STEGANOGRAPHY-BASED EMAIL OTP VERIFICATION SYSTEM

A PROJECT REPORT

Submitted by

KUNAL CHANDRA (19BCY10028)
GOPAL DANDOTIYA (19BCY10047)
SHIVANK SHARMA (19BCY10058)
JASLEEN KAUR (19BCY10150)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in
CSE specialization in Cyber Security and Digital Forensics

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT BHOPAL UNIVERSITY

KOTHRI KALAN, SEHORE
MADHYA PRADESH - 466114

MARCH 2023

VIT BHOPAL UNIVERSITY, KOTHRI KALAN, SEHORE MADHYA PRADESH – 466114

BONAFIDE CERTIFICATE

Certified that this project report titled **"REPLACING TRADITIONAL OTP SYSTEM WITH AN ENHANCED SECURITY TECHNIQUE"** is the bonafide work of "**KUNAL CHANDRA (19BCY10028), GOPAL DANDOTIYA (19BCY10047), SHIVANK SHARMA (19BCY10058), JASLEEN KAUR (19BCY10150)"** who carried out the project work under my supervision. Certified further that to the best of my knowledge, the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

DR. D. SARAVANAN
Program Chair - B.Tech Cyber Security and Digital Forensics
SCSE
VIT BHOPAL UNIVERSITY

DR. SOMA SAHA
Project Guide
SCSE
VIT BHOPAL UNIVERSITY

The Capstone is held on **31 MAR 2023**.

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to **Dr. Shishir Kumar Shandilya**, Head of the Department, School of Computer Science and Engineering, and **Dr. D. Saravanan,** Program Chair of our Cyber Security and Digital Forensics department for much of their valuable support and encouragement in carrying out this work.

I would like to thank my internal guide, **Dr. Soma Saha** for continually guiding and actively participating in my project, and giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of the School of Computer Science and engineering, who extended directly or indirectly all support.

Last, but not least, I am deeply indebted to my parents who have been the greatest support while I worked day and night for the project to make it a success.

ABSTRACT

The traditional one-time password or passcode (OTP) is a string of characters or numbers that authenticates a user for a single login attempt or transaction. An algorithm generates a unique value for each one-time password by factoring in contextual information, like time-based data or previous login events. The idea of our project is to build a stronger and more modern OTP system that would replace the traditional OTP methodology. This is going to be achieved by the implementation of steganography into the existing system which definitely has multiple flaws and we aim to implement our understanding of the concepts to overcome a select few of those. As known, In the general One-time password system the OTP comes through an SMS, anyone around the user could be able to take a peek at the user's device screen and get access to the OTP even through the lock screen as the OTP is visible on the screen upon arrival via notifications. We aim to enhance the security of this current system

[PURPOSE-METHODOLOGY-FINDINGS]

The purpose of our project is to enhance the current OTP system.

We're achieving the purpose of our project by using the Steganography technique in the current system.

We found out that the current OTP system is majorly flawed, so we came up with a better and more innovative solution for this problem.

TABLE OF CONTENTS

INTRODUCTION

The traditional one-time password or passcode (OTP) is a string of characters or numbers that authenticates a user for a single login attempt or transaction. An algorithm generates a unique value for each one-time password by factoring in contextual information, like time-based data or previous login events. The idea of our project is to build a stronger and more modern OTP system that would replace the traditional OTP methodology. This is going to be achieved by the implementation of steganography into the existing system which definitely has multiple flaws and we aim to implement our understanding of the concepts to overcome a few of those.

PROBLEM STATEMENT

As known, In the general One-time password system the OTP comes through an SMS, anyone around the user could be able to take a peek at the user's device screen and get access to the OTP even through the lock screen as the OTP is visible on the screen upon arrival via notifications. We aim to enhance the security of this current system.

OBJECTIVE OF THE WORK

Our main objective is the security enhancement of the currently used OTP verification system. We already know that the traditionally used system has many flaws. The major drawback we came across is that anyone can sneak peek into others' devices. Our project will try to come up with a solution for this drawback.

We aim to enhance the security of this current system. Then, the image will reach the decryption module and the OTP will be recovered from the image which will then be transferred to the Verification section to verify the OTP.

RELATED WORK INVESTIGATION

The existing work is known in the original system as, One Time Password, a.k.a OTP.

It is a channel from which we can access several services such as banking, shopping, social media accounts, and countless more. Simply put, we can always remain online and access the internet anytime we feel like it. However, this has also given a rise to data theft. We may not realize it, but our mobile devices can become a victim of malicious malware lurking on the net.

