Project Report on

URL Shortener Web Application

Submitted By: Abhishek Kumar,

UID – 24MCC20006

Under the Guidance of

Mrs. Winky Bhatia

University Institute of Computing,

Chandigarh University,

Mohali, Punjab

INDEX:

Table of content:

Introduction

In the modern digital era, managing and sharing long, complex URLs can often become inconvenient and error-prone. URL shorteners provide a practical solution by converting lengthy links into short, easy-to-share URLs. This mini-project, titled **"URL Shortener Web Application"**, aims to replicate the core functionality of popular services like Bitly or TinyURL, with a clean, responsive interface and robust backend support.

The project is developed using a full-stack approach — the frontend is built using **React.js** for a dynamic and interactive user experience, styled with **Bootstrap** for responsiveness and visual appeal. The backend uses **Node.js** with **Express.js** to handle API requests and route management. Data persistence is achieved using **MongoDB**, a NoSQL database, to store the mapping between long and short URLs.

Users can input a long URL, which is then sent to the backend, processed, and a unique shortened version is generated. This short URL can then be used to redirect users back to the original long URL. The shortened links are stored in the MongoDB database for future redirection and reference.

This project serves as a practical implementation of real-world web technologies and illustrates the seamless integration of frontend and backend systems, along with database operations. It not only helps in understanding the structure of modern web applications but also strengthens the ability to manage and develop scalable full-stack projects.

Additionally, the project emphasizes clean code structure, modular development, and efficient routing. Through this hands-on implementation, learners can grasp crucial concepts in API design, state management, and secure data handling. This application not only fulfills an everyday digital need but also showcases how technology can simplify online interactions through well-crafted tools.

This system also demonstrates the importance of URL analytics and tracking, which are valuable features for businesses and marketers. While the basic version of this project covers the core functionality, it lays the groundwork for future enhancements like user authentication, click statistics, QR code generation, and custom short URLs. These advanced capabilities can significantly extend the usefulness of the application and provide a more personalized user experience.

The URL Shortener Web Application highlights the synergy between user-centric design and technical efficiency. It not only meets academic requirements as a mini-project but also prepares developers for building real-world solutions that are scalable, maintainable, and secure.

The application is built using a **modern full-stack architecture**:

- **Frontend**: Developed with **React.js**, offering a fast, responsive user interface styled with **Bootstrap** for a clean and professional look.
- **Backend**: Powered by **Node.js and Express.js**, responsible for processing user requests and handling URL shortening logic.

- **Database**: Utilizes **MongoDB** for storing original and shortened URLs, ensuring data persistence and fast retrieval.

The user journey is simple: enter a long URL into the input field, click to shorten it, and instantly receive a unique, shortened URL that redirects to the original link when used. These generated URLs are stored in the database for future access and redirection.

Beyond its utility, this project also demonstrates the practical integration of client-side and serverside technologies, emphasizing RESTful API usage, database interactions, and deployment considerations. It offers a valuable learning experience in web development, database management, and user interface design, making it a well-rounded academic and realworld application.

Objectives

The main objective of this mini-project is to design and implement a full-stack **URL Shortener Web Application** that allows users to convert long URLs into short, easily shareable links. Below are the specific goals this project aims to achieve:

1. **To provide a user-friendly interface**
   Create an intuitive and responsive frontend using **React.js** and **Bootstrap** that allows users to enter long URLs and receive shortened ones quickly.

2. **To implement a functional backend service**
   Use **Node.js** and **Express.js** to build a backend capable of receiving URL inputs, generating unique short codes, handling redirections, and managing API endpoints. 3. **To ensure reliable data storage**
   Integrate **MongoDB** to store and retrieve original and shortened URLs efficiently, ensuring data persistence and quick access. 4. **To demonstrate RESTful API integration**
   Develop a seamless communication layer between frontend and backend using REST APIs for creating and retrieving URL records.

5. **To enhance understanding of full-stack development**
   Offer hands-on experience in managing frontend, backend, and database interactions, covering the entire web development lifecycle.

6. **To enable URL redirection functionality**
   Allow users to use the shortened URL to be redirected to the original long URL, mimicking real-world URL shorteners like Bitly or TinyURL.

7. **To practice secure and optimized coding**
   Ensure basic input validation and error handling to prevent invalid URLs and improve application stability.

8. **To lay the groundwork for future enhancements**
   Design the system in a modular way so features like user login, click tracking, QR code generation, and custom short links can be added later.

