# URL Shortener Microservice - System Design Document

## Executive Summary

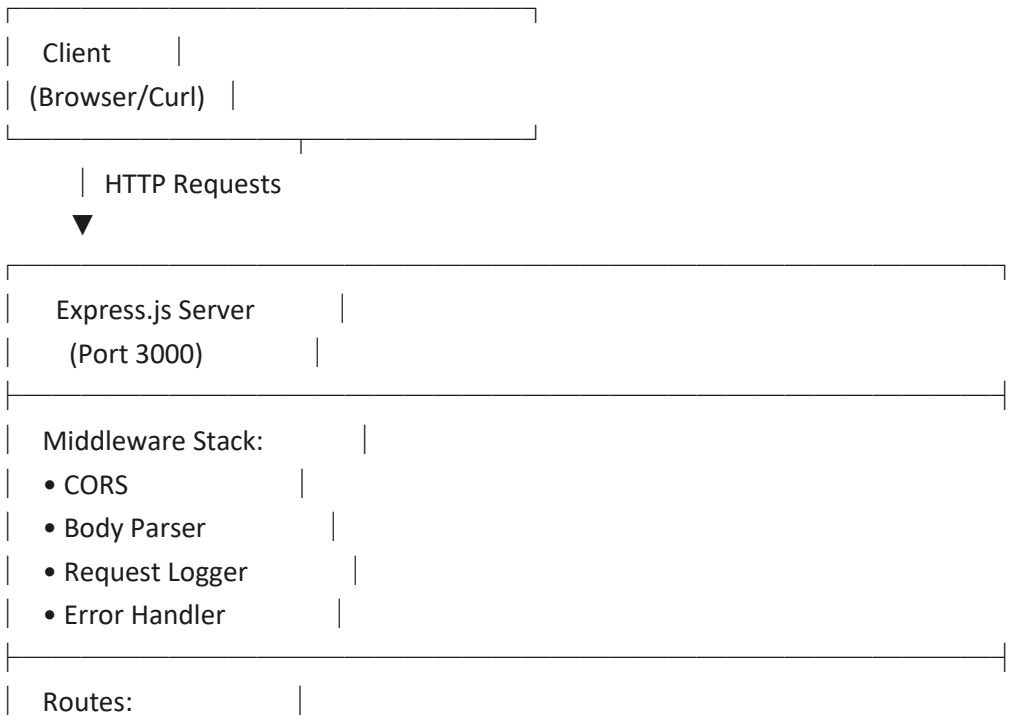
This document describes the **actual implementation** of a URL Shortener Microservice built as a prototype/demo system. The implementation is a **single Node.js application** with **in-memory storage**, designed to demonstrate core functionality rather than production scalability.

