A Project Report on

LINK SHORTENER

submitted in partial fulfillment for the Degree of

B. Tech

in

Computer Science & Engineering

6th Semester

By

Aadrsh Bajpai (22L-CSE-01)
Amit Kumar (22L-CSE-08)
Sudhakar (22L-CSE-44)

Under the guidance of

Ms. Bhawna Sharma
Dept. of CSE

pursuing in

Department of Computer Science and Engineering

BM College of Technology and Management

on

May, 2024

CERTIFICATE

This is to certify that the project report entitled **Link Shortener** submitted by **Mr. Aadrsh, Mr. Amit** and **Mr. Sudhakar** to the **BM College of Technology and Management, Gurugram**, in partial fulfillment for the award of the degree of **B. Tech (in Computer Science and Engineering)** is a bona fide record of project work carried out by them under my/our supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree or diploma.

Ms. Bhawna Sharma

Department of C.S.E.

DECLARATION

We declare that this project report titled **Link Shortener** submitted in partial fulfillment of the degree of **B. Tech (in Computer Science and Engineering)** is a record of original work carried out by us under the supervision of **Ms. Bhawna Sharma**, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Mr. Aadrsh Bajpai **22L-CSE-01**

Mr. Amit K. Yadav **22L-CSE-08**

Mr. Sudhakar Kumar **22L-CSE-44**

ACKNOWLEDGEMENT

We would like to thank **Ms. Bhawna Sharma, Lecturer, BM Group of Technology and Management** for her support and guidance in completing our project on **Link Shortener** which helped us understand the modern web technologies and made us learn how we can maximize the potential of it.

We would like to take this opportunity to express our gratitude to each other. The project would not have been successful without the cooperation and input of each member.

Aadrsh

Amit

Sudhakar

ABSTRACT

This project explores the advancement of link shortening technology, aiming to enhance its capabilities beyond basic URL compression. Motivated by real-world incidents, the project proposes an in-house solution with customizable short codes, link management features, QR code generation, and a user-friendly dashboard. Future developments include detailed analytics, time-sensitive link expiration, and enhanced security. The project aims to address the growing need for efficient link management in personal and business communications, offering a sophisticated solution to elevate organizational capabilities in the digital realm.

This project endeavors to push the boundaries of link shortening technology by introducing innovative features and capabilities. Motivated by practical experiences highlighting the limitations of existing solutions. By addressing the evolving demands of modern communication, this project aims to empower organizations with a sophisticated toolset for efficient link management, enhancing user experience and organizational capability in the digital landscape.

Table of Contents

Link Shortener

Introduction

Link shorteners are commonly used tools that transform lengthy URLs into more manageable, short links that are easier to share and remember. While the basic functionality of a link shortener is straightforward, this project aims to explore and expand the capabilities of this seemingly simple technology.

Why Use a Link Shortener?

Imagine trying to communicate the full name of Mr. Iyer from the movie *Dhamaal* (2007) every time you referred to him. His full name is "Shivavenkata Srinivasna Trichipalli Yekeparampir Perambdur Chinnaswami Muttuswami Venugopal Iyer", but for ease, we simply call him Iyer. Similarly, long URLs can be cumbersome and error-prone when shared; link shorteners condense these URLs into digestible snippets that are easier to handle and remember.

How It Works

At its core, a link shortener requires minimal space and processing power. It involves storing the original URL and assigning it a unique shortcode. This relationship is akin to using variables in programming, where the variable name is used to reference the stored data without needing to recall the data itself.

Motivation for This Project

During a recent internship, an incident involving a QR code linked to a registration form highlighted the limitations of using external link shortening services. The link was mistakenly blocked as suspicious, causing confusion and reducing event participation. This incident underscored the need for a reliable, in-house URL shortening solution that organizations can control directly, ensuring stability and access when most needed.

Current Features

- **Custom Shortcodes**: Customize the ending of the shortened URL.
- **Link Management**: Modify existing links; change both the destination URL and the shortcode at any time.
- **QR Code Generation**: Automatically generate QR codes for links by simply appending ".qr" to any shortcode.
- **Dashboard**: A user-friendly dashboard for creating, viewing, and managing shortened links with options to copy, edit, or delete each link.

Proposed Features

- **Analytics**: Track link usage with detailed analytics, including access timelines and device usage statistics.
- **Time Bomb Links**: Create links that automatically expire after a set period or event, perfect for time-sensitive content like registration deadlines.
- **Countdown Links**: This feature will provide us a feature to enable the links at a certain time it is somewhat similar to Time Bomb Links, but it focuses on enabling part.
- **Enhanced Security with Authentication**: Implement user authentication to ensure that only authorized personnel can create or modify links, making this tool particularly suited for corporate environments.
- **Custom QR Codes:** This will allow the users to create QR code with images in it to make it look more professional.
- **Proxy Serve**: This will serve the contents after downloading them first, instead of redirecting. This opens the gate for using this to shorten API calls and other possibilities.

Benefits of an In-House Solution

Using a dedicated link shortener provides significant advantages:

- **Control**: Full authority over link availability and management.
- **Security**: Reduced risk of links being mistakenly flagged or blocked.
- **Customization**: Tailor the tool's functionality to meet specific organizational needs.
- **Cost-Effective**: Can be hosted economically on low-cost cloud instances or on-premises.

Conclusion

As digital interactions continue to dominate business and personal communications, the necessity for efficient link management becomes undeniable. This project proposes a sophisticated, customizable link shortening solution that not only improves upon basic functionalities but also introduces advanced features to elevate organizational capability and user experience in the digital realm.