System Design

The **URL Shortener Web Application** is designed with a modular and scalable architecture following the **Model-View-Controller (MVC)** pattern. The system is divided into three primary layers — **Frontend (Client-Side)**, **Backend (Server-Side)**, and **Database (Storage Layer)** to maintain a clear separation of concerns and facilitate maintainability.

1. Frontend (View Layer):

- **Technology Used**: React.js, Bootstrap
- **Function**: Collects user input (long URL), sends it to the backend via API calls, and displays the resulting shortened URL.
- **Features**:
  - Responsive layout for desktop and mobile
  - Error messages for invalid inputs
  - Display of short URLs with clickable links

2. Backend (Controller Layer):

- **Technology Used**: Node.js, Express.js
- **Function**: Handles incoming requests, processes them, and communicates with the database.
- **Key Functions**:
  - Accepts POST requests containing long URLs
  - Generates unique short codes
  - Stores and retrieves URL mappings
  - Handles redirection based on the short code

3. Database (Model Layer):

- **Technology Used**: MongoDB
- **Function**: Stores the mapping of long URLs to short codes and handles quick data retrieval for redirection.
- **Collections**:
    - URL: Contains fields such as originalUrl, shortUrl, createdAt, and optionally clickCount.

Workflow Overview:

1. The user enters a long URL on the frontend.
2. A POST request is sent to the backend API endpoint /shorten.
3. The backend checks for an existing entry or creates a new unique short code.
4. The short URL and its mapping are stored in MongoDB.
5. The backend responds with the newly generated short URL.
6. The frontend displays the shortened URL to the user.
7. When someone accesses the short URL, a GET request is sent to the backend, which fetches the original URL and redirects the user.

Technology Stack

This project is built using a **modern full-stack JavaScript architecture**, ensuring seamless integration between all components. Each layer of the application has been carefully chosen to provide optimal performance, scalability, and ease of development.

Frontend:

- **React.js**:
  powerful JavaScript library for building user interfaces. React's component-based architecture allows for modular, maintainable, and reusable UI components.
- **Bootstrap**:

Used for styling and layout. It provides a responsive design framework that helps the application look clean and professional across various devices and screen sizes.

- **Axios**:
  Promise-based HTTP client used to communicate with the backend API endpoints, making asynchronous data fetching simple and efficient.

Backend:

- **Node.js**:
  runtime environment that allows JavaScript to be run on the server side. It provides a nonblocking I/O model that enhances performance.

- **Express.js**:
  fast and minimalist web framework for Node.js. It simplifies the creation of robust APIs, handling routing and middleware efficiently.

Database:

- **MongoDB**:
  NoSQL, document-oriented database used to store the mapping of long and short URLs. It provides flexibility, scalability, and easy integration with Node.js.

- **Mongoose** (if used):
  An ODM (Object Data Modeling) library for MongoDB and Node.js. It simplifies data validation, schema definition, and interaction with the database.

Other Tools and Dependencies:

- **Nodemon** (for development):

  Automatically restarts the server when code changes are detected.

- **React Scripts**:

  Used to run and build the React application easily.

- **Postman** (for testing APIs):

  Helpful in testing and debugging API endpoints during development.

Features

The **URL Shortener Web Application** includes a variety of features that ensure both usability and functionality. These features reflect real-world requirements and enhance the overall user experience.

1. URL Shortening

- Converts long URLs into short, manageable links.
- The short URL is unique and can be easily shared or embedded.

2. Redirection to Original URL

- When a user visits the short URL, the system redirects them to the corresponding original long URL stored in the database.

3. MongoDB Integration

- Stores long URLs along with their shortened versions.
- Ensures data persistence, allowing previously generated URLs to remain active and usable.

4. API-Based Architecture

- Backend built using RESTful APIs with Node.js and Express.
- Ensures separation of concerns and smooth frontend-backend communication.

5. Responsive and User-Friendly UI

- Frontend developed with React.js and styled using Bootstrap.
- Fully responsive layout that adapts to mobile, tablet, and desktop screens. **6. Error Handling**
- Alerts and error messages notify users of invalid inputs or failed requests.
- Backend checks for malformed URLs or duplicate entries.

7. Developer-Friendly Codebase

- Clean, modular code structure that is easy to understand and extend.
- Commented sections and reusable components for efficient maintenance.

