Project Title - URL Shortener

Project Description -

A URL shortener is a web application that allows users to tiny the long usual URLs of the address bar. The project is based on the MERN stack where MongoDB stores the long URLs, their shortened version, and the number of times users clicked on a short URL to keep track of analytics.
The database has two tables - one for users' information and one to store URLs. The primary key of user collection(user_id) is related to URL collection(as a foreign key) to keep dynamic URLs related to users.

The project uses Express as a web application framework built on top of Node js that helps manage servers and routes. Routes are created on the server side to Register, and Login Users, fetch URls based on Users' info, and post URLs to shorten them.

For asynchronous programming, NodeJS was used to handle server-side requests and to connect with the database.

For the front end, React JS Framework was used. This project has the following ReactJS components -
Registration Component: This component allows users to register by providing their information, such as username, email, and password. It communicates with the server to store the user's information in the user collection of the MongoDB database.
Login Component: The login component enables users to log in using their credentials, such as username and password. It authenticates the user by verifying the entered information against the stored user data in the database.
URL List Component: This component displays a list of URLs associated with the logged-in user. It fetches the URLs based on the user's information from the server and renders them on the screen. Each URL item can show additional information, such as the number of clicks.
URL Form Component: The URL Form component allows users to enter a long URL and submit it to be shortened. It sends a POST request to the server, which then generates a shortened URL and stores it in the URL collection of the database.

URL Item Component: This component represents an individual URL in the URL List. It displays the original URL, the shortened version, and the number of clicks. Users can click on the shortened URL to redirect to the original destination.

How to run the Project?

Use *npm install -I <dependency_name>* to install the following dependencies used in the project -

Server Side -

1. **Bcrypt -** For hashing passwords.
2. **Cors** - It is a protocol that enables scripts running on a browser client to interact with resources from a different origin.
3. **Dotenv -** Loads environment variables from .env file.
4. **Express** - Minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
5. **Express-async-handler** - Simple middleware for handling exceptions inside of async express routes and passing them to your express error handlers.
6. **Jsonwebtoken** - JWT allows you to decode, verify and generate JWT for authenticated login. It has three parts - header, payload (data by client, verify signature).
7. **Mongodb-** The official MongoDB driver for Node.js.
8. **Mongoose -** Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment.
9. **Shortid -** ShortId creates short non-sequential url-friendly unique ids. Used for shorten URLS.