To overcome such breaches many online service providers introduced the concept of SMS-based OTP authentication. Its goal is to reduce phishing attacks as well as to address any security-related risk on the internet in regard to user authentication. The concept is to enter an OTP received on your mobile to verify your credentials on the website you want to use. Simple and secure enough.

However, in the current OTP system, anyone around the user can peek into his mobile phone and get access to the OTP even through the lock screen as the SMS can be seen through the lock screen as well. This is considered to be a major drawback because privacy is being breached.

REQUIREMENT ARTIFACTS

Hardware Requirements

- x86 64-bit CPU (Intel / AMD architecture)
- 4 GB RAM.
- 5 GB free disk space

Software Requirements

- Operating system: Linux- Ubuntu 16.04 to 17.10, or Windows 7 to 10
- Nodejs IDE
- Flask framework
- Steganography tool

DESIGN METHODOLOGY AND ITS NOVELTY

The current OTP systems directly send randomly generated numbers as a verification mechanism, but our system would be an enhanced version of the current system using an encryption algorithm to ensure the privacy of the individual.

We would do so by using randomly generated images with the OTP steganographic ally placed on the image in an encrypted fashion.

The user will have to upload the image directly from their device, which would also prevent procedures like peeking etc.

MODULES

(1) Database Connection: A database connection module is a software component that provides a set of functions or methods to connect to a database system, establish a communication channel, execute queries or commands, and retrieve or update data.
The module includes parameters or settings to specify the database type, hostname or IP address, port number, authentication credentials, encryption options, and other configuration options.
The module may also provide error handling mechanisms, such as exceptions or error codes, to report and handle connection errors, query syntax errors, data type errors, or other types of errors that may occur during the database interaction.
We have used Mongo dB to connect our database.

(2) User Schema: In Node.js, a user schema is typically used to define the structure of a user object or document that is stored in a database. The user schema defines the properties or fields that a user object should have and their data types, validation rules, and default values.

(3) Server Configurations: We have considered various parameters to ensure the optimal performance and security of our Server.
Set up a replica set: A replica set is a group of MongoDB servers that maintain the same data set.

- Enable authentication: By default, MongoDB does not require authentication for database access. However, it is recommended to enable authentication by creating user accounts with specific roles and permissions.
- Use connection pooling: Connection pooling is a technique that allows multiple clients to share a pool of database connections.

- Optimize query performance: MongoDB provides various query optimization techniques, such as indexing, aggregation, and projection, to improve query performance.
- Set up SSL/TLS encryption: SSL/TLS encryption can secure the communication between the MongoDB server and clients, such as Node.js applications.
- Use environment variables for sensitive information: To avoid hard-coding sensitive information, such as database credentials, in our Node.js code, we have used environment variables.

(4) User Verification: User verification in Node.js typically involves verifying the user's identity using a combination of authentication and authorization techniques.
Here are some common steps that we followed diligently to implement user verification in Node.js:

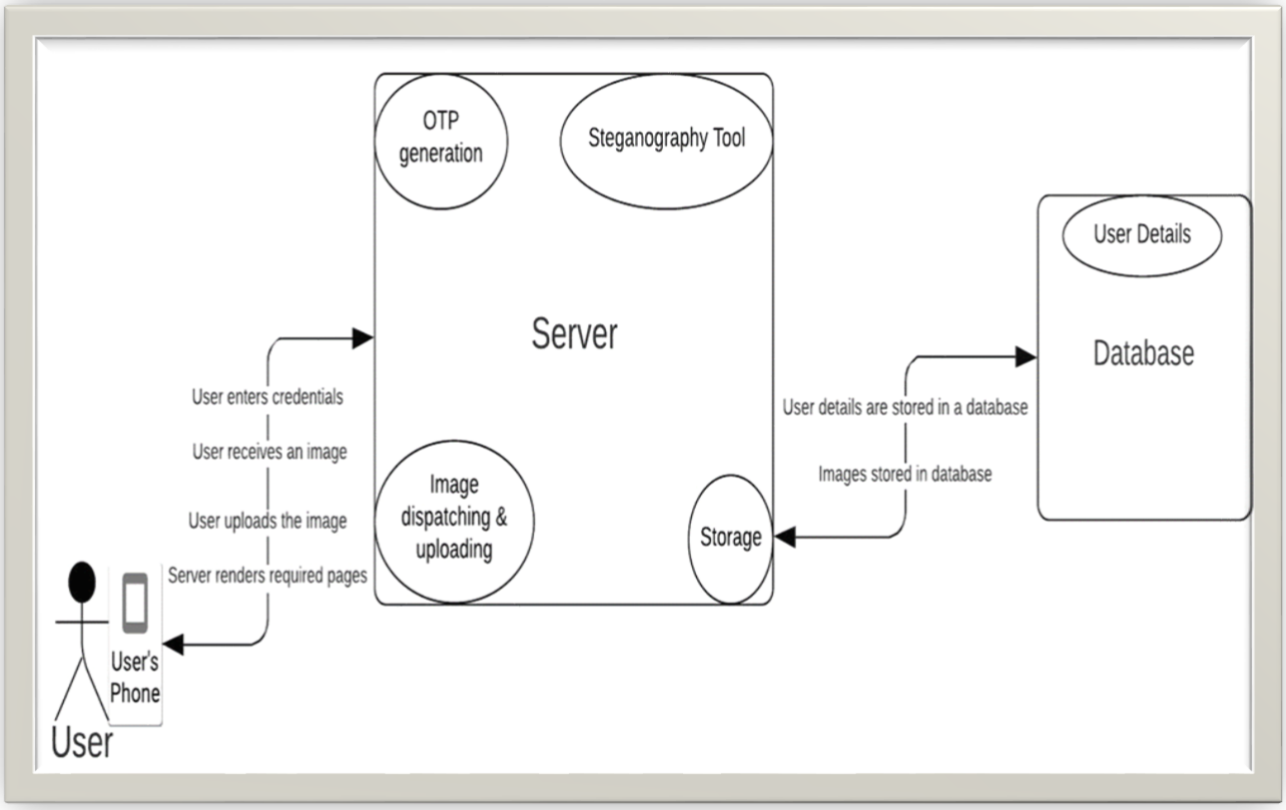
- Implement authentication: The first step is to authenticate the user's identity using a username and password or other credentials.
- Store user information securely: Once a user is authenticated, we store their information securely in a database or other data store.
- Implement authorization: Once a user is authenticated, you should implement authorization to control access to protected resources or actions.

(5) API Testing: API testing is a type of software testing that involves testing the functionality, reliability, performance, and security of an application programming interface (API). API testing is essential for ensuring that the API functions as intended and integrates smoothly with other components of the software system.

Here are some key steps for conducting API testing:

- Identify the API endpoints: The first step in API testing is to identify the API endpoints, which are the URLs that the client applications use to communicate with the API.
- Create test cases: Once you have identified the API endpoints, you can create test cases to verify the functionality and performance of the API.
- Execute test cases: After creating test cases and selecting a testing tool, you can execute the test cases and analyze the results.
- Monitor API performance: In addition to testing the functionality of the API, it is important to monitor its performance over time.
- Conduct security testing: Finally, it is important to conduct security testing to identify and mitigate potential security vulnerabilities in the API.

ARCHITECTURAL DESIGN



TECHNICAL IMPLEMENTATION & ANALYSIS

Database Connection

```
const mongoose = require('mongoose');

const connectDB = async () => {
    try {
        mongoose.set("strictQuery", false);
        const conn = mongoose.connect('mongodb://localhost:27017/capstone', {
            useUnifiedTopology: true,
            useNewUrlParser: true,
        })


        if (conn != undefined) {
            console.log("Connected to database");
        }
    } catch (error) {
        console.log("Failed to connect to database");
    }
}

module.exports = connectDB;
```

User Schema

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true,
        minLength: 8
    }
})

const User = mongoose.model('user', userSchema);

module.exports = User;
```

Server Configuration

```
const express = require('express');
const app = express();
const connectDB = require('./config/db');
const authRouter = require('./routes/authRoute');
const userRouter = require('./routes/userRoutes');
const cookieParser=require('cookie-parser');

require('dotenv').config({path:'./config/.env'});

app.use(express.json());
app.use(express.urlencoded({
    extended: true
}));

app.use(cookieParser());

app.use(express.static('public'))

app.use('/auth', authRouter);
app.use('/user', userRouter);

app.listen(3000, () => {
    console.log("Server listening on port 3000");
    connectDB();
})
```

User Controller Methods

