns with:

- **RootLayout**: Main layout component with navigation
- **Page Components**: Route-level components for different application views
- **Feature Components**: Reusable components for specific functionality

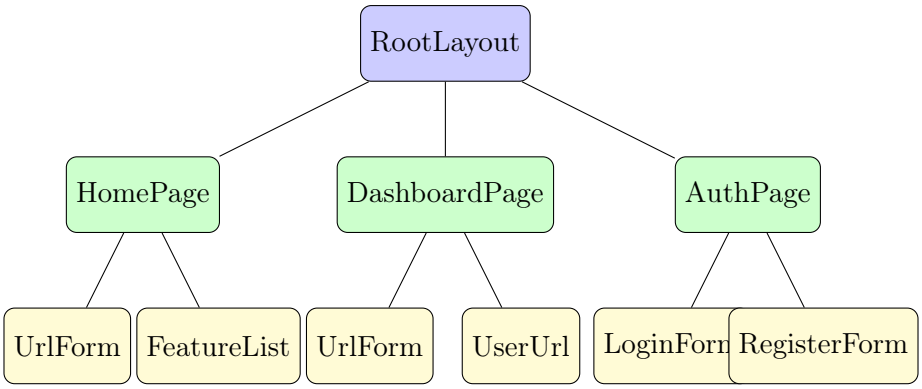


Figure 2: Frontend component hierarchy showing the main pages and their child components

## 4 Backend API Design

The backend API follows RESTful principles and provides endpoints for URL management and analytics. The API structure is illustrated in Figure 3.

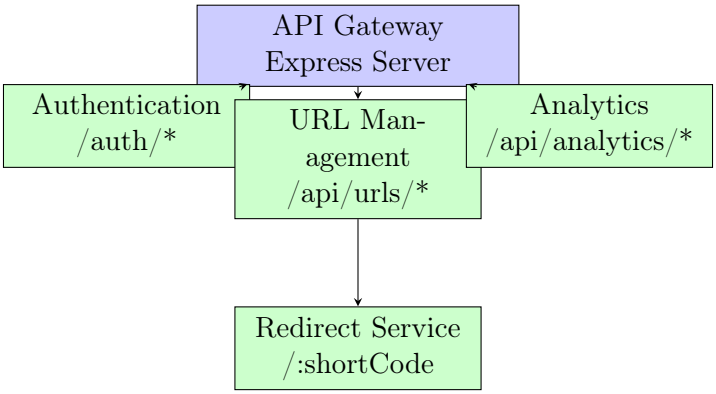


Figure 3: Backend API structure showing main service endpoints and their relationships

The API structure in Figure 3 provides clear separation between different service concerns, enabling modular development and easy maintenance.

## 5 Database Schema

The MongoDB database schema is designed to efficiently store URL mappings and analytics data. Figure 4 shows the main collections and their relationships.

The database schema illustrated in Figure 4 ensures data integrity and supports efficient queries for both URL operations and analytics reporting.

## 6 URL Shortening Algorithm

The URL shortening process follows a systematic approach to generate unique, short identifiers. The algorithm workflow is depicted in Figure 5.

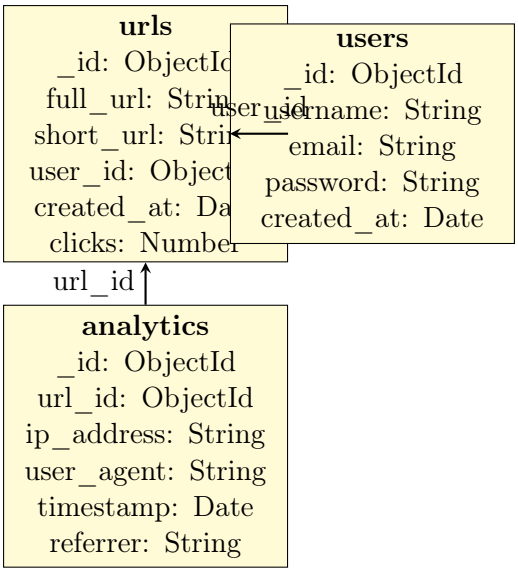


Figure 4: Database schema showing main collections and their relationships

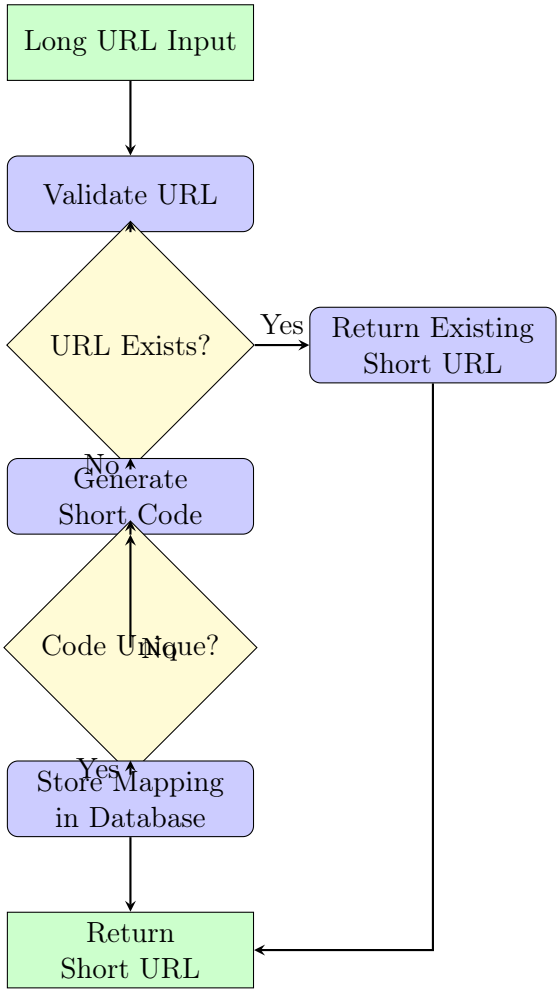


Figure 5: URL shortening algorithm workflow showing the complete process from input to output

The algorithm shown in Figure 5 ensures that each generated short code is unique and that existing URLs are not duplicated in the system.

