System Design and Code Refactoring Proposal

# 1. Introduction

This document outlines proposed enhancements to the current URL shortener application, focusing on backend structure, frontend consistency, and long-term maintainability. The intent is to optimize the system’s reliability, scalability, and developer experience.

# 2. Existing Setup

## Frontend

- index.html: Traditional HTML file with embedded JavaScript logic and static UI elements.

- UrlForm.jsx: A React component responsible for sending POST requests to the backend using axios.

## Backend

- server.js: Node.js + Express application that handles URL shortening and redirection. It logs clicks and handles expiration but stores data in an in-memory array (records[]).

# 3. Observations and Issues

|  |  |
| --- | --- |
| Category | Details |
| Data Volatility | URLs are kept in memory; lost on server restart. |
| Frontend Sync | HTML frontend uses separate logic and doesn’t communicate with backend. |
| Redundant Logic | URL expiration, ID generation, and click tracking repeated on both sides. |
| Scalability Limits | No support for database, routing separation, or analytics APIs. |
| Error Feedback | User error messages are minimal and displayed via alert(). |

# 4. Target Architecture

[React Frontend]

↓

(API Calls)

↓

[Express Backend]

↓

[Database]

## Suggested Stack

- Frontend: React + Axios

- Backend: Node.js + Express.js

- Database: MongoDB (or SQLite for local testing)

# 5. Recommended Improvements

## A. Backend (Node.js/Express)

* - Persistent Data Layer: Replace in-memory `records[]` with a database collection (e.g., MongoDB).
* - REST API Structure: Split routes into modules and add endpoints like /:id/stats and /urls.
* - Validation Middleware: Sanitize and validate inputs using express-validator.
* - Logging and Monitoring: Use morgan for request logs and consider adding metrics tracking.

## B. Frontend (React)

* - Single Page Application (SPA): Remove HTML logic and use only React components.
* - Statistics Display: Show click count, expiry status, and click logs after URL creation.
* - Improved UI/UX: Use modern UI libraries and toast notifications.
* - Enhanced Error Handling: Replace alert() with contextual notifications.

# 6. Suggested Project Structure

project/  
├── client/ # React frontend  
│ ├── src/  
│ │ ├── components/  
│ │ │ ├── UrlForm.jsx  
│ │ │ └── UrlTable.jsx  
│ │ └── App.jsx  
│ └── public/  
│ └── index.html  
├── server/ # Express backend  
│ ├── routes/  
│ │ ├── shorten.js  
│ │ └── redirect.js  
│ ├── models/  
│ │ └── Url.js  
│ ├── db.js  
│ └── server.js  
├── package.json  
└── README.md

# 7. Benefits of These Changes

* - Reliable Storage: Data persists across server restarts.
* - Clear Codebase: Separated concerns between backend and frontend.
* - Consistent Behavior: Shared logic in one location, no duplication.
* - User Insights: Access to usage logs and URL statistics.
* - Ease of Expansion: Easy to add features like authentication or admin panel.

# 8. Implementation Roadmap

Step-by-step plan:

* - Connect MongoDB and migrate data storage from memory
* - Replace embedded HTML logic with React-based frontend
* - Build new API endpoints for stats and listing
* - Integrate validation and error management
* - Add UI enhancements and deploy to cloud