```
const User = require('../models/userModel');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

module.exports.register = async (req, res) => {
    try {
        const {
            name,
            email,
            password
        } = req.body

        const hash = await bcrypt.hash(password, 10);

        const newUser = await User.create({
            name: name,
            email: email,
            password: hash
        })

        if (newUser != undefined) {
            res.status(200).json({
                message: "User created successfully",
                success: true
            })
        } else {
```

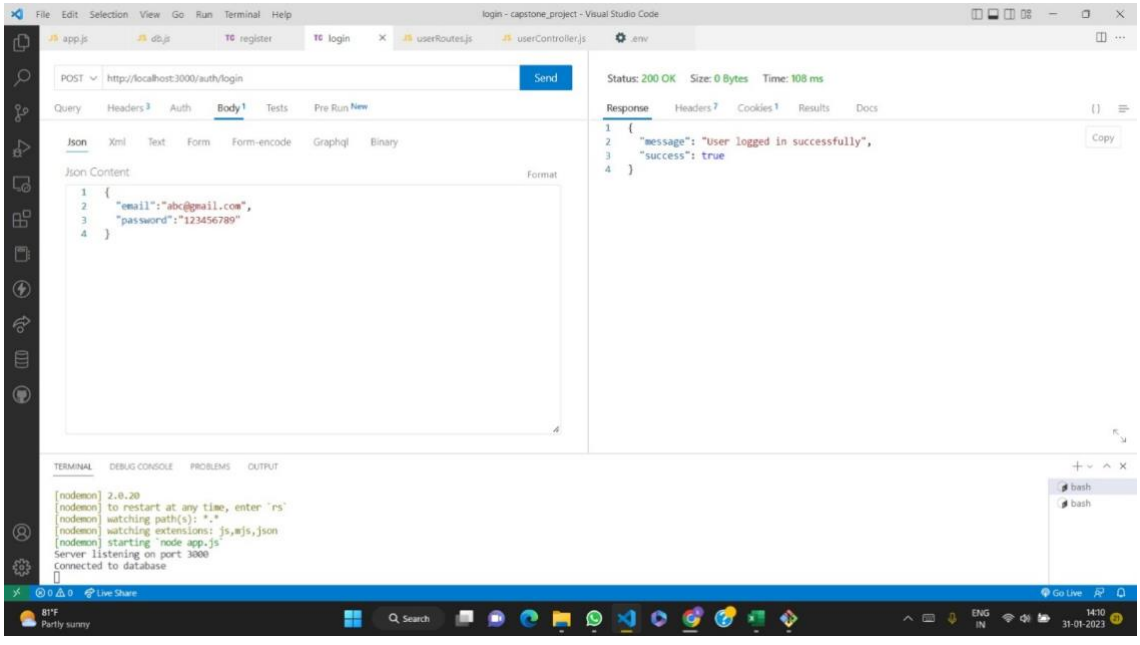
User Verification

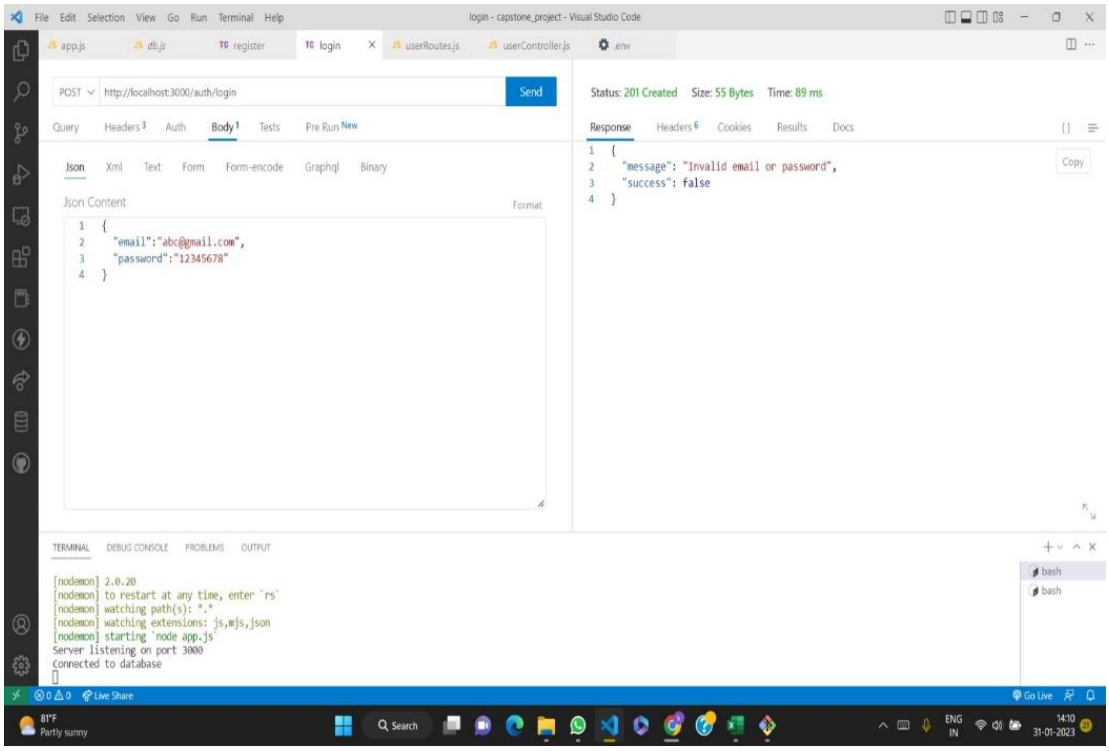
```
const jwt = require('jsonwebtoken');

