# **URL Shortener Microservice – Design Document**

## **1. Overview**

The **URL Shortener Microservice** allows:

* Creation of shortened URLs with user-defined shortcodes and expiry.
* Retrieval of statistics for each shortened URL, including click data.
* Centralized logging of operations and errors via an external logging service.

This system is built to be lightweight, modular, and ready for extension with analytics, authentication, and production deployment features.

## **2. Architecture Overview**

### **Component Breakdown:**

* **Client/API User**: Sends HTTP requests to create or fetch URL data.
* **Express Application**: Handles routing, logging, validation, and response generation.
* **MongoDB**: Stores shortened URLs and click analytics.
* **Logger Utility**: Sends logs to a remote evaluation/logging server using axios.

### **Logical Flow:**

1. **Client** sends a request to the API.
2. The request passes through **middleware** (for logging).
3. The **controller** handles the business logic.
4. Data is persisted in or read from **MongoDB**.
5. Structured logs are sent to an **external logging server**.

## **3. Technology Choices**

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology** | **Justification** |
| Web Framework | **Express.js** | Lightweight, fast, and ideal for building RESTful APIs |
| Database | **MongoDB + Mongoose** | Flexible schema, supports embedded documents (like click logs) |
| Logger | **Axios** (to remote log server) | Decouples app logic from logging infrastructure |
| Configuration | **dotenv** | Environment management without hardcoding sensitive values |
| Middleware | **Custom Logger** | Provides centralized log tracking for all requests |

## **4. Key Design Decisions**

* **Separation of Concerns**: Each part of the app (models, routes, controllers, utils) is in a separate folder.
* **Embedded Click Logs**: For each short URL, click history is stored within the document. Simpler retrieval at scale.
* **Logging Integration**: The logger utility validates and sends logs to the specified remote API with token-based auth.
* **Custom Shortcodes**: Shortcodes are user-defined. This makes testing easier and prevents collisions for now.
* **Extensible Structure**: Easy to plug in features like rate limiting, auth, or analytics.

## **5. Data Modeling**

### **ShortUrl Schema**

{  
 originalUrl: String,  
 shortcode: String,  
 createdAt: Date,  
 expiresAt: Date,  
 clicks: [  
 {  
 timestamp: Date,  
 source: String,  
 location: String  
 }  
 ]  
}

* **Shortcode** is unique and indexed.
* **Clicks** are stored as an array inside the document.
* Scalable for small to medium-sized applications. Can be normalized later if needed.

## **6. API Design**

### **1. POST /shorturls**

**Purpose**: Create a new shortened URL

**Request Body**:

{  
 "url": "<https://example.com>",  
 "validity": "86400",   
 "shortcode": "xyz123"  
}

**Response**:

{  
 "shortlink": "<http://localhost:4000/xyz123>",  
 "expiry": "2025-06-29T18:22:00.000Z"  
}

### **2. GET /shorturls/:shortcode**

**Purpose**: Retrieve statistics for a specific shortcode

**Response**:

{  
 "totalClicks": 4,  
 "originalUrl": "<https://example.com>",  
 "createdAt": "...",  
 "expiresAt": "...",  
 "clickDetails": [  
 {  
 "timestamp": "...",  
 "source": "direct",  
 "location": "India"  
 }  
 ]  
}

## **7. Logging & Observability**

* Logging is handled by a reusable Log(stack, level, package, message) function.
* Logs are sent to an external service via POST request.
* Each API interaction (success/failure) is logged centrally.
* All logs must conform to defined enums for stack, level, and package.

## **8. Assumptions**

* The logging server is available and responds to authorized requests.
* The system is initially open-access (no authentication).
* Validity is given in **seconds**.
* Location data is **mocked**; not derived from IP resolution.
* The client provides a unique shortcode.

## **9. Scalability & Maintainability**

|  |  |
| --- | --- |
| **Concern** | **Strategy** |
| High traffic | Stateless API – deploy behind load balancer |
| Growing clicks | Store click logs separately when needed (future refactor) |
| Logging latency | Logging done asynchronously and won't block core API flow |
| Extensibility | Modular structure with separated concerns – new features can be plugged in easily |

## **10. Testing Approach**

* Used **Postman** or **cURL** for endpoint testing.
* **MongoDB Memory Server** can be used for integration tests without affecting production DB.

## **11. Conclusion**

This microservice offers a foundational yet extensible design for managing shortened URLs and observing usage patterns. It’s built for maintainability, scalability, and easy integration into a broader microservices ecosystem.