Submitted By:
Guramritpal Singh Saggu 2017IMT-*036*
Ojaswa Sharma 2017IMT-*102*

Project Report

Title: Web Service to shorten URLs, like goo.gl and bit.ly

Introduction

URL shortener generates shortened links for all the websites on the internet which have huge links and are often inconvenient to share. Your shortened URLs can be used in publications, advertisements, blogs, forums, e-mails, instant messages, and other locations. Although the primary objective is to provide convenience while assuring security, sometimes shortened URLs are also necessary while dealing with word limits, character filters etc.

The service provides an alpha-numeric short URL which redirects the user to the original link while maintaining efficiency. The following are some features of the developed service:

Easy

URL Shortener is easy and fast, enter the long link to get your shortened link

Shortened

Use any link, no matter what size, URL Shortener always shortens

Secure

It is fast and secure, our service have HTTPS protocol and data encryption

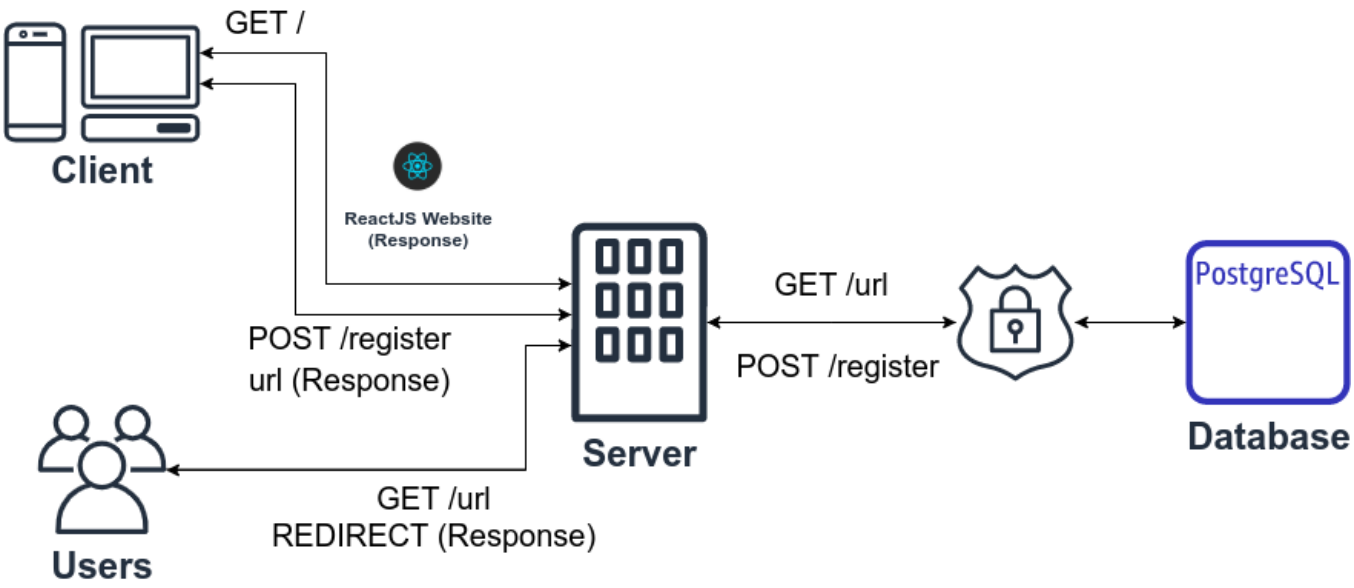
Devices

Compatible with smartphones, tablets and desktop

Objective: To provide an easy, secure, responsive and multi-platform solution for shortening web URLs (Uniform Resource Locator) used to access the World Wide Web (usually referred to as, *The Internet* or *The Web*).

Algorithms & Methodology

Architecture:



The above drawn methodology was used in deploying the web service. To elaborate, following is the description of various entities:

1. *Client*

   Clients are users which generate a short URL from a long URL for their personal or professional usage.

2. *Users*

   Users are those who visit the generated short URL, which leads them to the original long URL via redirection.