8. Tailwind/Bootstrap Ready UI

- Option to switch styling frameworks or improve UI further using modern design practices.

System Architecture

The **URL Shortener Web Application** follows a modular full-stack architecture that ensures scalability, maintainability, and performance. It consists of three primary layers: frontend, backend, and database. Each component plays a specific role in the workflow of the application.

1. Frontend (Client Side)

- **Technology:** React.js, Bootstrap
- **Role:** The frontend provides the user interface through which users interact with the application. It includes input fields for submitting URLs, buttons for generating short URLs, and displays the results in a user-friendly format.
- **Features:**
- Form to input long URLs
- Display area for the generated short URL
- Client-side validations and alerts
- Responsive design with Bootstrap

2. Backend (Server Side)

- **Technology:** Node.js, Express.js
- **Role:** The backend handles business logic, URL validation, short code generation, and communication with the database. It exposes RESTful APIs used by the frontend to interact with the system.
- **Features:**
- API endpoint to receive long URLs and return short ones
- Endpoint to redirect short URLs to their original destinations
- Middleware for error handling and logging
- URL validation and uniqueness checking

3. Database (Storage Layer)

- **Technology:** MongoDB (NoSQL)

- **Role:** MongoDB stores the mapping between original long URLs and their corresponding short URLs. Each entry includes timestamps and can be expanded to store analytics or userrelated data in the future.
- **Features:**
- Efficient storage and retrieval of URL data
- Schema-less design to allow flexibility
- High scalability for growing datasets

Workflow Overview:

1. The user inputs a long URL in the React frontend.
2. A POST request is sent to the Express.js backend.
3. The backend checks if the URL is valid and either:
    - Returns an existing short URL from the database, or
    - Generates a new short URL and stores the mapping in

MongoDB.

4. The frontend receives the response and displays the short URL.
5. When a short URL is accessed, the backend finds the original URL in MongoDB and redirects the user.

Implementation Details

The development of the URL Shortener Web Application was carried out step-by-step to ensure smooth integration of all technologies involved. Each part of the application — frontend, backend, and database — was designed to interact seamlessly using a RESTful API approach.

1. Project Structure

The main folder is named AIP PROJECT and contains two subfolders:

- /frontend: React-based client application
- /backend: Node.js and Express-based server application

2. Frontend Implementation (React + Bootstrap)

- **React Functional Components** were used to create the UI.
- **Axios** is used for making HTTP POST requests to the backend.
- **Bootstrap** was integrated to style components and ensure responsiveness across all devices.
- Features include:
- Input field to enter the long URL
- Button to trigger the shortening process
- Dynamic display of the shortened URL
- Validation and alert messages for user

feedback

Example: <input type="text"

className="form-control" placeholder="Enter long

URL" value={longUrl} onChange={(e)

=> setLongUrl(e.target.value)}

/>

3. Backend Implementation (Node.js + Express.js)

- A REST API was developed with two main routes:
- POST /shorten — accepts a long URL and returns a shortened version
- GET /:shortId — redirects to the original long URL
- URL validation is done before storing.
- If a long URL already exists, it returns the existing short one.
- **Short IDs** are generated using custom functions or third-party packages like shortid or nanoid.

Example: js

```
CopyEdit app.post('/shorten', async (req, res) =>
{  const { originalUrl } = req.body;  const
shortId = generateShortId();  const shortUrl =
`${BASE_URL}/${shortId}`;  await
Url.create({ originalUrl, shortUrl });  res.json({ shortUrl
});
});
```

4. Database (MongoDB)

- MongoDB is used to store the mappings between long URLs and short URLs.
- Each entry includes:
- Original URL
- Shortened URL
- Creation date/time (timestamp)
- Mongoose is used as an ODM to simplify database interactions.

Example Schema:

```
js
CopyEdit  const  urlSchema  =  new
mongoose.Schema({
 originalUrl: String,  shortUrl: String,  date: {
type: String, default: Date.now }
});
```

5. Other Tools & Enhancements

- **React Bootstrap Alerts** for success/error messages

- **Tailwind CDN / Bootstrap** for glossy background and modern UI
- **Proxy** in package.json set to communicate between frontend and backend
- **Environment Variables** (optional) to store base URLs and database URIs securely

Challenges Faced

During the development of the URL Shortener Web Application, several challenges were encountered that tested both the technical knowledge and problem-solving skills involved in fullstack development. Below are some of the notable issues faced and how they were resolved:

1. Tailwind CSS Setup Issues

- **Problem:** Tailwind CSS did not install correctly using npx tailwindcss init -p, and the system did not recognize tailwind as a command.
- **Solution:** Switched to using **Tailwind via CDN** temporarily, ensuring that basic styling could still be applied without full local configuration. Bootstrap was later used as the main styling framework due to its seamless integration.