Workflow Overview**:**

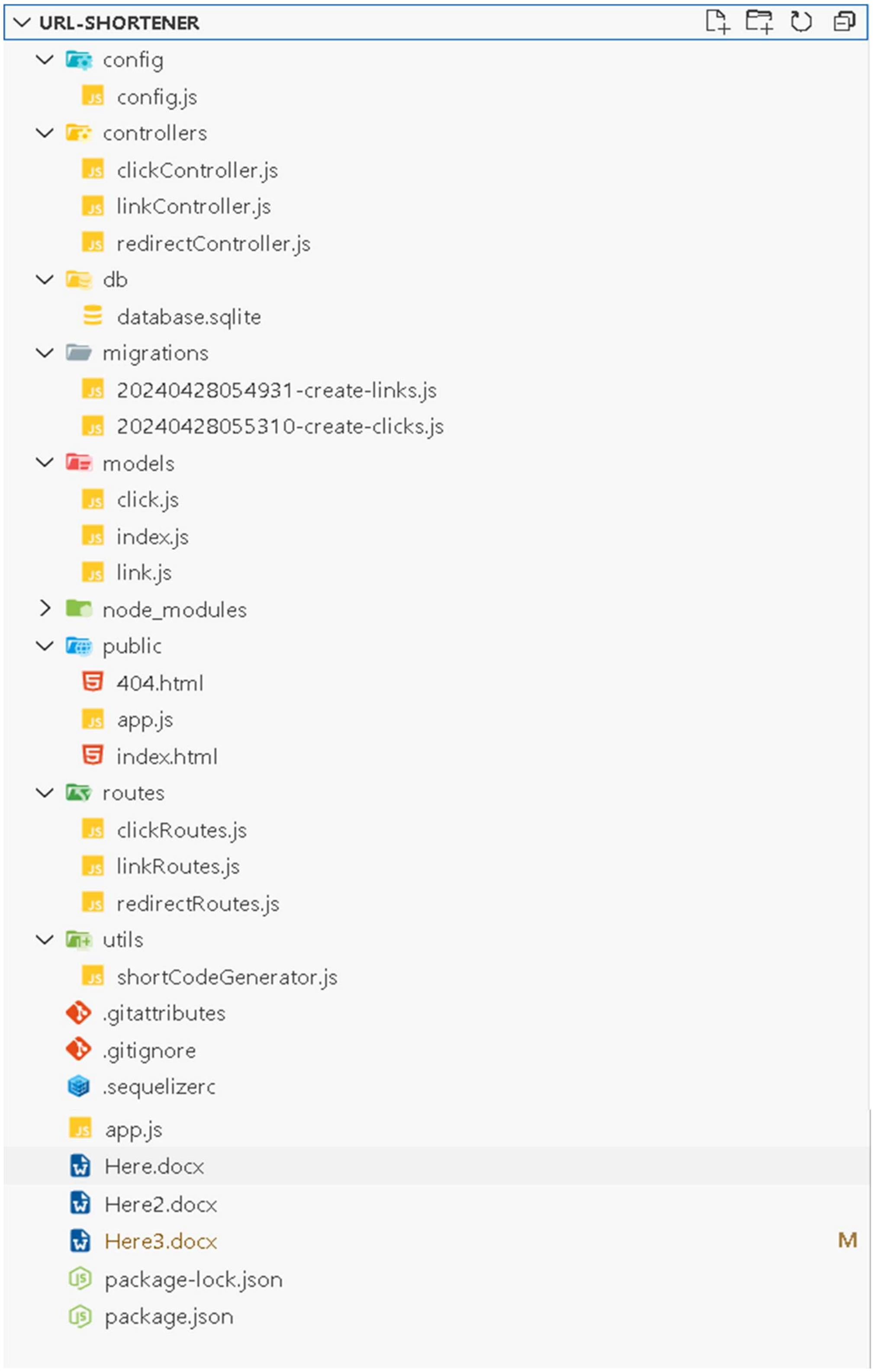
Technologies Used

1. **Version Control**: We utilize GitHub for version control, facilitating smoother development and enabling team collaboration.
2. **Environment**: Our primary environment is Node.js (v 18.04.0). We leverage several essential packages for our Node.js environment:
    - **dotenv**
    - **express**
    - **express-useragent**
    - **request-ip**
    - **sequelize**
    - **sqlite3**
    - **qrcode**
3. **Hosting**: The project is hosted on AWS, utilizing a t2.micro instance.
4. **Domain Management**: Our domain is managed by Cloudflare.

Methodology

- **Agile Project Management**: We adhere to Agile methodologies, which allow us to iteratively develop and adapt to changes efficiently.
- **Incremental Development**: We began by implementing core functionalities and iteratively designed additional features as needed. This approach enables us to manage unit tests for functionalities effectively.
- **Focus on Current Requirements**: Rather than overwhelming ourselves with future requirements, we prioritize and implement current requirements first, incorporating others as they arise. This approach ensures a more manageable development process and allows for agile responses to evolving needs.