3. *Server*

   It is the self-hosted express server which facilitates all types of actions through it. It is solely responsible for establishing connection between various entities, managing connection state and providing/exchanging information. It is a central entity through which all other entities must pass in order to function correctly.

4. *Database*

   Database is a storage entity which stores a map of {long URL, short URL}. The storage is persistent and the current TTL (Time to Live) for the data is infinite, that means the data will live for as long as the server is up on the internet.

The web system connects the above specified entities in a way to complement each other and thus fulfill the objective of this system. The Clients & Users interact with the server using HTTP(s) Protocols using REST API, whose endpoints can be defined as follows:

1. *GET /* : Serves the URL shortener front-end which can be accessed on https://url.ojaswa.com.

2. POST /register: Generates a new short URL for the provided long URL, saves both of them into the database and returns the newly generated short URL. Further details on the generation are specified after this sub-section.

3. GET /{url}: Here, the url refers to the short URL generated by any client. When users access this endpoint (or URL), the database is accessed for corresponding long URL and the user is redirected to the respective site. In case, the entry does not exist in the database, the user is responded with a 404 error.

Generation of short URLs:

The generation of short URLs is a significant process. It is necessary to ensure that the short URLs (or aliases) are unique and thus maintain the integrity of the URL map in the persistent storage or database. For this purpose, we use random generation of alpha-numeric characters upto length 6 i.e. for each place, there is a possibility of 62 characters:

1. 26 Alphabets (A-Z),
2. 26 Alphabets (a-z),
3. 10 numbers (0-9)

This takes the overall possibilities to 62^6 which is equal to 56,800,235,584 which is nearly equal to 57 Billion possible unique combinations, which is fairly tough to achieve. The uniqueness however, can be further imposed using UNIQUE or PRIMARY key in the SQL database, which will not allow any duplicate values of short URL, ultimately maintaining the integrity of URLs.

Implementation Language & Source Code

1. The Frontend

The front-end is implemented using **ReactJs**, which is a JavaScript-based web development library. The code is hence written in **JavaScript** using the latest **ECMAScript** specifications. The following is a snapshot of front-end dependencies:

```
"react": "^16.13.1",
"react-dom": "^16.13.1",
"react-scripts": "3.4.1",
"tachyons": "^4.12.0"
```

react & react-dom: React libraries required to sustain development

react-scripts: Provides an additional feature of aliases to scripts and is usually used to perform actions during the course of development. Following are the react scripts defined for this project:

```
"scripts": {
  "start": "nodemon server.js",
  "front": "react-scripts start",
  "build": "react-scripts build"
},
```

tachyons: Tachyons is a styling library and provides predefined short-hand class names to apply CSS on HTML elements

The complete web design is broken into different Components and rendered together conditionally. Here is an example of a Component *Header.js*:

Header.js

```
import React from 'react';

const Header = ({onInput, onClick, onKey}) =>{
    return(
        <header className="bg-gold sans-serif">
            <div className="mw9 center pa3 pt5-ns ph7-l pb1">

              <h3 className="f2 f1-m f-headline-l measure-narrow lh-title mv0">
                <span className="bg-dark-gray lh-copy white pa1 tracked-tight">
                  Cut the Crap
                </span>
              </h3>
              <h4 className="f3 dark-gray fw1 georgia i">Shorten those long URLs for convenience.</h4>
            </div>
        </header>
    );
}
export default Header;
```

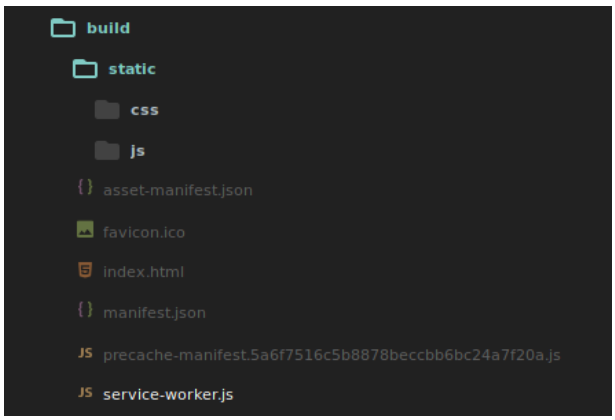
As we can see in the above code, a Component is similar to a Function which takes some input and returns the HTML equivalent JSX (within JavaScript with React in scope). When the Component is rendered, the output **JSX** is compiled into an HTML element by React and thus displaying the element on the webpage.