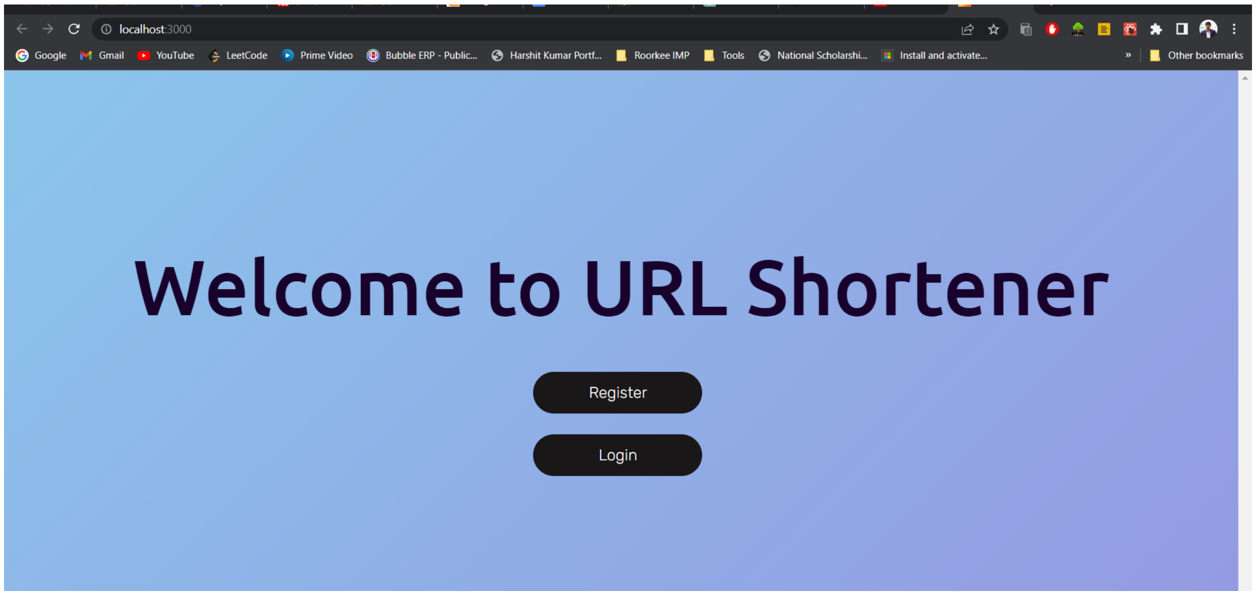
Client Side -

1. **React -** The react package contains only the functionality necessary to define React components.
2. **React-dom -** This package serves as the entry point to the DOM and server renderers for React.
3. **React-router -** For routing within components and defining Links.

To Run the Project -

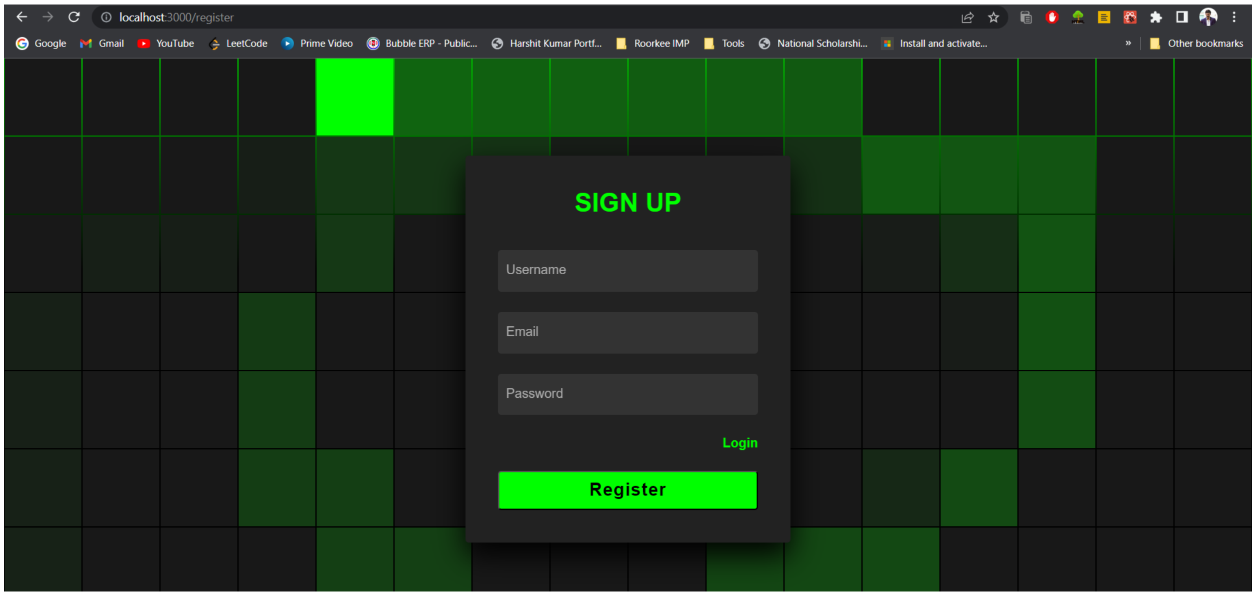
- The repository has two main folders - client and server.
  Client - for frontend, listening user requests.
  Server - for backend, listening client requests.
  - Server is running on `"http://localhost:3001"`.
  - Client is running on `"http://localhost:3000"`.
  - Server and client have their own package.josn files.
  - Under package.json of Client, **proxy:** `"http://localhost:3001"`, is used to connect with server.
- Open the terminal, type *cd client -> npm start* to open client side.
- Open another terminal and do follow the same code to open server side.
- Make sure console is displaying the message "Database is connected - <database_name>
- After database is connected, open `"http://localhost:3000"` on your browser.
- The page must be displaying Welcome message with Register and Login buttons.
- If you are already registered, proceed to Login button, else click Register button.