const verify = async (req, res, next) => {
    try {
        const token = await req.cookies['secret'];

        const payload = jwt.verify(token, process.env.SECRET_KEY);

        req.user = payload.user;

        next();

    } catch (error) {
        res.status(500).json({
            message: "User not authorized, internal server error",
            error: error
        });
    }
}

module.exports = verify;
```

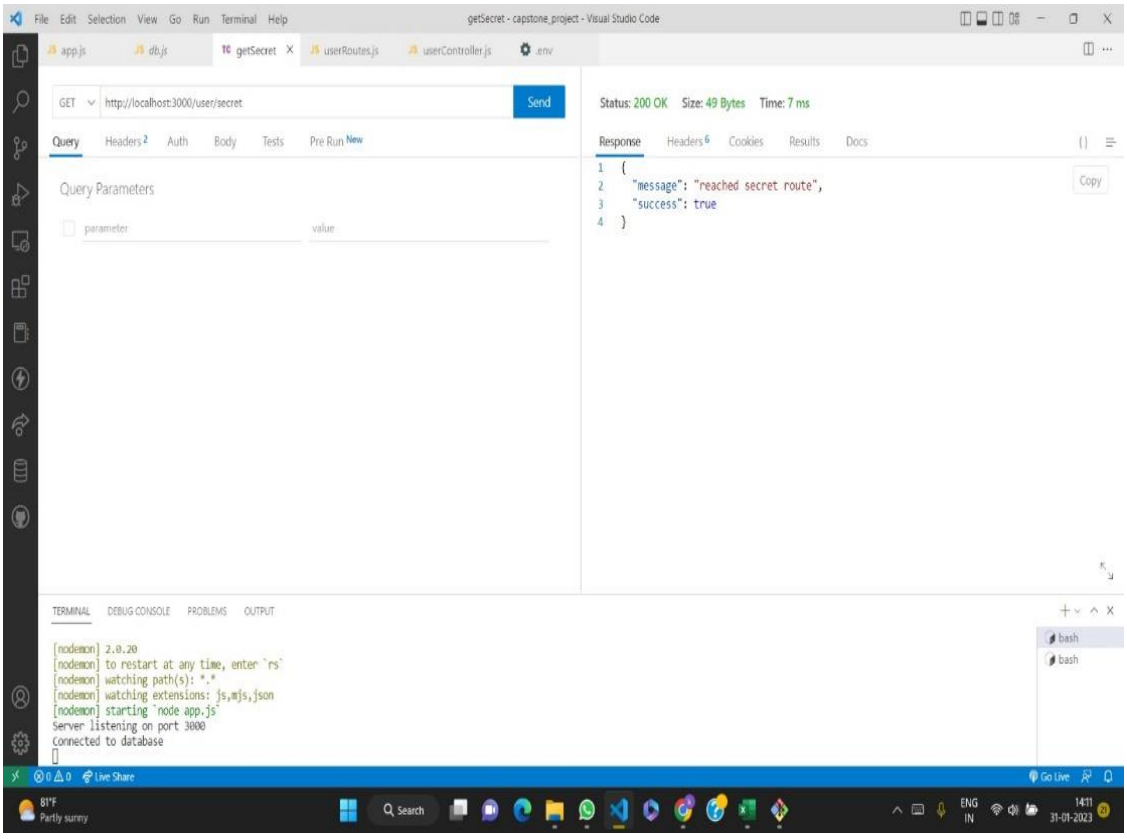
Sending OTP via email

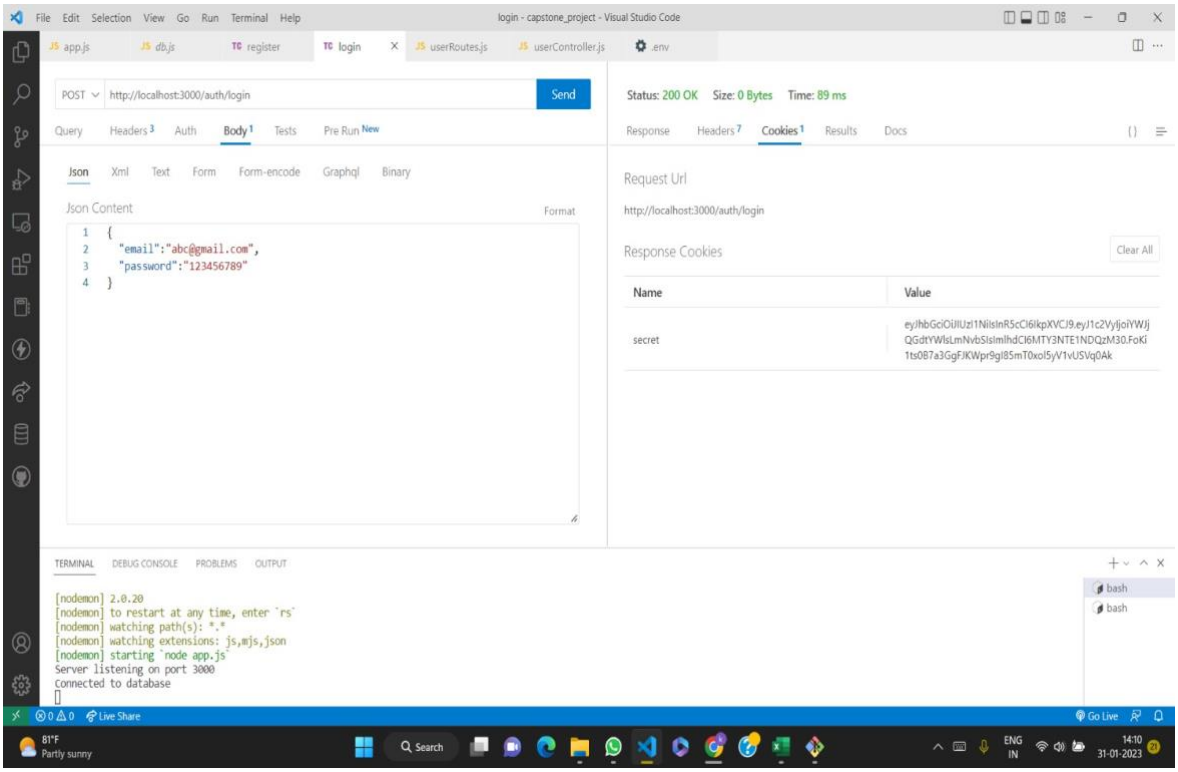
```
const {
    v4
} = require('uuid');