2. Backend Not Responding in VS Code

- **Problem:** Even though the server was running (confirmed by console logs), accessing http://localhost:3000 showed the error "This site can't be reached."
- **Solution:** Confirmed that the backend server and frontend were running on different ports. Used the proxy key in package.json to correctly route API calls from the frontend to the backend.

3. Webpack and Crypto Module Errors

- **Problem:** Encountered Webpack errors like digital envelope routines::unsupported due to incompatibility with Node.js version 17+.
- **Solution:** Downgraded Node.js to version 16 or added an environment variable NODE_OPTIONS=--openssl-legacy-provider to allow the project to build successfully.

4. MongoDB Connection Warnings

- **Problem:** Deprecated options like useNewUrlParser and useUnifiedTopology showed warnings in the console.
- **Solution:** These options were removed from the connection string, as they are no longer required in Mongoose 6+.

5. Deployment & Localhost Conflicts

- **Problem:** Confusion between multiple localhost ports (3000, 3002, 5000) sometimes caused the wrong service to be accessed or failed to load.
- **Solution:** Standardized the frontend on port 3000 and the backend on 5000. Added a proxy to the frontend's package.json to handle requests appropriately.

6. Styling and Responsiveness

- **Problem:** Making the UI look modern and mobile-friendly using just basic CSS was difficult.
- **Solution:** Integrated Bootstrap for responsive layouts, buttons, forms, and alerts to enhance the visual design and usability of the app.

These challenges helped reinforce debugging skills, understanding of environment configurations, and highlighted the importance of proper setup and communication between services in a fullstack application.

Conclusion

The development of the **URL Shortener Web Application** has been a rewarding journey that combined essential concepts of full-stack web development. By replicating the functionality of real-world services like Bitly, the project provided hands-on experience in building scalable and user-friendly web applications.

From setting up a modern React frontend to designing a Node.js and Express.js-powered backend, and integrating MongoDB for data persistence, each phase of the project contributed to a deeper understanding of the technology stack. It also highlighted the importance of API communication, routing, error handling, and database design.

The user interface, styled with Bootstrap, ensured a responsive and visually appealing experience, while the backend ensured efficient processing and redirection of URLs. Real-world challenges

such as dependency issues, environment configuration errors, and deployment inconsistencies were tackled, adding to the learning value of the project.

This project not only serves as a useful tool for shortening and managing URLs but also as a demonstration of practical development skills that are relevant in both academic and industry settings. It reinforces the knowledge of RESTful architecture, modular coding practices, and efficient database interactions.

Overall, the project successfully achieves its objectives and provides a solid foundation for building more advanced web applications in the future.

Future Enhancements

While the current version of the URL Shortener Web Application is fully functional and meets its core objectives, there are several opportunities to expand and improve its features to align with industry standards and enhance user experience:

1. **User Authentication & Profiles** ○ Implement a login system allowing users to create accounts and manage their shortened URLs.
    - ○ Users can view analytics, edit or delete their links, and manage privacy settings.
2. **Click Analytics Dashboard** ○ Track the number of clicks on each shortened URL. ○ Display geolocation, device type, and browser information of users who accessed the link.
3. **Custom Short URLs** ○ Allow users to define their own custom slugs (e.g., bitly.com/myproject) instead of random strings.
4. **QR Code Generation** ○ Automatically generate a QR code for each short URL for easy sharing and mobile access.

6. **Expiration & Deactivation Options**

o Provide an option to set an expiration time for URLs.

o Allow manual deactivation of links when they're no longer needed.

6. **Admin Dashboard** ○ Create an admin panel to monitor system usage, manage users, remove spam links, and ensure smooth operation.

7. **Improved Error Handling & Validation** ○ Enhance user feedback with more specific error messages for invalid URLs or server issues.

8. **Deployment on Cloud Services** ○ Deploy the application on platforms like Vercel, Netlify (for frontend), and Render or Railway (for backend + database).

○ Ensure HTTPS, domain name, and continuous integration support.