Landing Page -



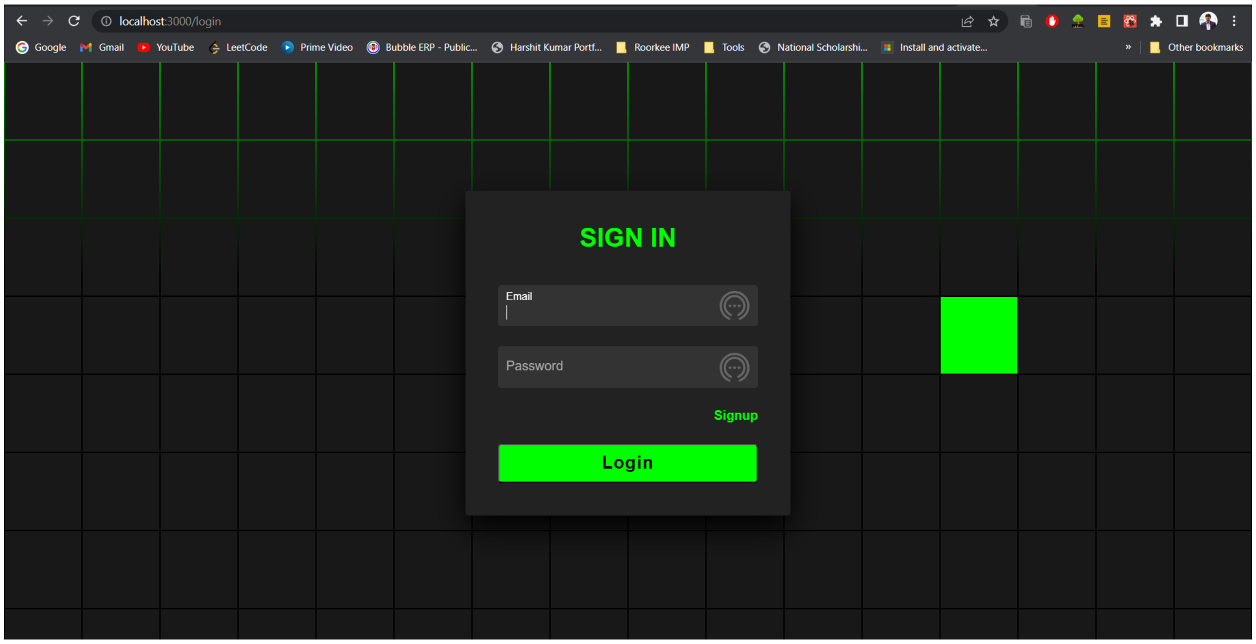
Welcome to URL Shortener

- After clicking on the **register** button, you will be redirected to "http://localhost:3000/register" which will load the Register component from the client.
  Register Page -



- The register page uses a form to take input of the following fields - username, email (unique to each user), and password. We have used the useState property of react to set the value of the mentioned fields.
- After getting the required information, a post request is sent to http://localhost:3001/users/register with req. body = {username, email, password}.
- Under *server/controller/userController.js* request is received, the code ensures that the user is already registered and has entered the correct details.
- After checking, it generates a hashed password for the user using bcrypt.hash method. The user is then stored in the database with the hashed password generated from the given password.
- When the post request is successful, the register component navigates the user to log in using **navigate(./login).**
- On the Login page, the user is requested to enter the email and password they used for registering.