const CryptoJS = require("crypto-js");
const nodemailer=require('nodemailer');

const fs = require('fs');

module.exports.getOTP = async (req, res) => {
    try {

        let transporter = nodemailer.createTransport({
            service: 'gmail',
            auth: {
              user: process.env.EMAIL, //sender
              pass: process.env.PASS
            }
        });


        const id = v4();
        const otp = id.substring(0, 6);

        let data = fs.readFileSync('public/img/sample2.png', 'base64');
```

API Testing

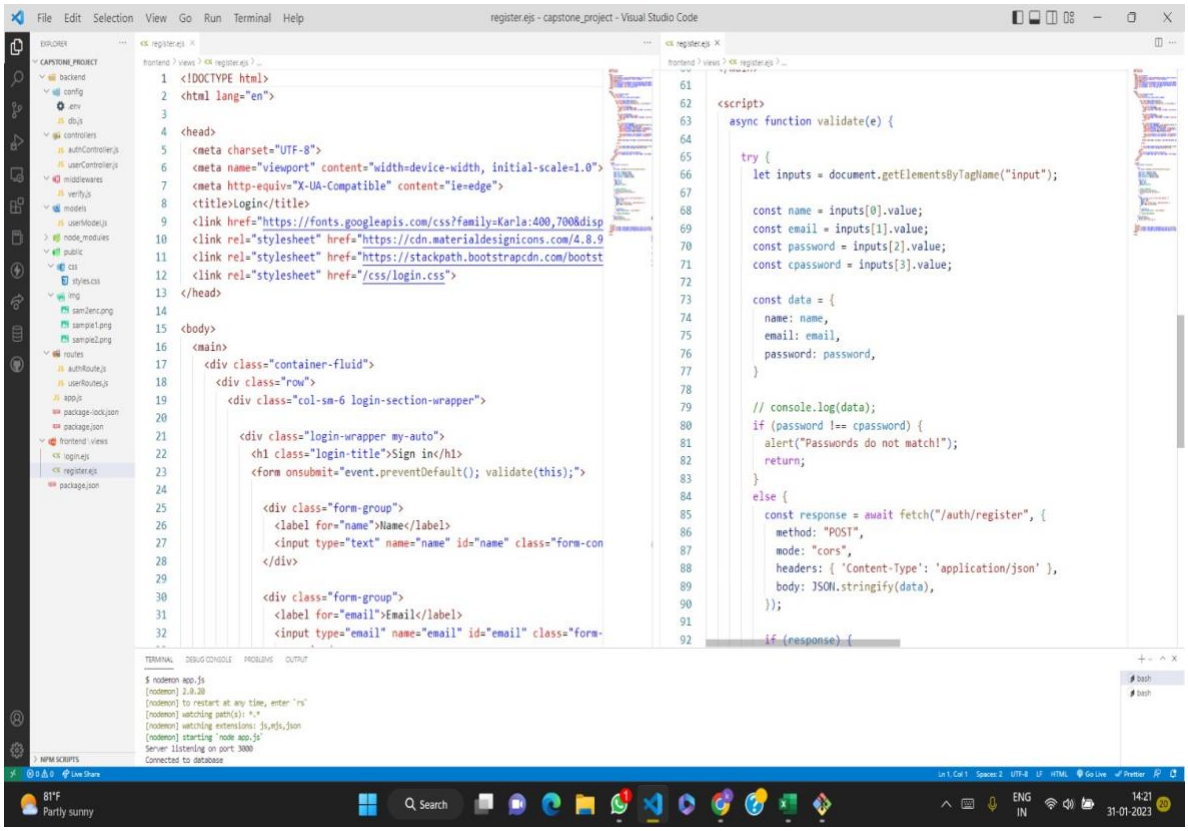



login - capstone_project - Visual Studio Code
POST http://localhost:3000/auth/login
Send
Status: 201 Created Size: 55 Bytes Time: 89 ms
{
"email":"abc@gmail.com",
"password":"12345678"
}
{
"message": "Invalid email or password",
"success": false
}
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server listening on port 3000
Connected to database


getSecret - capstone_project - Visual Studio Code
GET http://localhost:3000/user/secret
Send
Status: 200 OK Size: 49 Bytes Time: 7 ms
{
"message": "reached secret route",
"success": true
}
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server listening on port 3000
Connected to database


File Edit Selection View Go Run Terminal Help
login - capstone_project - Visual Studio Code
app.js
db.js
register
login
userRoutes.js
userController.js
.env
POST
http://localhost:3000/auth/login
Send
Status: 200 OK
Size: 0 Bytes