Similar to *Header*, various other components are defined in the project structure, such as *Result, Form and Footer*.

Here is an example of how multiple Components are rendered together (from App.js):

```
render() {
  return (
    <div className="App w-100 bg-dark-gray">
      <Header className= " " />
      <Form className=""
        onClick={this.onClick}
        onKey ={this.onKey}
        onInput = {this.onInput} />

    {(this.state.result_status)?
      <Result className="" result={this.state.result} /> : <div />
  }
      <Footer />
    </div>
  );
}
```

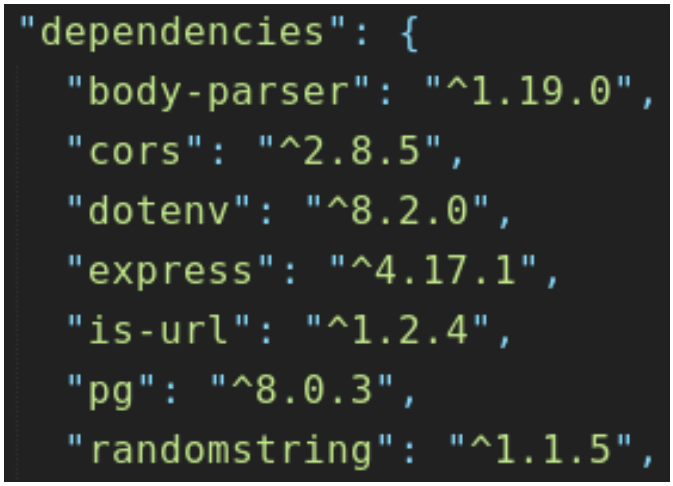
After the development is complete, the React JavaScript files are built into normal JS, **CSS** and HTML files which are familiar web browsers. The following is the structure of the public folder after build:



The complete source code is Open Source and can be accessed on GitHub here: https://github.com/ojaswa1942/url-shortener.

2. The Server

The server is implemented using **Express**, which is a minimal and flexible Node.js web application framework that provides a robust set of features for web & mobile applications and APIs. **Node.js** is a **JavaScript** runtime built on Chrome's V8 JavaScript engine. The following is a screenshot of the server dependencies:



```
"dependencies": {
  "body-parser": "^1.19.0",
  "cors": "^2.8.5",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "is-url": "^1.2.4",
  "pg": "^8.0.3",
  "randomstring": "^1.1.5",
```

body-parser: Parses incoming request bodies

cors: Configures the Access-Control-Allow-Origin CORS header and allows local development

dotenv: Allows convenient access to environment variables

is-url: Used to validate a URL (Simple RegEx)

pg: Used to establish connection with PostgreSQL Database

randomstring: Used to generate short URL

The server comprises different functions which run when the corresponding endpoint is hit. As specified in the previous section, we have 3 endpoints available for this application:

```
app.get('/', (req,res)=>{res.sendFile('index.html');});
app.post('/register', (req, res) => {handleRegister(req, res, db)});
app.get('/*', (req,res)=>{handleFetchRequest(req, res, db)});
```

Following is a snapshot of *server.js:*

```
const express=require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const { Client } = require('pg');
const { handleRegister } = require('./controllers/register');
const { handleFetchRequest } = require('./controllers/fetch');
require('dotenv').config();

const db = new Client();
// Takes default config from ENV
db.connect(error => {
    if(error)
        console.error(`Error connecting to database`);
});

db.on('notice', msg => console.warn('notice:', msg))

const app=express();

app.use(cors());
app.use(bodyParser.json());

app.use(express.static(__dirname + '/build'));

app.get('/', (req,res)=>{res.sendFile('index.html');});
app.post('/register', (req, res) => {handleRegister(req, res, db)});
app.get('/*', (req,res)=>{handleFetchRequest(req, res, db)});

const port = process.env.PORT || 3007;

app.listen(port, ()=>{
    console.log(`We are on on port ${port}!`);
})
```

The above file (server.js) initiates the server on a port, establishes connection to the database, declares various middlewares and defines all the endpoints or controllers.

