**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.
* **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.

**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.

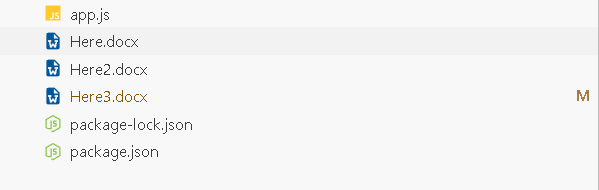
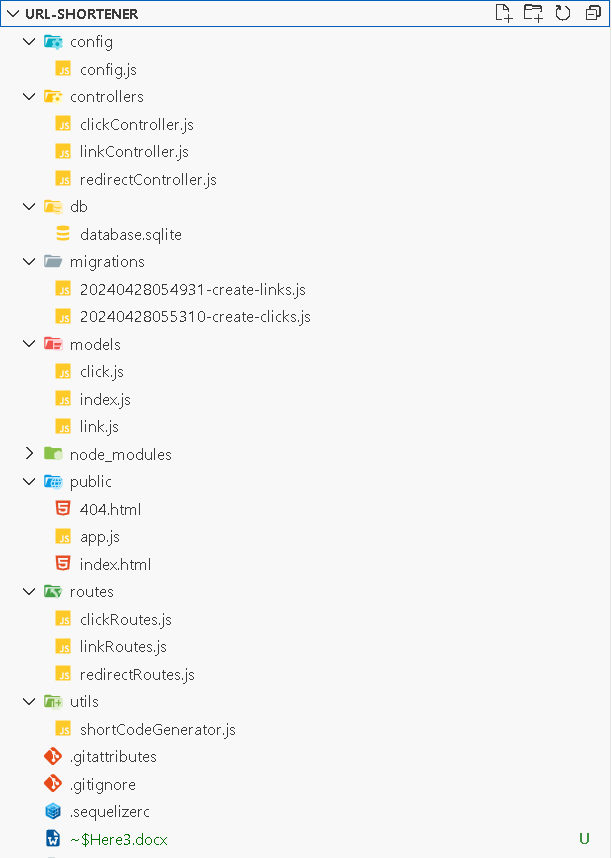
**Workflow Overview:**

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**



**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.

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.

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]

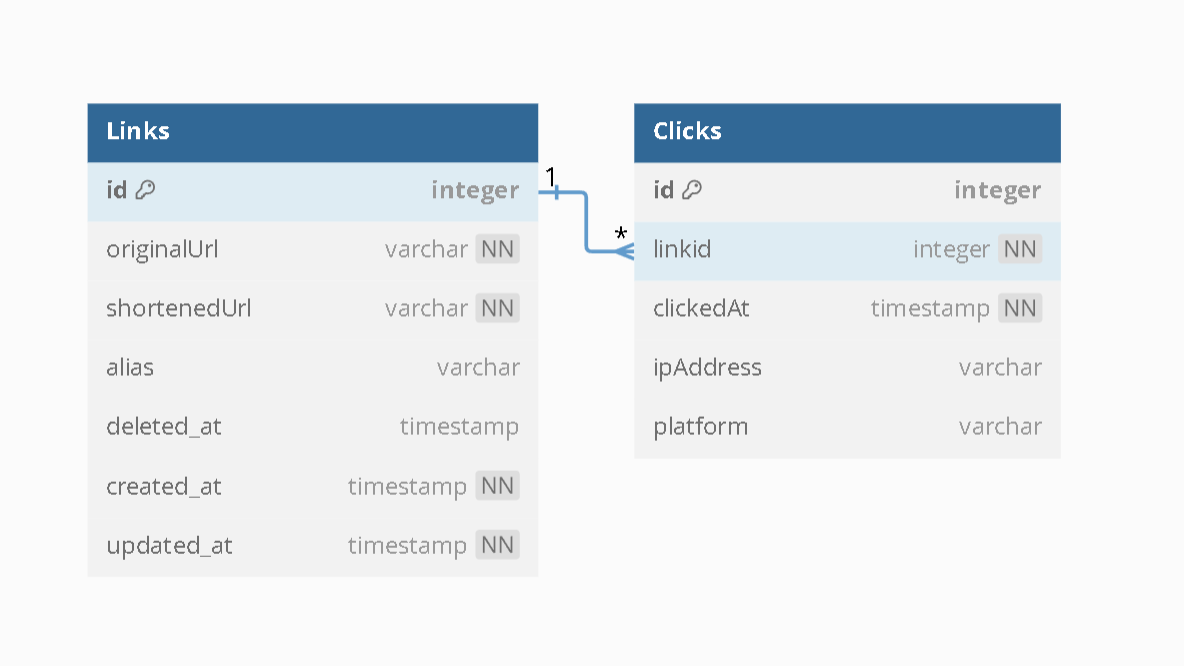
  clickedAt timestamp [not null]

  ipAddress varchar [null]

  platform varchar [null]

}

Ref: Links.id < Clicks.linkid

Here’s the visualization of our db diagram

**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.

**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'

  }

};

**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.

// 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.