Our Project Structure



```
URL-SHORTENER
  config
    config.js
  controllers
    clickController.js
    linkController.js
    redirectController.js
  db
    database.sqlite
  migrations
    20240428054931-create-links.js
    20240428055310-create-clicks.js
  models
    click.js
    index.js
    link.js
  node_modules
  public
    404.html
    app.js
    index.html
  routes
    clickRoutes.js
    linkRoutes.js
    redirectRoutes.js
  utils
    shortCodeGenerator.js
  .gitattributes
  .gitignore
  .sequelizerc
  app.js
  Here.docx
  Here2.docx
  Here3.docx                                      M
  package-lock.json
  package.json
```

Querying the Database

Sequelize ORM

In Node.js, Sequelize is a popular ORM (Object-Relational Mapping) library used to interact with relational databases such as MySQL, PostgreSQL, SQLite, and MSSQL. Sequelize abstracts database interactions, allowing developers to manipulate and query data using JavaScript objects and functions instead of writing raw SQL queries. One of the core components of Sequelize is its model system, which represents tables in the database as classes in JavaScript.

Database Used

The database we are using is SQL, however for small project we preferred Lite version of SQLite, this enabled us to share db along with other data as sqlite db being lightweight for sharing. Also, it requires minimal efforts to setup, allowing us to focus on project itself keeping us away from the technicality of creating a new instance of sql server.

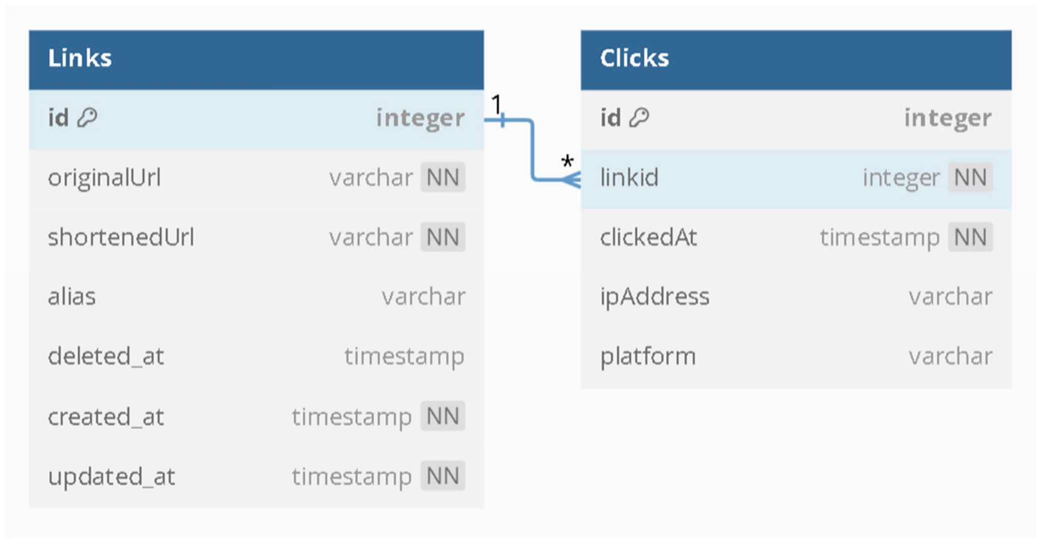
Database Schema

The datascheme can be described by the shortcode

```
Table Links {
  id integer [primary key]
  originalUrl varchar [not null]
  shortenedUrl varchar [unique, not null]
  alias varchar [null]
  deleted_at timestamp [null]
  created_at timestamp [not null]
  updated_at timestamp [not null]
}
Table Clicks {
  id integer [primary key]
  linkid integer [not null]
  clickedAt timestamp [not null]
  ipAddress varchar [null]
  platform varchar [null]
}
Ref: Links.id < Clicks.linkid
```

Visualization

Here's the visualization of our db diagram



Links	
id	integer
originalUrl	varchar NN
shortenedUrl	varchar NN
alias	varchar
deleted_at	timestamp
created_at	timestamp NN
updated_at	timestamp NN

Clicks	
id	integer
linkid	integer NN
clickedAt	timestamp NN
ipAddress	varchar
platform	varchar

Summary of Tables:

Links Table

- **id**: Unique identifier for the links, also serves as the primary key.
- **originalUrl**: The original long URL which we want to shorten. Cannot be null.
- **shortenedUrl**: The suffix for our short URL. Cannot be null.
- **alias**: Optional nickname given to URLs. Can be null.
- **deleted_at**: Refers to the date the record was deleted, i.e., deletion date of the link.
- **created_at**: Creation date of the record, i.e., creation date of the link.
- **updated_at**: Last updated time of the link.

Clicks Table

- **id**: Unique identifier for each click, also serves as the primary key.
- **linkId**: The ID of the link to which the click is associated.
- **clickedAt**: Timestamp indicating when the link was clicked.
- **ipAddress**: IP address of the visitor who clicked the link.
- **platform**: The platform the user is using while accessing the link.

Relationship

- The relationship defined between both tables is "**Link has Many Clicks**" (**One to Many**). This relationship will be reflected in our data models and migration scripts.

Index

- The index in SQL helps SQL to search the content faster, basically the index in SQL is implicitly defined for primary key, but we can define it on other aspects with criteria of Not Null and Unique Values. We have applied index of links table on **shortenedUrl** which we will discuss later.

Database Configuration

This file contains configuration settings related to the database, which may vary depending on the environment in which our application is deployed. We can have different database connections configured based on the requirements of the environment and the needs of our application.

```
require('dotenv').config();
module.exports = {
  development: {
    dialect: 'sqlite',
    storage: './db/database.sqlite',
  },
  test: {
    dialect: 'sqlite',
    storage: './db/database.sqlite'
  },
  production: {
    dialect: 'sqlite',
    storage: './db/database.sqlite'
  }
};
```

Database Migration Scripts

Migration scripts are a crucial component in the management and evolution of databases, ensuring that changes in the database schema or data are applied systematically and reliably across different environments (development, testing, production). These scripts are essentially sets of commands that modify the database structure (like tables, columns, indexes) or manipulate data (adding, modifying, or deleting entries) in a controlled manner.