Following is the screenshot of **POST /register** controller:

```
const randomstring = require("randomstring");
const isUrl = require("is-url");

const handleRegister = ( req, res, db ) => {
    const url = 'https://url.ojaswa.com/'
    const {input} = req.body;
    if(!isUrl(input)){
        return res.status(400).json({result_status: false});
    }
    console.log('hey');
    const QUERY_TEXT = `INSERT INTO entries(long_url, short_url, date) VALUES($1, $2, $3) RETURNING short_url`

    return db.query(QUERY_TEXT, [ input, randomstring.generate(6), new Date() ])
    .then(({rows}) => {
        res.status(200).json({
            result_status: true,
            result: url+rows[0].short_url
        })
    })
    .catch(err=> res.status(400).json(err))
}

module.exports = {
    handleRegister
};
```

Following is the screenshot of **GET /*** controller:

```
const handleFetchRequest = ( req, res, db ) => {
    const request_route = req.url;
    const request_url = request_route.substr(1);

    const QUERY_TEXT = `SELECT * FROM ENTRIES WHERE short_url = $1`;
    return db.query(QUERY_TEXT, [request_url])
    .then(({ rows }) => {
        if(rows.length)
            res.redirect(rows[0].long_url);
        else
            res.status(404).json(`Seems you're lost`);
    })
    .catch(err => res.status(404).json('Something is wrong'));
}

module.exports = {
    handleFetchRequest
};
```

The above code also specifies various queries used to exchange data with the server.

3. **The Database**

As specified in various sections above, **PostgreSQL** server is used to maintain a map (table) between long URL and short URL. The following SQL query is used to generate the database structure required for the usage of this application:

```
url=# create table entries (long_url varchar(2000) not null, short_url char(6) not null, date date
not null default current_date);
```

The connection is established using **environment variables**, which are declared in the *.env* file in the project root. Following is an example file:

```
PORT=3000
PGHOST='localhost'
PGUSER=process.env.USER
PGDATABASE=process.env.USER
PGPASSWORD=null
PGPORT=5432
```

All the queries used to communicate with the database are specified in the controllers in the previous section.

4. **The Ops and Deployment**

The complete infrastructure is **self-hosted** on **AWS Lightsail Instance**. **Nginx** is used for Web Server and directs all requests towards the URL: https://url.ojaswa.com to the server using **Reverse Proxy**. The following is the server-block:

```
server {
        server_name url.ojaswa.com;


        location / {
                proxy_pass http://127.0.0.1:3007/;
        }

        listen 443 ssl; # managed by Certbot
}
```

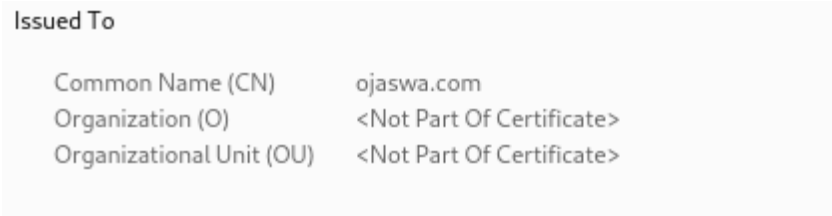
Similarly, the database also runs on the server which is then accessed when the need arises. Here is a snapshot of the ***postgresql.service*** running on web instance:

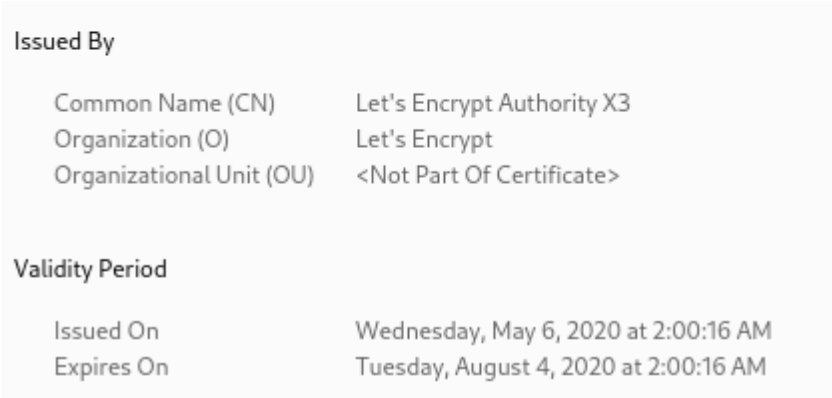
```
ubuntu@ip-172-26-12-163:~$ sudo systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
   Active: active (exited) since Thu 2019-08-22 06:13:56 UTC; 8 months 20 days ago
 Main PID: 9882 (code=exited, status=0/SUCCESS)
    Tasks: 0 (limit: 1152)
   CGroup: /system.slice/postgresql.service

Aug 22 06:13:56 ip-172-26-12-163 systemd[1]: Starting PostgreSQL RDBMS...
Aug 22 06:13:56 ip-172-26-12-163 systemd[1]: Started PostgreSQL RDBMS.
```

For the URL to work as required, **DNS** records (A-Record) for https://url.ojaswa.com points to the server. The following is the output for DNS lookup:

```
09:31:06 ojaswa@fruit-salad ~ → dig url.ojaswa.com

; <<>> DiG 9.16.0 <<>> url.ojaswa.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62379
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1452
;; QUESTION SECTION:
;url.ojaswa.com.                        IN      A

;; ANSWER SECTION:
url.ojaswa.com.         300     IN      A       13.234.121.10

;; Query time: 313 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Tue May 12 21:31:14 IST 2020
;; MSG SIZE  rcvd: 73
```

For secure connection, a self-signed **SSL** certificate to facilitate HTTPS is also issued using ***letsencrypt***. The following are the certificate details:





Finally, to facilitate a better development experience, **Continuous Deployment** is set up using **GitHub Actions** to replicate the deployment steps automatically on each commit in order for regular deployment and **automation**. The following is the *configuration yml file*:

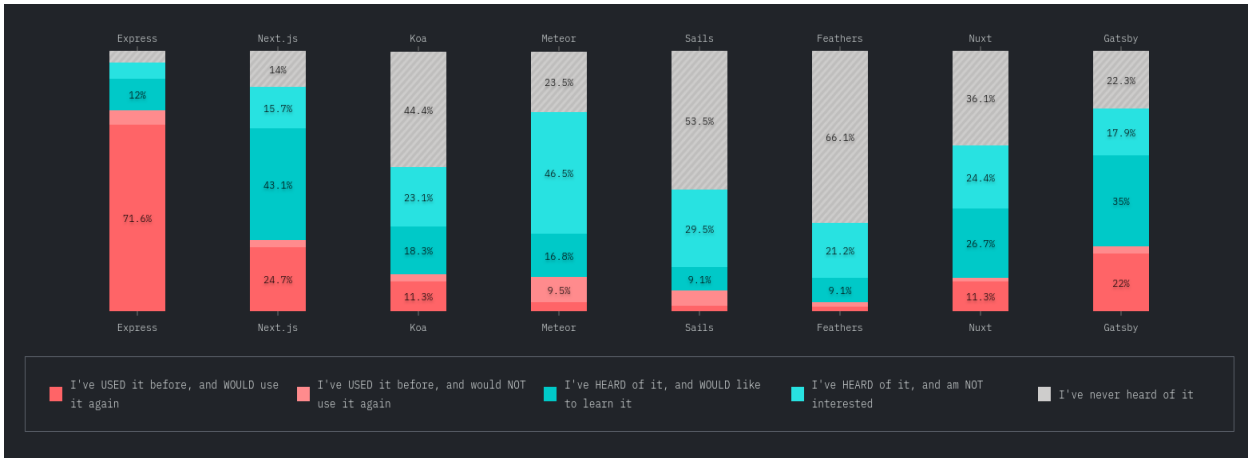
```yaml
name: CI

on:
  push:
    branches: [ master ]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - uses: actions/setup-node@master
      with:
        node-version: 12.16

    - name: Save key
      env:
        ACCESS_KEY: ${{ secrets.OPS_KEY_ENC }}
      run: |
        echo "$ACCESS_KEY" | base64 -d > $HOME/KEY
        chmod 400 $HOME/KEY

    - name: Install dependencies
      run: yarn install

    - name: Build
      run: yarn build

    - name: Save environment vars
      env:
        ENV_FILE: ${{ secrets.ENV }}
      run: echo "$ENV_FILE" | base64 -d > $HOME/.env

    - name: Deploy to server
      run: |
        scp -r -i "$HOME/KEY" -o StrictHostKeyChecking=no ./build ops@url.ojaswa.com:/var/www/url/
        scp -r -i "$HOME/KEY" -o StrictHostKeyChecking=no ./controllers ops@url.ojaswa.com:/var/www/url/
        scp -r -i "$HOME/KEY" -o StrictHostKeyChecking=no $HOME/.env ops@url.ojaswa.com:/var/www/url/
        scp -r -i "$HOME/KEY" -o StrictHostKeyChecking=no ./server.js ops@url.ojaswa.com:/var/www/url/
```