Time: 89 ms
Query
Headers 3
Auth
Body 1
Tests
Pre Run New
Response
Headers 7
Cookies 1
Results
Docs
Json
Xml
Text
Form
Form-encode
Graphql
Binary
Json Content
Format
{
"email":"abc@gmail.com",
"password":"123456789"
}
Request Url
http://localhost:3000/auth/login
Response Cookies
Clear All
Name
Value
secret
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiYWJjQGdtYWlsLmNvbSIsImlhdCI6MTY3NTE1NDQzM30.FoKi1ts0B7a3GgFJKWpr9gl85mT0xol5yV1vUSVq0Ak
TERMINAL
DEBUG CONSOLE
PROBLEMS
OUTPUT
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server listening on port 3000
Connected to database
bash
bash
Live Share
Go Live
81°F
Partly sunny
Search
ENG
IN
14:10
31-01-2023

Signup-Code

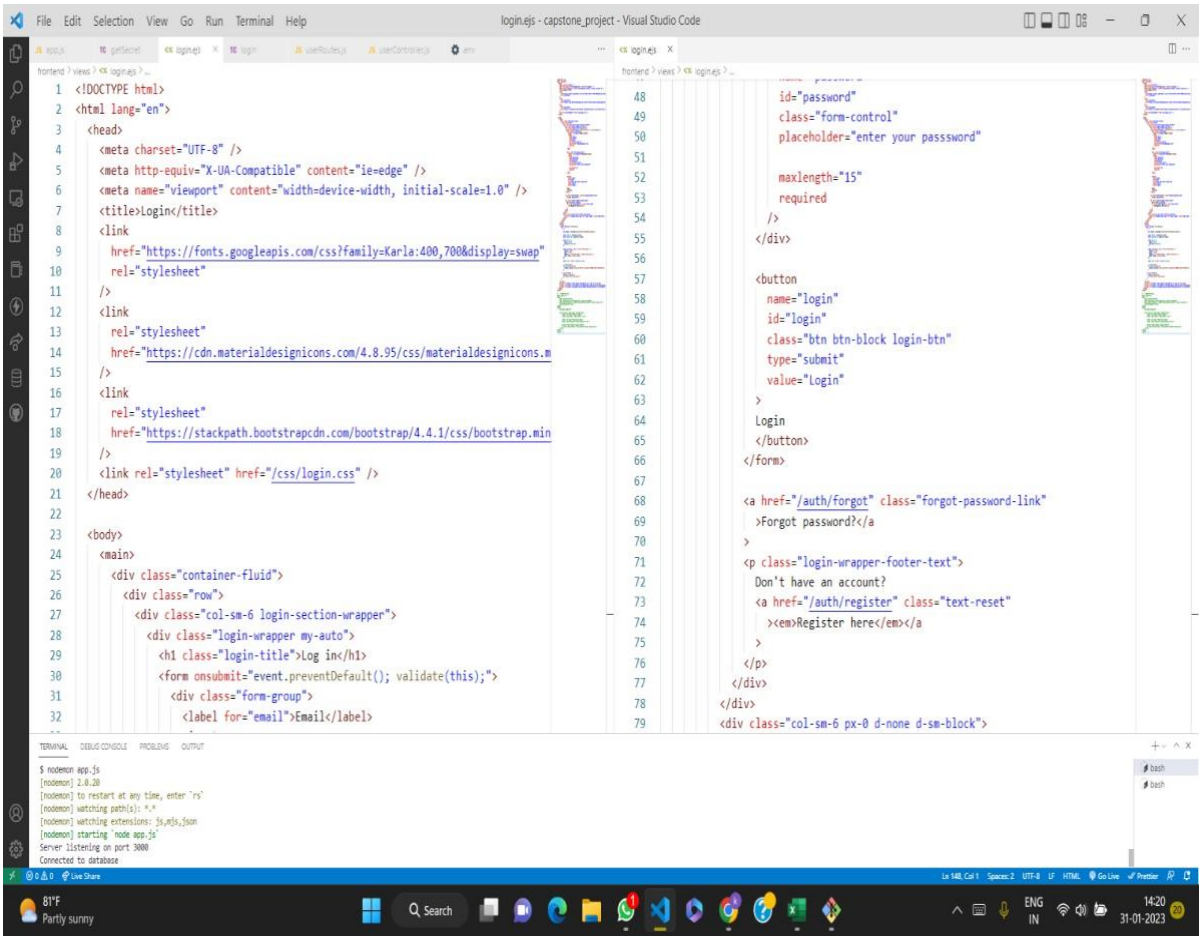

File Edit Selection View Go Run Terminal Help
register.ejs - capstone_project - Visual Studio Code
EXPLORER
CAPSTONE_PROJECT
backend
config
.env
db.js
controllers
authController.js
userController.js
middlewares
verify.js
models
userModel.js
node_modules
public
css
styles.css
img
sam2enc.png
sample1.png
sample2.png
routes
authRoute.js
userRoutes.js
app.js
package-lock.json
package.json
frontend \ views
login.ejs
register.ejs
package.json
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Login</title>
<link href="https://fonts.googleapis.com/css?family=Karla:400,700&disp
<link rel="stylesheet" href="https://cdn.materialdesignicons.com/4.8.9
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootst
<link rel="stylesheet" href="/css/login.css">