Links Table Migration Script

This script was used by sequelize to create the table **Links** by in result of migration.

```
"use strict";
module.exports = {
  up: async (queryInterface, Sequelize) => {
    // Create the 'Links' table
    await queryInterface.createTable("Links", {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER,
      },
      originalUrl: {
        type: Sequelize.STRING,
        allowNull: false,
      },
      shortenedUrl: {
        type: Sequelize.STRING,
        allowNull: false,
      },
      alias: {
        type: Sequelize.STRING,
        allowNull: true,
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
        defaultValue: Sequelize.literal("CURRENT_TIMESTAMP"),
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
        defaultValue: Sequelize.literal("CURRENT_TIMESTAMP"),
      },
      deletedAt: {
        type: Sequelize.DATE,
        allowNull: true,
      },
    });

    // Add a unique index to the 'shortenedUrl' column
    await queryInterface.addIndex("Links", ["shortenedUrl"], {
      unique: true,
      fields: ["shortenedUrl"],
    });
  },

  down: async (queryInterface, Sequelize) => {
    // Remove the unique index from the 'shortenedUrl' column
    await queryInterface.removeIndex("Links", "shortenedUrl");
    // Drop the 'Links' table
    await queryInterface.dropTable("Links");
  },
};
```

Clicks Table Migration Script

This script was used by sequelize to create the table **Clicks** by in result of migration.

```
"use strict";


module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable("Clicks", {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER,
      },
      linkId: {
        type: Sequelize.INTEGER,
        allowNull: false,
        references: {
          model: "Links",
          key: "id",
        },
      },
      clickedAt: {
        allowNull: false,
        type: Sequelize.DATE,
        defaultValue: Sequelize.literal("CURRENT_TIMESTAMP"),
      },
      ipAddress: {
        type: Sequelize.STRING,
        allowNull: true, // Depending on requirements, this can be set to
false if IP is always expected
      },
      platform: {
        type: Sequelize.STRING,
        allowNull: true, // This can also be a non-nullable field, if
platform information is always available
      },
    });
  },
  down: async (queryInterface, Sequelize) => {
    await queryInterface.dropTable("Clicks");
  },
};
```

Models

Models in Sequelize are the essence of Sequelize's ORM capabilities. A model in Sequelize is a representation of a table in a database. Each model maps to a single table in the database and instances of a model represent rows in that table. Models define the structure of the table including its columns, their data types, and other attributes like validations, default values, and relationships to other tables.

Link Model

This is used by controllers and sequelize to reference the table **Links** as an object in server.
Here in the link model, we've implemented **paranoid : true** this flag ensures that if we call destroy on any row this will soft delete the row, and not actually deleting the row.

```
// models/link.js
const { Model, DataTypes } = require("sequelize");

module.exports = (sequelize) => {
  class Link extends Model {}
  Link.init(
    {
      originalUrl: DataTypes.STRING,
      shortenedUrl: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true,
      },
      alias: DataTypes.STRING,
      deletedAt: DataTypes.DATE,
    },
    {
      sequelize,
      modelName: "Link",
      paranoid: true, // This enables soft deletes, will use deletedAt.
      timestamps: true, // Explicitly stating to use timestamps.
      indexes: [
        {
          unique: true,
          fields: ["shortenedUrl"],
        },
      ],
    }
  );
  return Link;
};
```

Click Model

This is used by controllers and sequelize to reference the table **Clicks** as an object in server.
In the click model, we are storing the clicks that we've received on the links, this Click Model will be helpful in creating analysis for the links not just counting how many times the links are clicked.
It uses reference id of Links to associate the Click with the Link.

```
const { Model, DataTypes } = require("sequelize");

module.exports = (sequelize) => {
  class Click extends Model {}
  Click.init(
    {
      linkId: {
        type: DataTypes.INTEGER,
        allowNull: false,
        references: {
          model: "Links", // This model name should match the table name if
not explicitly defined
          key: "id",
        },
      },
      clickedAt: {
        type: DataTypes.DATE,
        allowNull: false,
        defaultValue: DataTypes.NOW, // Use NOW for current timestamp
      },
      ipAddress: {
        type: DataTypes.STRING,
        allowNull: true, // Adjust allowNull based on whether IP address is
mandatory
      },
      platform: {
        type: DataTypes.STRING,
        allowNull: true, // Adjust allowNull based on whether platform is
mandatory
      },
    },
    {
      sequelize,
      modelName: "Click",
      timestamps: false, // Ensure this is false if you are managing created
timestamps manually
    }
  );
  return Click;
};
```

Controllers

In the context of Node.js web applications, especially those built using frameworks like Express.js, controllers are key components that manage the logic between user interface actions and data models. Controllers act as intermediaries, processing incoming requests, interacting with models to access and manipulate data, and sending back the appropriate responses to the client.