As you can see, the files are deployed to the server using **SCP over SSH**. The access is controlled using **Asymmetric Encryption** (Public-Private Key Infrastructure).

Discussion

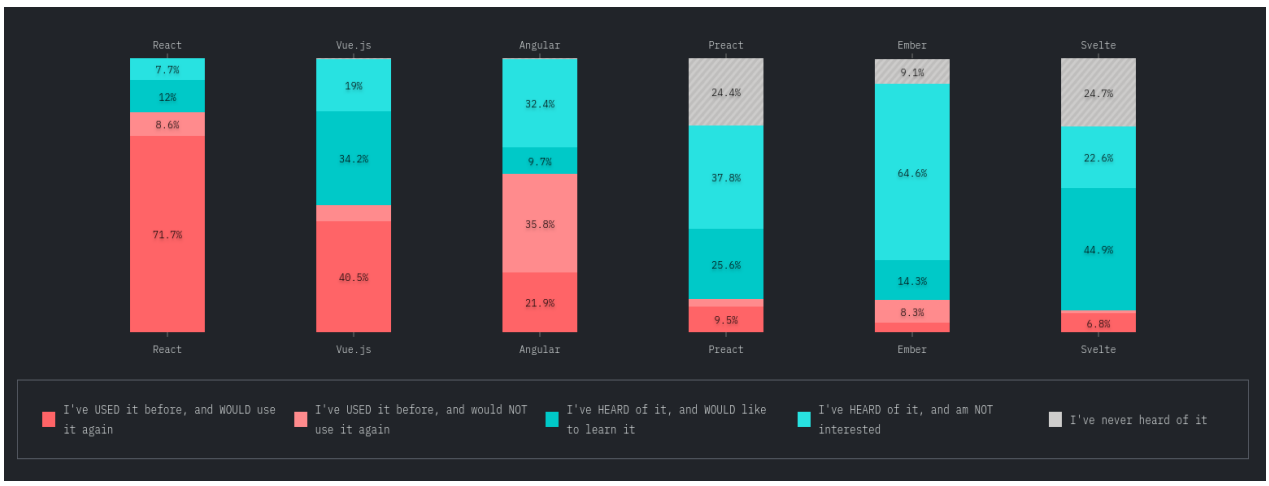
Why Express?

It allows simple creation of web servers as well as complex APIs. It has high capability to handle extreme network loads efficiently. The following chart shows the popularity of Express as compared to its alternatives, the darker indicates more positive than a lighter color:



Why ReactJS?