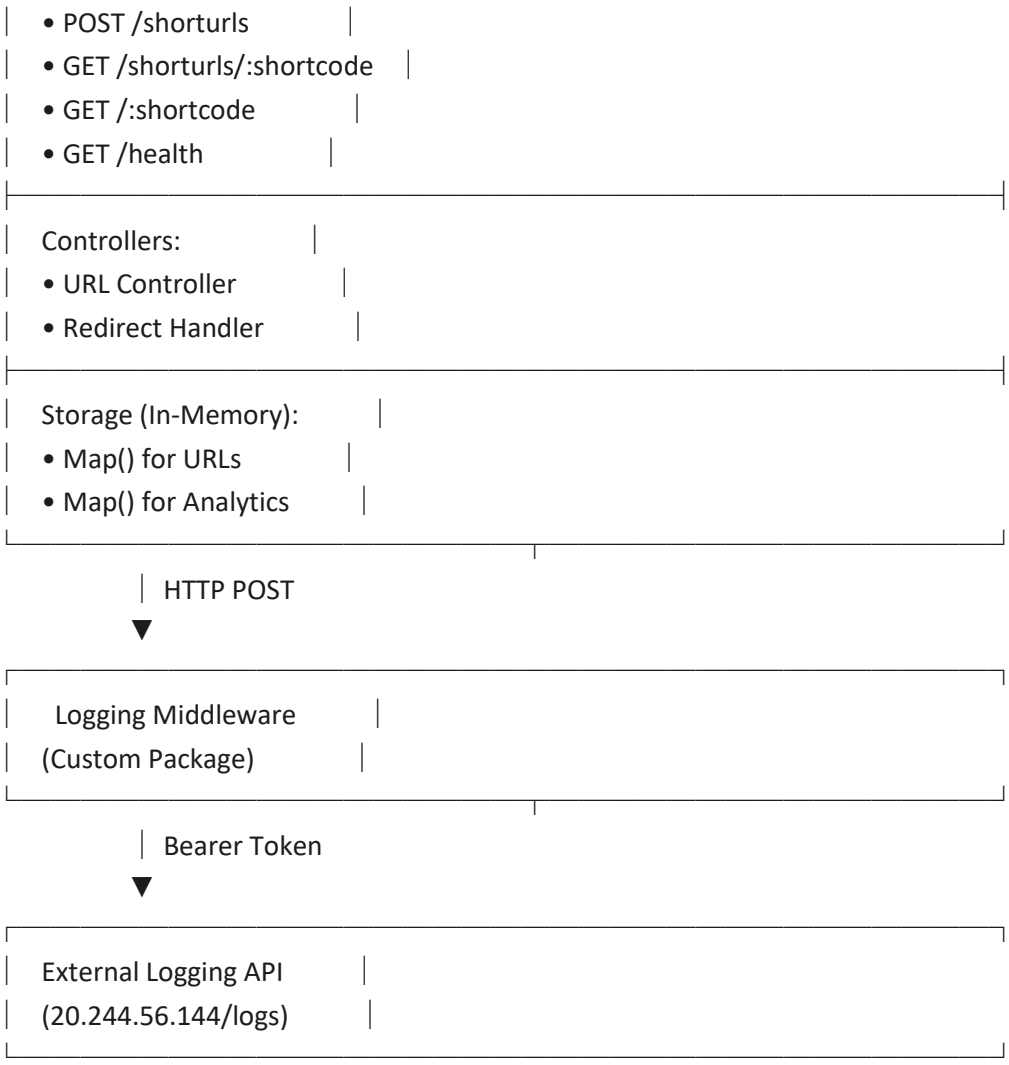
### What We Actually Built

- Single Node.js Express server running on port 3000
- In-memory storage using JavaScript Maps
- Custom logging middleware that sends logs to external API
- RESTful endpoints for URL shortening and analytics
- Basic error handling and validation

## Actual System Architecture

### Real Implementation Architecture (What We Built)






## Component Structure (Actual Files)

```
backend/
├── src/
│   ├── server.js          # Main Express application
│   ├── storage.js         # In-memory storage using Maps
│   ├── utils.js           # Utility functions
│   ├── controllers/
│   │   └── urlController.js    # URL creation and stats
│   ├── handlers/
│   │   └── redirectHandler.js  # URL redirection logic
│   ├── routes/
│   │   └── urlRoutes.js        # Route definitions
```

```
│   └──── middleware/
│       └──── index.js          # Custom middleware
├──── .env                  # Environment variables
├──── package.json          # Dependencies
└──── tests/
    └──── api.test.js       # Basic API tests
```

# Technology Stack (Implemented)

## Actually Used Technologies

| Component | Technology | Version | Why Chosen |
|---|---|---|---|
| **Runtime** | Node.js | 18+ | Required for JavaScript backend |
| **Framework** | Express.js | ^4.18.0 | Simple, fast web framework |
| **Language** | JavaScript | ES6+ | Project requirement |
| **HTTP Client** | Axios | ^1.6.0 | For external API calls |
| **Storage** | JavaScript Maps | Native | Simple in-memory storage |
| **Environment** | dotenv | ^16.0.0 | Environment variable management |
| **CORS** | cors | ^2.8.5 | Cross-origin support |
| **Testing** | Jest + Supertest | ^29.0.0 | Testing framework |

## Dependencies (package.json)

```
{
  "dependencies": {
```

```
    "express": "^4.18.0",
    "cors": "^2.8.5",
    "body-parser": "^1.20.0",
    "axios": "^1.6.0",
    "dotenv": "^16.0.0"
  },
  "devDependencies": {
    "nodemon": "^3.0.0",
    "jest": "^29.0.0",
    "supertest": "^6.3.0"
  }
}
```