Links Controller

Contains essential functions to fetch or manipulate the data from Links Table.

```
const { Link, Click } = require("../models");
const Sequelize = require("sequelize");

const generateShortCode = require("../utils/shortCodeGenerator"); // Import
the utility function

const createLink = async (req, res) => {
  const { originalUrl, alias, shortenedUrl } = req.body;
  try {
    const { success, code, message } = await generateShortCode(shortenedUrl);
    if (!success) {
      return res.status(409).json({ error: message }); // 409 Conflict is
often used for duplicate resource
    }
    const link = await Link.create({ originalUrl, shortenedUrl: code, alias
});
    res.json(link);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

const getLink = async (req, res) => {
  const { id } = req.params;
  try {
    const link = await Link.findByPk(id);
    if (!link) {
      return res.status(404).json({ message: "Link not found" });
    }
    res.json(link);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

const updateLink = async (req, res) => {
  const { id } = req.params;
  const { originalUrl, shortenedUrl, alias } = req.body;
  try {
    const link = await Link.findByPk(id);
    if (!link) {
      return res.status(404).json({ message: "Link not found" });
    }

    // Generate or validate the new shortened URL
    const { success, code, message } = await generateShortCode(shortenedUrl);
    if (!success) {
      return res.status(409).json({ error: message }); // Conflict if the new
shortened URL is already in use
    }

    // Update the link with new values
    link.originalUrl = originalUrl;
    link.shortenedUrl = code; // Use the new or validated code
    link.alias = alias;
    await link.save();
    res.json(link);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

const deleteLink = async (req, res) => {
  const { id } = req.params;
  try {
    const link = await Link.findByPk(id);
    if (!link) {
      return res.status(404).json({ message: "Link not found" });
    }
    link.shortenedUrl = link.shortenedUrl + "_" + link.id;
    await link.save();
    await link.destroy();
    res.json({ message: "Link deleted" });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

const getAllLinks = async (req, res) => {
  try {
    console.log("getAllLinks called");
    const links = await Link.findAll({
      attributes: [
        "id",
        "originalUrl",
        "shortenedUrl",
        "alias", // include all required Link attributes
        [Sequelize.fn("COUNT", Sequelize.col("clicks.id")), "clickCount"], //
Aggregate function to count Clicks
      ],
      include: [
        {
          model: Click,
          as: "clicks",
          attributes: [], // No attributes needed from Clicks
        },
      ],
      group: [
        "Link.id",
        "Link.originalUrl",
        "Link.shortenedUrl",
        "Link.alias",
        "Link.deletedAt",
        "Link.createdAt",
        "Link.updatedAt",
      ], // Ensure grouping by all Link attributes
      order: [["updatedAt", "DESC"]], // Sorting links by updatedAt in
descending order
    });

    // Converting to JSON might be necessary to properly see the results
    res.json(links.map((link) => link.toJSON()));
  } catch (error) {
    console.error("Error in getAllLinks:", error);
    res.status(500).json({ error: error.message });
  }
};

module.exports = {
  createLink,
  getLink,
  updateLink,
  deleteLink,
  getAllLinks,
};
```

Click Controller

Contains essential functions to fetch or manipulate the data from Click Table. This will be used to server analytics that’s why there are not much controller functions in it.

```
const { Click, Link } = require("../models");

const recordClick = async (linkId, ipAddress, platform) => {
  const clickedAt = new Date();
  try {
    const click = await Click.create({
      linkId,
      clickedAt,
      ipAddress,
      platform,
    });
    return click;
  } catch (error) {
    throw error;
  }
};

const getClicks = async (req, res) => {
  const { linkId } = req.params;
  try {
    const clicks = await Click.findAll({
      where: { linkId },
      include: [Link],
    });
    res.json(clicks);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

module.exports = {
  recordClick,
  getClicks,
};
```

Redirect Controller

Contains the function call to get the original link from the database querying the index of **shortenedUrl** that is being received in request object and redirect the client response to that URL. Along with it, it also saves the important parameters from the request

And if the requested shortcode is not found then it shows a **404.html** page which can be seen in last page of our project.

```
const { Link } = require("../models");
const { recordClick } = require("./clickController");
const requestIp = require("request-ip");

const getExternalLink = async (req, res) => {
  try {
    const { shortcode } = req.params;
    const link = await Link.findOne({ where: { shortenedUrl: shortcode } });
    if (link) {
      recordClick(link.id, requestIp.getClientIp(req),
req.useragent.platform);
      res.redirect(link.originalUrl);
    } else {
      res.status(404).redirect("/404.html");
    }
  } catch (error) {
    console.error("Database or server error:", error);
    res.status(500).send("Server error");
  }
};

module.exports = {
  getExternalLink,
};
```

Routes

Routes are an integral part of the architecture, serving as the means by which applications respond to client requests at different URLs (also known as paths or endpoints). They play a critical role in defining how an application responds to various HTTP methods (GET, POST, PUT, DELETE, etc.), linking these methods with the appropriate business logic and output.

Link Routes

```
const express = require('express');
const router = express.Router();
const { createLink, getLink, updateLink, deleteLink, getAllLinks } = 
require('../controllers/linkController');

router.post('/', createLink);
router.get('/:id', getLink);
router.get('/', getAllLinks);  // Fetch all links
router.put('/:id', updateLink);
router.delete('/:id', deleteLink);

module.exports = router;
```

Click Routes

The click routes for now only provides routes for the number of clicks a link has received.

```
const express = require("express");
const router = express.Router();
const { recordClick, getClicks } = require("../controllers/clickController");

router.get("/:linkId", getClicks);

module.exports = router;
```

Redirect Routes

The Redirect Routes is important part of the project as they are receiving the requests shortcode and is in frontline when the third user is interacting with a short link.