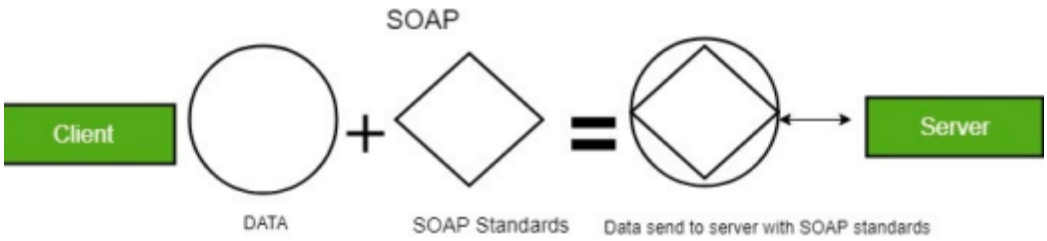
ReactJs is a light-weighted library maintained and developed by Facebook. It is easy to learn and provides strong control over the data. The following chart shows the popularity of Express as compared to its alternatives, the darker indicates more positive than a lighter color:

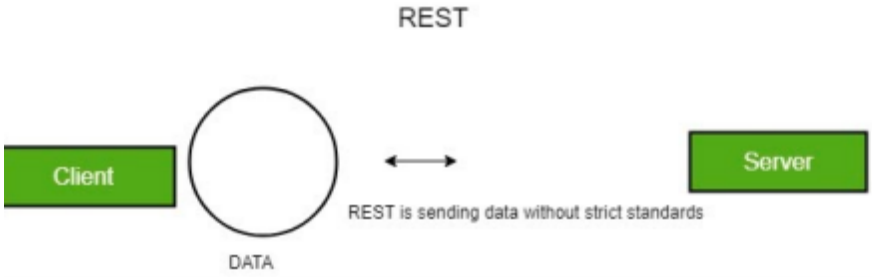


The server uses the RESTful Architecture which is the acronym for Representational State Transfer. REST is a software architectural style that defines the set of rules to be used for creating web services. It is a lighter weight alternative to its predecessor SOAP for accessing web services as it makes use of HTTP, unlike SOAP which relies heavily on XML.

Few more advantages of REST over SOAP are:

- Smaller learning curve
- Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
- Fast (no extensive processing required)
- Closer to other web technologies in design philosophy



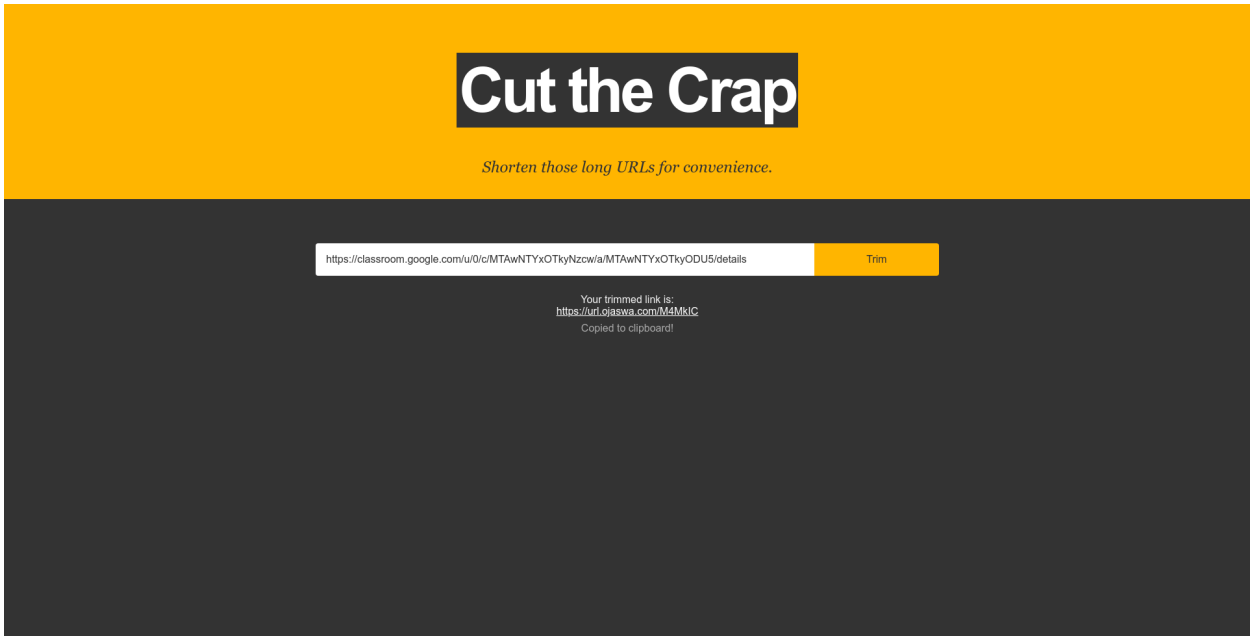


Conclusion

URL Shortener generates a short URL for any type of link or URL. The report discusses various technologies and methodologies used to develop and deploy the service. A central express server serves the front-end and provides an API for facilitating the service. The API generates a short URL for the provided long URL and redirects users to the corresponding website when they access a trimmed link.