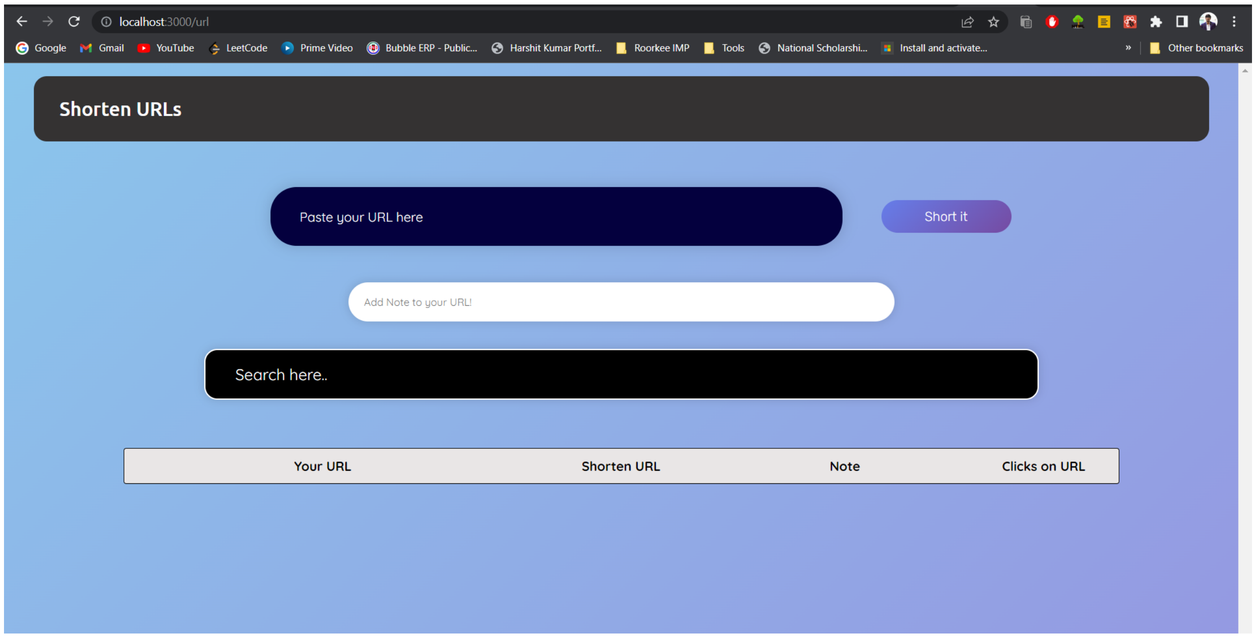
- After the user enters their email and password, the login page sends post request to "http://localhost:3001/users/login".
  Sign-in Page-



- Under *server/controller/userController.js* login request is received, based on request, server checks if the email exists.
- If the email exists, it compares passwords stored in bcrypt form using bcrypt.compare method.
- If a user successfully logs in, it generates a jwt token using jwt.sign method which requires three parameters- user data, token secret key, expiration time.
- Further, the token is sent to the login component and the token is set to localStorage using setItem method and the user is redirected to "http://localhost:3000/url".
- This loads the URL component which fetches the get request to http://localhost:3001/url.
- Under *server/controller/urlController.js,* the request is received and it passes through middleware validate token. The middleware ensures authentication using jwt.verify method.
- After the request, it fetches all the URLs. Since, the user is new, there are no URLs present in the table.

- We have Paste URL input box, and url note input box present in the URL component which takes the value of full url and note respectively and post it to http://localhost:3001/url.