# Data Models (Current Implementation)

## In-Memory Storage Structure

### URL Storage (Map)

```
// Key: shortcode (string)
// Value: urlData (object)
{
  originalUrl: "[https://example.com/very-long-url](https://example.com/very-long-url)",
  shortcode: "abc123",
  createdAt: "2025-09-08T10:30:00.000Z",
  expiresAt: "2025-09-08T11:00:00.000Z",
  validity: 30  // minutes
}
```

### Analytics Storage (Map)

```
// Key: shortcode (string)
// Value: Array of click events
[
  {
    timestamp: "2025-09-08T10:35:00.000Z",
    referrer: "[https://google.com](https://google.com)",
    userAgent: "Mozilla/5.0...",
    ipAddress: "192.168.1.1",
    location: "Local Network"
  }
```

]

### Storage Implementation (storage.js)

```
class URLStorage {
   constructor() {
      this.urls = new Map();      // shortcode -> url data
      this.analytics = new Map(); // shortcode -> click data array
   }

   // Methods: storeURL, getURL, shortcodeExists, recordClick, getAnalytics
}
```

## API Design (As Built)

### Implemented Endpoints

#### 1. Create Short URL

```
POST /shorturls
Content-Type: application/json

{
  "url": "[https://example.com](https://example.com)",
  "validity": 30,         // optional, default 30
  "shortcode": "custom123"  // optional
}

Response 201:
{
  "shortLink": "http://localhost:3000/custom123",
  "expiry": "2025-09-08T11:00:00.000Z"
}
```

#### 2. Get Statistics

```
GET /shorturls/custom123

Response 200:
{
```

```
  "shortcode": "custom123",
  "originalUrl": "[https://example.com](https://example.com)",
  "createdAt": "2025-09-08T10:30:00.000Z",
  "expiresAt": "2025-09-08T11:00:00.000Z",
  "isExpired": false,
  "totalClicks": 3,
  "clickHistory": [...]
}
```

#### 3. Redirect

```
GET /custom123

Response 302:
Location: [https://example.com](https://example.com)
```

#### 4. Health Check

```
GET /health

Response 200:
{
  "status": "healthy",
  "timestamp": "2025-09-08T10:30:00.000Z",
  "uptime": 1234
}
```

# Design Patterns

## 1. Express Middleware Pattern

```
// middleware/index.js
app.use(corsConfig);
app.use(requestLogger);
app.use(errorHandler);
app.use(notFoundHandler);
```

## 2. Controller Pattern

```
// controllers/urlController.js
async function createShortUrl(req, res) {
  // Handle URL creation logic
}
```

### 3. Simple Repository Pattern

```
// storage.js - Data access abstraction
class URLStorage {
  async storeURL(shortcode, urlData) { /* ... */ }
  async getURL(shortcode) { /* ... */ }
}
```

### 4. Utility Functions Pattern

```
// utils.js
function generateShortcode(length = 6) { /* ... */ }
function isValidUrl(url) { /* ... */ }
function createExpiryDate(minutes) { /* ... */ }
```

### 5. Logging Middleware Pattern

```
// Custom logging throughout application
await Log('backend', 'info', 'controller', 'URL created successfully');
```

# Logging Implementation

## Actual Logging Middleware Structure

```
// logging_middleware/src/
├── constants.js    # Validation constants
├── logger.js       # Core logging logic
└── index.js        # Main exports
```

## How Logging Actually Works

1. **Import in Backend:**

```
const { Log } = require('../../logging_middleware/src/index');
```

2. **Usage Throughout Code:**

```
await Log('backend', 'info', 'controller', 'URL created successfully');
await Log('backend', 'error', 'db', 'Failed to store URL');
```

3. **External API Call:**

```
// Sends POST to: [http://20.244.56.144/evaluation-
service/logs](http://20.244.56.144/evaluation-service/logs)
{
  "stack": "backend",
  "level": "info",
  "package": "controller",
  "message": "URL created successfully"
}
```

4. **Authentication:**

```
Authorization: Bearer <Token>
```

## Log Levels Used:

- debug - Detailed debugging info
- info - General operations
- warn - Warning conditions
- error - Error conditions
- fatal - Critical errors

## Backend Packages Logged:

- service - Server startup/shutdown
- controller - API request handling
- handler - Business logic
- db - Data operations
- route - Route access
- middleware - Middleware operations