In the redirect route we've introduced a variable that receives the shortcode and passes it to the controller.

In the redirect route we've also introduced a .qr function that returns the QR code of the short url.

```
const express = require("express");

const qr = require("qrcode");

const router = express.Router();
const { getExternalLink } = require("../controllers/redirectController");
```

```
router.get("/:shortcode.qr", (req, res) => {
  const { shortcode } = req.params;
  // Generate QR code for the long URL
  qr.toDataURL("https://codedar.win/" + shortcode, (err, qrUrl) => {
    if (err) {
      console.error("Error generating QR code:", err);
      res.status(500).send("Error generating QR code");
    } else {
      // Return the QR code image
      res.send(`<img src="${qrUrl}" alt="QR Code"/>`);
    }
  });
});

router.get("/:shortcode", getExternalLink);

module.exports = router;
```

Utility Functions

The misc functions that plays important role and appears to have abstract form but have a significant role in whole project.

generateRandomCode

This function generates a **6-digit random code** using **alphanumeric** characters only

generateShortCode

This function calls **generateRandomCode** to get the random short code and makes sure that it is not already in database and it prefers shortcode given by user if it is unique.

```
    const { Link } = require('../models');

    // Function to generate a random string of length 6
    function generateRandomCode(length = 6) {
        const characters =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
        let result = '';
        const charactersLength = characters.length;
        for (let i = 0; i < length; i++) {
            result += characters.charAt(Math.floor(Math.random() *
charactersLength));
        }
        return result;
    }
    async function generateShortCode(preferredCode) {
        preferredCode = preferredCode.trim();
        let uniqueCode = preferredCode.length > 0 ? preferredCode :
generateRandomCode();

        while (true) {
            const existingLink = await Link.findOne({ where: { shortenedUrl:
uniqueCode } });

            if (!existingLink) {
                return { success: true, code: uniqueCode }; // Unique code
found or generated
            } else {
                if (uniqueCode === preferredCode && preferredCode.length > 0)
{
                    return { success: false, message: "Preferred Short Code is
already in use." }; // Preferred code is not unique
                }
                uniqueCode = generateRandomCode(); // Generate a new random
code
            }
        }
    }


    module.exports = generateShortCode;
```

The Main Server Script

This is the starter script that we are using to start the server.

This is using express as framework for routing and managing APIs. Using app.use we can use the routes that was created earlier.

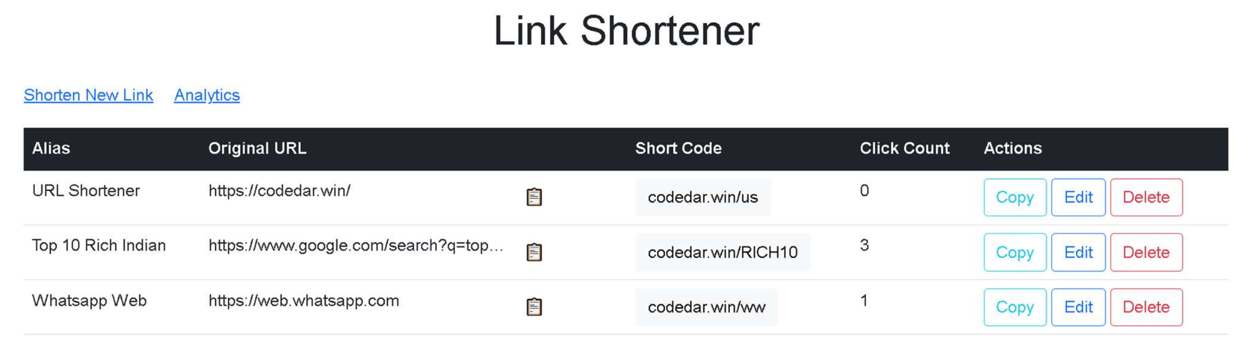
The files that are need to be served to clients accessing the home pages are in public directory. We are using **user-agent** to collect more data from client for analytics