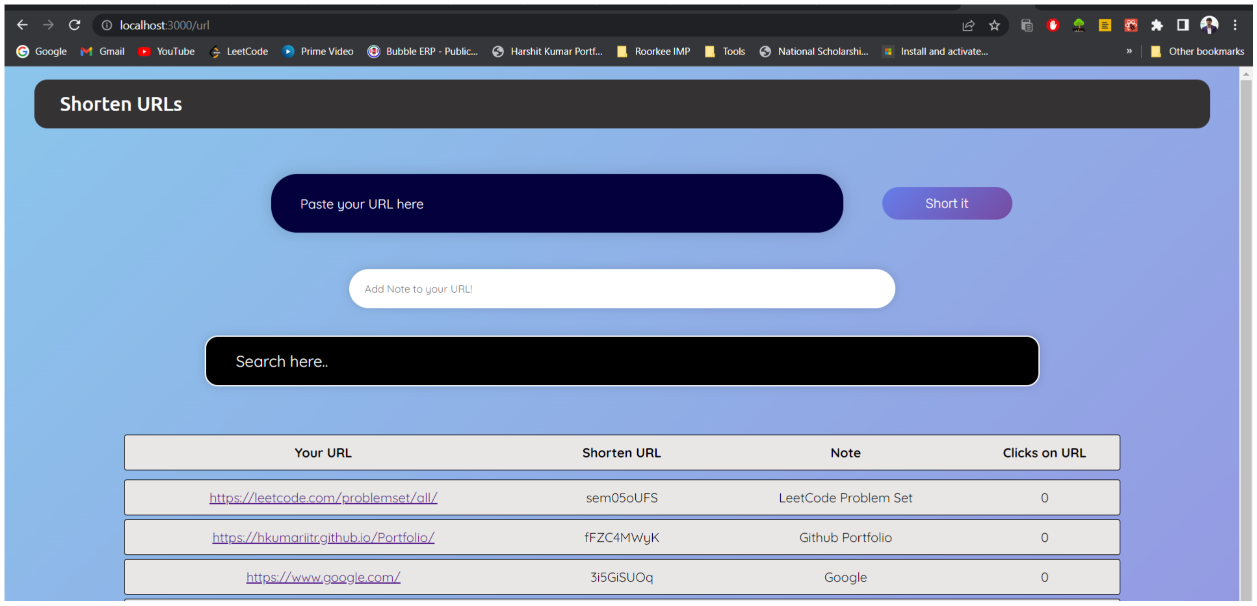
URL Component (with no data) -



Your URL	Shorten URL	Note	Clicks on URL

- After submitting our URL in paste box with note added in it, user clicks on short it button to post request.
- Server listens the request and make shorturl corresponding to URL given, and sends it to frontend.

URL Component (with data) -



Your URL	Shorten URL	Note	Clicks on URL
https://leetcode.com/problemset/all/	sem05oUFS	LeetCode Problem Set	0
https://hkumariitr.github.io/Portfolio/	fFZC4MWyK	Github Portfolio	0
https://www.google.com/	3i5GiSUOq	Google	0

- Based on our input given , it sends back the shortened urls generated using shortid package, with notes and clicks section.
- Click section updated each time when we click on short url.
- Clicking on short url do the following task - it first fetches the full url associated with it from the backend, then after receiving it on frontend, window.open(fullURL) is used to open the corresponding url and respectively the clicks gets updated to +1 each time we click on shorturl.
- This is done with the help of post request to `` `/url/${shortUrl}. ``
- To implement search functionality to the application, we have given a search input box on the url component.
- Whenever user types a query 'q' on search box corresponding get request is sent.
- On the server side, q is fetched using req.query and then applying filter method on the database using this q gives us the desired database.

```
shorturls.filter(url=> url.full.includes(q) ||
url.short.includes(q) || url.note.includes(q))
```

- Whenever user types a query 'q' on search box corresponding get request is sent.
- Searching is done on backend.
- This way the user securely give URLs, add note to it and generate the urls keeping track of the analytics.

Learning Takeaways -

- Using cors becomes a necessity when server and client are used on different ports.
- Password hashing becomes useful while securing users information.
- It must be ensured that headers are set correctly while making a request to server.
- Server must send required data to client. Irrelevant information creates confusion.
- Securing server requests using middlewares to enable user authentication.

References -

- https://www.npmjs.com/ - for reading package related information (short id)
- https://www.youtube.com/watch?v=MY6ZZIn93V8&pp=ygUcc2VhcmNoIGZ1bmNhdGlvbGl0eSBsYW1hIGRldg%3D%3D - to implement search functionality.