The service hence provides an easy, secure, responsive and multi-platform solution for trimming long web URLs.

The solution is deployed at: https://url.ojaswa.com. Following is a snapshot of the working product:



References

- Uniform Resource Locator (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/URL
- World Wide Web (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/World_Wide_Web
- URL Redirection (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/URL_redirection
- Software Architecture (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Software_architecture
- Client (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Client_(computing)
- User (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/User_(computing)
- Server (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Server_(computing)
- Database (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Database
- Web Page (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Database
- HTTP (n.d.), retrieved from World Wide Web Consortium: https://www.w3.org/Protocols/
- HTTPS (n.d.), retrieved from Cloudflare Documentation:

https://www.cloudflare.com/learning/ssl/what-is-https/

- Time to Live (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Time_to_live
- HTTP Methods (n.d.), retrieved from World Wide Web Consortium: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
- PostgreSQL (n.d.), retrieved from PostgreSQL documentation: https://www.postgresql.org/docs/12/index.html
- Primary Key (n.d.), retrieved from PostgreSQL documentation: https://www.postgresql.org/docs/12/ddl-constraints.html#DDL-CONSTRAINTS-PRIMARY-KEYS
- Unique Key (n.d.), retrieved from PostgreSQL documentation: https://www.postgresql.org/docs/12/ddl-constraints.html#DDL-CONSTRAINTS-UNIQUE-CONSTRAINTS
- ReactJs (n.d.), retrieved from ReactJs documentation by Facebook: https://reactjs.org/docs/getting-started.html
- JavaScript (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/JavaScript
- JavaScript (n.d.), retrieved from MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript
- React Compiler (n.d.), retrieved from Facebook Prepack Wiki: https://github.com/facebook/prepack/wiki/React-Compiler
- Components (n.d.), retrieved from ReactJs documentation by Facebook: https://reactjs.org/docs/components-and-props.html
- JSX (n.d.), retrieved from ReactJs documentation by Facebook: https://reactjs.org/docs/introducing-jsx.html
- REST (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Representational_state_transfer
- Express (n.d.), retrieved from ExpressJs official website: https://expressjs.com
- NodeJs (n.d.), retrieved from NodeJS Official Website: https://nodejs.org/
- CORS (n.d.), retrieved from MDN: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- V8 (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/V8_(JavaScript_engine)
- Environment Variables (n.d.), retrieved from The Wikipedia:

https://en.wikipedia.org/wiki/Environment_variable

- AWS (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Amazon_Web_Services
- Hosting (n.d.), retrieved from NameCheap Documentation: https://www.namecheap.com/hosting/what-is-web-hosting-definition/
- LightSail (n.d.), retrieved from AWS Documentation: https://aws.amazon.com/lightsail/
- Nginx (n.d.), retrieved from official NginX site: https://www.nginx.com/
- System Service (n.d.), retrieved from Linux programming Manual: http://man7.org/linux/man-pages/man5/systemd.service.5.html
- Reverse Proxy (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Reverse_proxy
- DNS (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Domain_Name_System
- SSL (n.d.), retrieved from a private SSL providing company: https://www.ssl.com/faqs/faq-what-is-ssl/
- LetsEncrypt (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Let%27s_Encrypt
- CI / CD (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Continuous_integration and https://en.wikipedia.org/wiki/Continuous_delivery
- GitHub Actions (n.d.), retrieved from official page: https://github.com/features/actions
- YAML (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/YAML
- Secure Shell (n.d.), retrieved from The Wikipedia: https://en.wikipedia.org/wiki/Secure_Shell
- JS Survey (n.d.), retrieved from State of Javascript: https://2019.stateofjs.com/