PORT 3000 is being used during the execution of script if no port is explicitly defined.

```
const express = require("express");
const app = express();
const useragent = require("express-useragent");

app.use(express.json());
app.use(express.static("public")); // Serve static files
app.use(useragent.express());

const linkRoutes = require("./routes/linkRoutes");
const clickRoutes = require("./routes/clickRoutes");
const redirectRoutes = require("./routes/redirectRoutes");

app.use(express.json());

app.use("/api/links", linkRoutes);
app.use("/api/clicks", clickRoutes);
app.use("/", redirectRoutes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Client-Side Design:

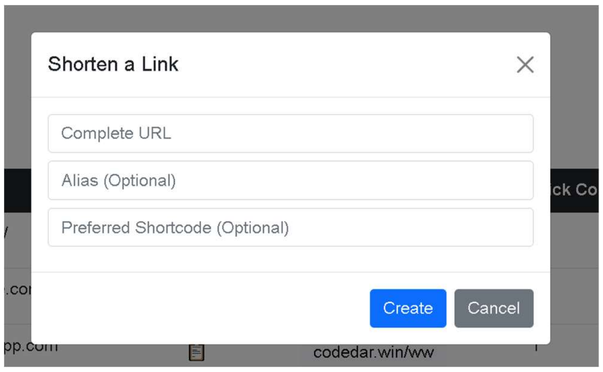
Previews



Link Shortener

Shorten New Link Analytics

Alias	Original URL	Short Code	Click Count	Actions
URL Shortener	https://codedar.win/	codedar.win/us	0	Copy Edit Delete
Top 10 Rich Indian	https://www.google.com/search?q=top...	codedar.win/RICH10	3	Copy Edit Delete
Whatsapp Web	https://web.whatsapp.com	codedar.win/ww	1	Copy Edit Delete



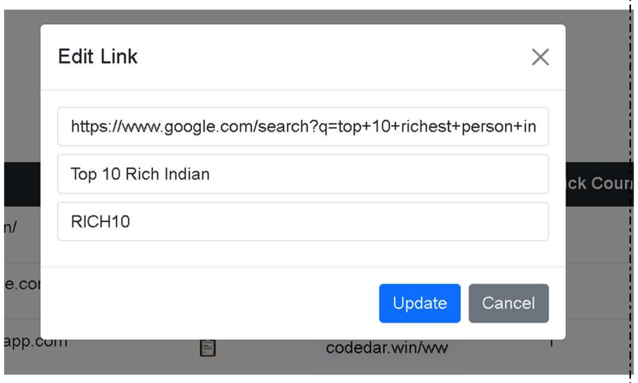
Shorten a Link

Complete URL

Alias (Optional)

Preferred Shortcode (Optional)

Create Cancel



Edit Link

https://www.google.com/search?q=top+10+richest+person+in

Top 10 Rich Indian

RICH10

Update Cancel

index.html

The index.html which is being served as our **Homepage** for now has user experience that allows us to do all the essential operations on the links.

We are planning on having this page authenticated.

This page allows us to create links, edit, delete and update exiting links allows a user-friendly UI to copy links access them and provides the number of times the links are clicked.

This index.html page is backed by the app.js for client side which helps the page to call proper API, provide more control over the interface, rendering models on requirements, handling exceptions and much more.

Client Side Script

app.js
This script has REST function calls to **get, post, delete and put** the data when needed in index page. When this script is loaded it calls to get the details of all the links, adds the details in a row and add the rows in the table to be viewed by the user.

Handles callbacks for Modals for creating, editing and deletion.
Shows alert on actions using the **toastify.js** library.

```
window.onload = function () {
  fetchLinks();
};

async function fetchLinks() {
  const response = await fetch("/api/links");
  const links = await response.json();
  const tbody = document
    .getElementById("linksTable")
    .getElementsByTagName("tbody")[0];
  tbody.innerHTML = ""; // Clear current content
  links.forEach((link) => {
    let row = tbody.insertRow();
    let aliasCell = row.insertCell(0); // Cell for the alias
    let originalCell = row.insertCell(1);
    let shortCell = row.insertCell(2);
    let clickCount = row.insertCell(3);
    let editCell = row.insertCell(4);
    const orgDiv = `<div style="display:flex;"><div style="width:300px;white-
space: nowrap; overflow: hidden; text-overflow:
ellipsis;">${link.originalUrl}</div> <button class="btn"
onclick="copyContent('${link.originalUrl}');">📋</button></div>`;
    originalCell.innerHTML = orgDiv;
    shortCell.innerHTML = `<button class="btn btn-light"
onclick="window.open('${link.shortenedUrl}')">${window.location.host}/${link.s
hortenedUrl}</button>`;
    aliasCell.textContent = link.alias; // Populate the alias cell
    clickCount.textContent = link.clickCount;
    const breakDiv = ` `;
    const editButtom = `<button class="btn btn-outline-primary"
onclick="createEditForm('${link.id}', '${link.originalUrl}',
'${link.alias}','${link.shortenedUrl}')">Edit</button>`;
    const deleteButton = `<button class="btn btn-outline-danger"
onclick="createDeleteForm('${link.id}' ,'${link.originalUrl}',
'${link.alias}','${link.shortenedUrl}')">Delete</button>`;
```

```
    const copyToClipBoard = `<button class="btn btn-outline-info"
onclick="copyContent('${window.location.host}/${link.shortenedUrl}')">Copy</bu
tton>`;
    editCell.innerHTML =
      copyToClipBoard + breakDiv + editButtom + breakDiv + deleteButton;
  });
}

const copyContent = async (content) => {
  try {
    await navigator.clipboard.writeText(content);
    showMessage("Content Copied to Clipboard");
  } catch (err) {
    console.error("Failed to copy: ", err);
  }
};

function createDeleteForm(id, originalUrl, alias, shortCode) {
  // Update input fields
  document.getElementById("deleteOriginalUrl").innerHTML = originalUrl;
  document.getElementById("deleteAlias").innerHTML = alias ? alias : "";
  document.getElementById("deleteShortCode").innerHTML = shortCode;
  window.currentLinkId = id; // Store current editing ID
  // Show the modal using Bootstrap's modal method
  var deleteModal = new
bootstrap.Modal(document.getElementById("deleteForm"));
  deleteModal.show();
}

async function deleteLink() {
  const id = window.currentLinkId;

  const response = await fetch(`/api/links/${id}`, {
    method: "DELETE",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ id }),
  });
  if (!response.ok) {
    const errorResult = await response.json();
    alert(`Error: ${errorResult.error}`);
  } else {
    const result = await response.json();
    showMessage("Link deleted");
    closeDeleteForm();
    fetchLinks(); // Refresh list
  }
}
```