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 to get the direct link function to get the direct link from the database using index of shortcode that is being received in request object

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**

const express = require("express");

const router = express.Router();

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

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

module.exports = router;

**Redirect Routes**

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 provides you a random code that is not already in database and it prefers your given shortcode if it doesn’t exist it provides you that

    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 in which the server will be opened on port 3000 if accessed in a development environment and accordingly when needed in production.

This provides api routes to the app from routes files, Uses user-agent to collect more data from client for analytics

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**

**index.html**

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>URL Shortener</title>

    <link

      href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"

      rel="stylesheet"

    />

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>

    <link

      rel="stylesheet"

      type="text/css"

      href="https://cdn.jsdelivr.net/npm/toastify-js/src/toastify.min.css"

    />

    <title>Link Shortener</title>

    <style>

      h1 {

        text-align: center;

        padding: 20px;

      }

      body {

        margin: 0;

        padding: 20px;

        font-family: Arial, sans-serif; /\* Ensures font consistency \*/

      }

      table {

        width: 100%; /\* Makes the table use the full width of the viewport \*/

        border-collapse: collapse; /\* Removes space between borders \*/

      }

      th,

      td {

        white-space: nowrap;

        overflow: hidden;

        text-overflow: ellipsis;

        max-width: 400px;

        min-width: 100px;

        text-align: left;

        padding: 8px; /\* Adds padding to table cells \*/

        border-bottom: 1px solid #ddd; /\* Adds a light line between rows \*/

      }

      th {

        background-color: #f4f4f4; /\* Light grey background for headers \*/

      }

    </style>

  </head>

  <body>

    <h1>Link Shortener</h1>

    <div style="display: flex; align-items: center; justify-content: start">

      <a href="javascript:void(0);" onclick="createLinkForm()"

        >Shorten New Link</a

      >

      <div style="width: 20px"></div>

      <a href="javascript:void(0);">Analytics</a>

      <div style="width: 20px"></div>

    </div>

    <div style="height: 20px"></div>

    <div

      id="createForm"

      class="modal"

      tabindex="-1"

      role="dialog"

      style="display: none"

    >

      <div class="modal-dialog" role="document">

        <div class="modal-content">

          <div class="modal-header">

            <h5 class="modal-title">Shorten a Link</h5>

            <button

              type="button"

              class="btn-close"

              onclick="closeCreateForm()"

            ></button>

          </div>

          <div class="modal-body">

            <input

              type="text"

              id="newOriginalUrl"

              class="form-control mb-2"

              placeholder="Complete URL"

              required

            />

            <input

              type="text"

              id="newAlias"

              class="form-control mb-2"

              placeholder="Alias (Optional)"

            />

            <input

              type="text"

              id="newShortCode"

              class="form-control mb-2"

              placeholder="Preferred Shortcode (Optional)"

              required

            />

          </div>

          <div class="modal-footer">

            <button onclick="createLink()" class="btn btn-primary">

              Create

            </button>

            <button onclick="closeCreateForm()" class="btn btn-secondary">

              Cancel

            </button>

          </div>

        </div>

      </div>

    </div>

    <table id="linksTable" class="table">

      <thead class="table-dark">

        <tr>

          <th>Alias</th>

          <!-- New column for Alias -->

          <th>Original URL</th>

          <th>Short Code</th>

          <th>Click Count</th>

          <th>Actions</th>

        </tr>

      </thead>

      <tbody>

        <!-- Rows will be added here dynamically -->

      </tbody>

    </table>

    <div

      id="editForm"

      class="modal"

      tabindex="-1"

      role="dialog"

      style="display: none"

    >

      <div class="modal-dialog" role="document">

        <div class="modal-content">

          <div class="modal-header">

            <h5 class="modal-title">Edit Link</h5>

            <button

              type="button"

              class="btn-close"

              onclick="closeEditForm()"

