**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.

Lets start with the workflow we are going to adapt in this project

* 1. We are using github for version control to make the developments at much ease and enabling team to collaborate better.
  2. The main environment we are using is Node.js (v 18.04.0)

In NodeJs we are using mentioned packages which we will talk about in details while discussing the workflow:

* + 1. dotenv
    2. express
    3. express-useragent
    4. request-ip
    5. sequelize
    6. sqlite3
  1. We are using AWS hosting for the project, the instance we are using is t2.micro.
  2. And for our domain is managed by Cloudflare.

We are using **Agile Project Management Methodology**

We started with base functionalities implemented them designed more when required.

This helped us in managing unit tests for a functionalites better than other approach.

Instead of overburdening ourselves from upcoming requirements we focused on current ones and implemented others as we are getting one by one.

**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]

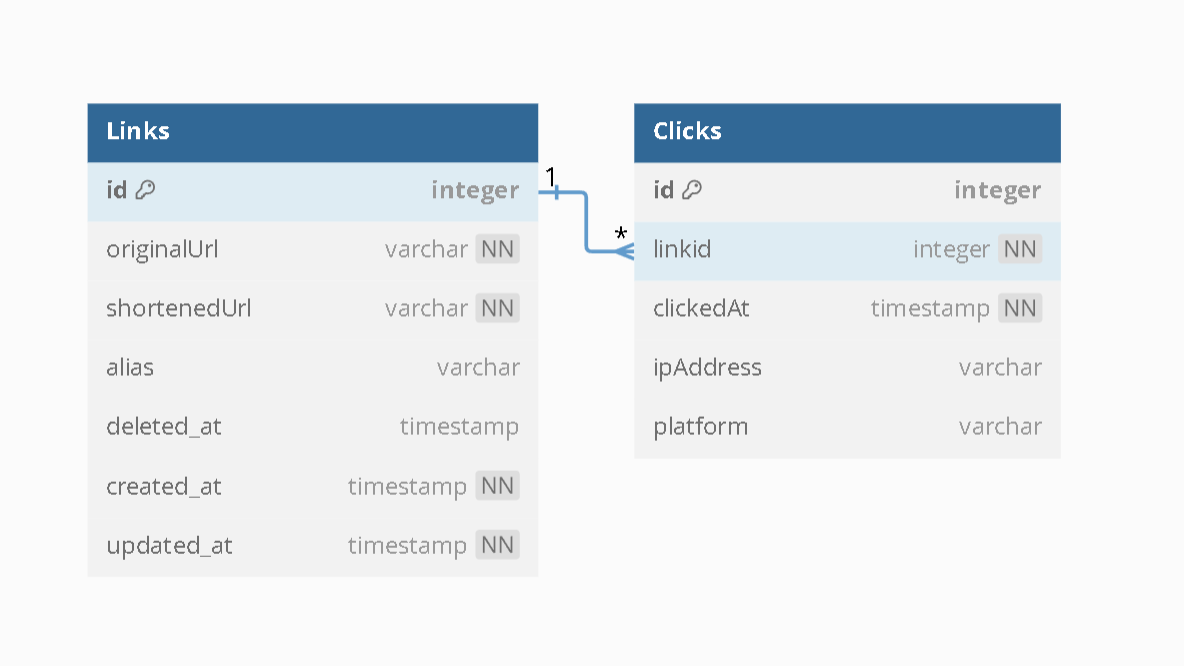
  ipAddress varchar [null]

  platform varchar [null]

}

Ref: Links.id < Clicks.linkid

Here’s the visualization of our db diagram



A little Summary for our Links Table

id -> unique id of the links also the pk

originalUrl -> our original long url which we want to shorten this cannot be null.

shortenedUrl -> the suffix for our short url (cannot be null)

alias -> the nickname we want to give to urls (it is optional and can be null)

deleted\_at -> this refers to the date the record was deleted i.e. deletion date of link

created\_at -> creation date of record i.e. creation date of link

updated\_at -> last updation time of the link

Clicks Table

id -> unique id for each click also the pk

linkId -> the id of link to which the click is associated with

clickedAt -> at which timestamp the link was clicked at

ipAddress -> ip address of the visitor

platform -> the platform the user is using while accessing the link

The relationship defined between both table is **Link has Many Clicks** (One to Many) which we will be seeing our data models and in our migration scripts too.

**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**

"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**

"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**

// 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**

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**

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**

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**

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;