```javascript
function closeDeleteForm() {
  var deleteModal = bootstrap.Modal.getInstance(
    document.getElementById("deleteForm")
  );
  deleteModal.hide();
}

function createEditForm(id, originalUrl, alias, shortCode) {
  // Update input fields
  document.getElementById("editOriginalUrl").value = originalUrl;
  document.getElementById("editAlias").value = alias ? alias : "";
  document.getElementById("editShortCode").value = shortCode;
  window.currentLinkId = id; // Store current editing ID
  // Show the modal using Bootstrap's modal method
  var editModal = new bootstrap.Modal(document.getElementById("editForm"));
  editModal.show();
}

async function updateLink() {
  const id = window.currentLinkId;
  const originalUrl = document.getElementById("editOriginalUrl").value;
  const alias = document.getElementById("editAlias").value;
  const shortenedUrl = document.getElementById("editShortCode").value;

  const response = await fetch(`/api/links/${id}`, {
    method: "PUT",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ originalUrl, alias, shortenedUrl }),
  });

  if (!response.ok) {
    const errorResult = await response.json();
    alert(`Error: ${errorResult.error}`);
  } else {
    const result = await response.json();
    alert("Link updated");
    closeEditForm();
    fetchLinks(); // Refresh list
  }
}

function closeEditForm() {
  var editModal = bootstrap.Modal.getInstance(
    document.getElementById("editForm")
  );
  editModal.hide();
}
```

```
function createLinkForm() {
  // Show the modal using Bootstrap's modal method
  var editModal = new bootstrap.Modal(document.getElementById("createForm"));
  editModal.show();
}

async function createLink() {
  const originalUrl = document.getElementById("newOriginalUrl").value;
  const alias = document.getElementById("newAlias").value;
  const shortenedUrl = document.getElementById("newShortCode").value;

  const response = await fetch(`/api/links/`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ originalUrl, alias, shortenedUrl }),
  });

  if (!response.ok) {
    const errorResult = await response.json();
    alert(`Error: ${errorResult.error}`);
  } else {
    const result = await response.json();
    alert(`Shortened URL: ${result.shortenedUrl}`);
    document.getElementById("newOriginalUrl").value = "";
    document.getElementById("newAlias").value = "";
    document.getElementById("newShortCode").value = "";
    closeCreateForm();
  }

  fetchLinks(); // Refresh list
}

function closeCreateForm() {
  var createModal = bootstrap.Modal.getInstance(
    document.getElementById("createForm")
  );
  createModal.hide();
}

function showMessage(message) {
  Toastify({
    text: message,
    duration: 3000,
    gravity: "top", // `top` or `bottom`
    position: "right", // `left`, `center` or `right`
    stopOnFocus: false, // Prevents dismissing of toast on hover
    style: {
      background: "#32CD32",
    },
    onClick: function () {}, // Callback after click
  }).showToast();
}
```

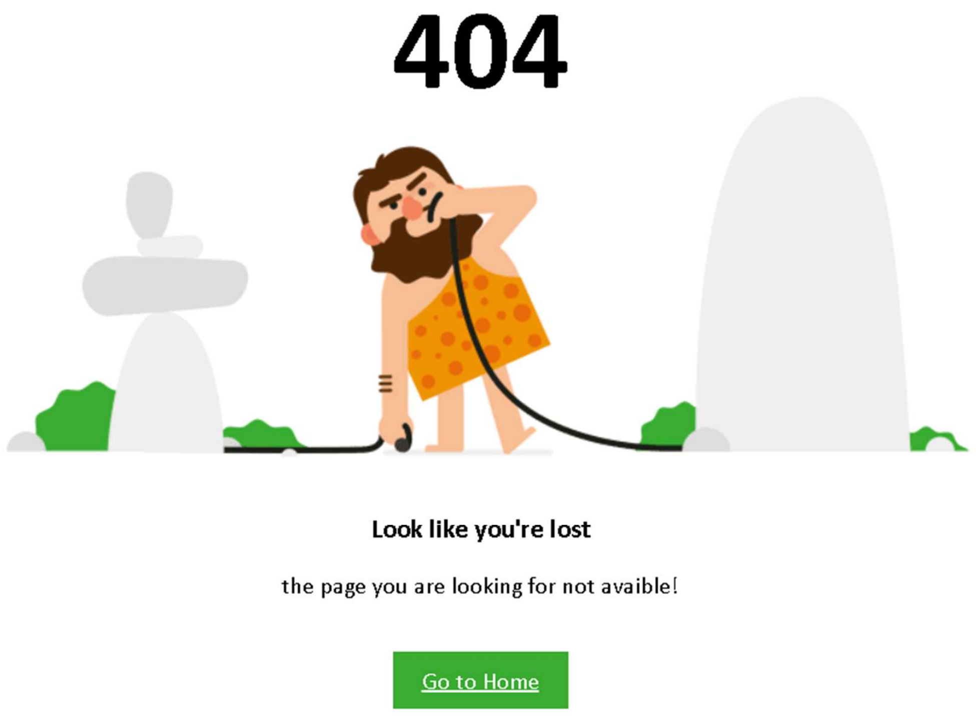
QR Code Generated by Our Tool vs QR of Actual URL

Here we’ve a quick comparison of having a short QR and a Short URL which can be customized.

URL : https://codedar.win/RICH10

Actual URL:
https://www.google.com/search?q=top+10+richest+person+in+india&oq=top+10+richest+person+in+india&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIGCAEQLhhA0gEIODAwMGowajmoAgCwAgE&sourceid=chrome&ie=UTF-8

And End of Our File with something that every project needs
A 404 Page



404

Look like you're lost

the page you are looking for not avaible!

Thank You !