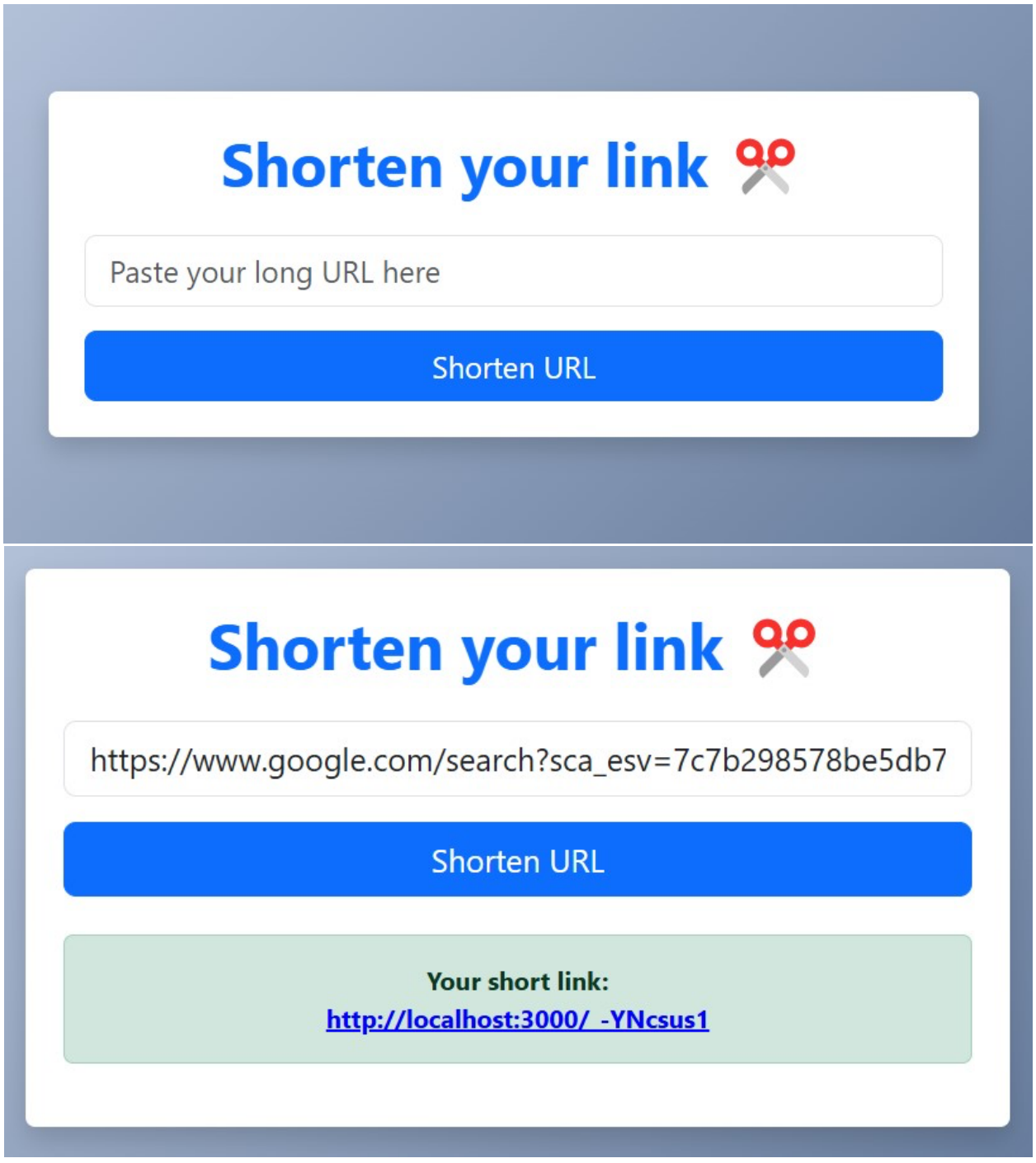
8. **Dark Mode / Theme Switcher**

○ Add a toggle to switch between light and dark themes for better accessibility.

10. **Performance Optimization**

- Apply lazy loading, efficient state management, and code splitting to improve the performance of both frontend and backend.

Result


Shorten your link
Paste your long URL here
Shorten URL
Shorten your link
https://www.google.com/search?sca_esv=7c7b298578be5db7
Shorten URL
Your short link:
http://localhost:3000/_-YNcsus1