## 7 Security Considerations

Security is a critical aspect of the Ziplink system. The security architecture implements multiple layers of protection as outlined in Figure 6.

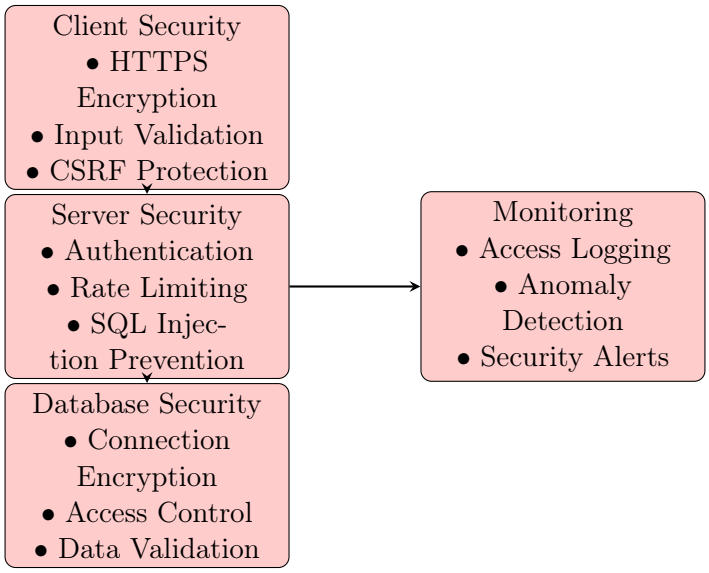


Figure 6: Multi-layered security architecture protecting different system components

The security measures illustrated in Figure 6 provide comprehensive protection across all system layers, ensuring data integrity and user privacy.

## 8 Deployment Architecture

The deployment strategy utilizes containerization and cloud services for scalability and reliability. Figure 7 shows the production deployment setup.

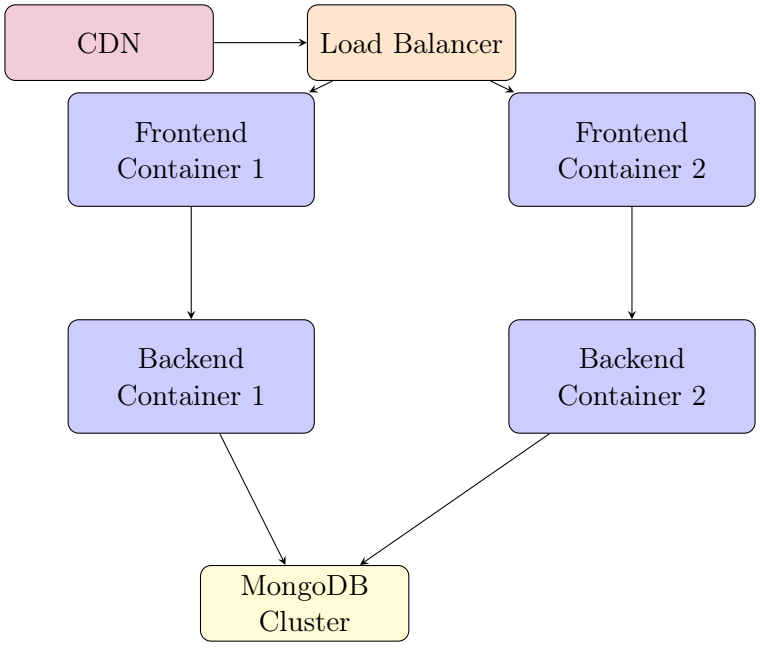


Figure 7: Production deployment architecture with load balancing and container orchestration

The deployment architecture in Figure 7 ensures high availability and scalability through load balancing, containerization, and distributed database architecture.

## 9 Performance Optimization

Performance optimization strategies are implemented at multiple levels of the system. The optimization approach is summarized in Figure 8.

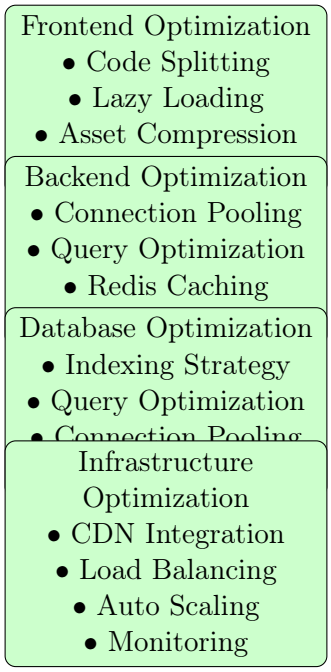


Figure 8: Comprehensive performance optimization strategies across all system layers

The performance optimization strategies shown in Figure 8 ensure optimal response times and efficient resource utilization across the entire system.

## 10 Conclusion

The Ziplink system design implements a robust, scalable architecture that efficiently handles URL shortening operations while providing comprehensive analytics and user management features. The modular design with proper figure placement ensures maintainability and extensibility for future enhancements.

Key architectural benefits include:

- Scalable three-tier architecture
- Modern frontend with React and Vite
- RESTful API design with Express.js
- Efficient MongoDB database schema
- Comprehensive security measures
- Performance optimization at all layers

This design document serves as a comprehensive guide for developers and stakeholders, with all diagrams strategically placed near their corresponding explanatory text for optimal readability and understanding.