            ></button>

          </div>

          <div class="modal-body">

            <input

              type="text"

              id="editOriginalUrl"

              class="form-control mb-2"

              placeholder="Edit URL"

              required

            />

            <input

              type="text"

              id="editAlias"

              class="form-control mb-2"

              placeholder="Edit Alias"

            />

            <input

              type="text"

              id="editShortCode"

              class="form-control mb-2"

              placeholder="Edit Shortcode (Optional)"

              required

            />

          </div>

          <div class="modal-footer">

            <button onclick="updateLink()" class="btn btn-primary">

              Update

            </button>

            <button onclick="closeEditForm()" class="btn btn-secondary">

              Cancel

            </button>

          </div>

        </div>

      </div>

    </div>

    <div

      id="deleteForm"

      class="modal"

      tabindex="-1"

      role="dialog"

      style="display: none"

    >

      <div class="modal-dialog" role="document">

        <div class="modal-content">

          <div class="modal-header">

            <h5 class="modal-title">Edit Link</h5>

            <button

              type="button"

              class="btn-close"

              onclick="closeDeleteForm()"

            ></button>

          </div>

          <div class="modal-body">

            <p type="text" id="deleteOriginalUrl" class="form-control mb-2"></p>

            <p type="text" id="deleteAlias" class="form-control mb-2"></p>

            <p type="text" id="deleteShortCode" class="form-control mb-2"></p>

          </div>

          <div class="modal-footer">

            <button onclick="deleteLink()" class="btn btn-primary">

              Delete

            </button>

            <button onclick="closeDeleteForm()" class="btn btn-secondary">

              Cancel

            </button>

          </div>

        </div>

      </div>

    </div>

    <script src="app.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/toastify-js"></script>

  </body>

</html>

**Client Side Script**

**app.js**

This script has REST function calls to get delete put the data when needed in index page

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.shortenedUrl}</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</button>`;

    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

  }

}

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",

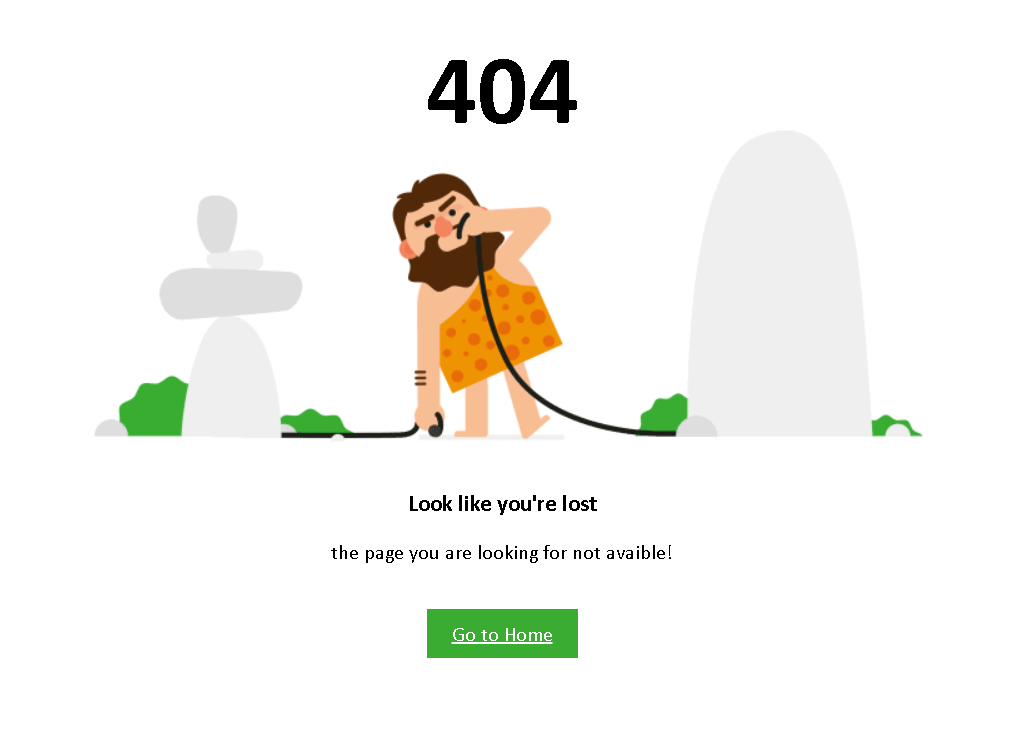
    },

    onClick: function () {}, // Callback after click

  }).showToast();

}

**And End of Our File with something that every project needs**

**A 404 Page   
**