</head>
<body>
<main>

<h1 class="login-title">Sign in</h1>
<form onsubmit="event.preventDefault(); validate(this);">

<label for="name">Name</label>
<input type="text" name="name" id="name" class="form-con

<label for="email">Email</label>
<input type="email" name="email" id="email" class="form-
<script>
async function validate(e) {
try {
let inputs = document.getElementsByTagName("input");
const name = inputs[0].value;
const email = inputs[1].value;
const password = inputs[2].value;
const cpassword = inputs[3].value;
const data = {
name: name,
email: email,
password: password,
}
// console.log(data);
if (password !== cpassword) {
alert("Passwords do not match!");
return;
}
else {
const response = await fetch("/auth/register", {
method: "POST",
mode: "cors",
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify(data),
});
if (response) {
TERMINAL
DEBUG CONSOLE
PROBLEMS
OUTPUT
$ nodemon app.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server listening on port 3000
Connected to database
bash
bash
NPM SCRIPTS
Live Share
Go Live
Prettier
81°F
Partly sunny
Search
ENG
IN
14:21
31-01-2023

Login- Code


File Edit Selection View Go Run Terminal Help
login.ejs - capstone_project - Visual Studio Code
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="ie=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Login</title>
<link
href="https://fonts.googleapis.com/css?family=Karla:400,700&display=swap"
rel="stylesheet"
/>
<link
rel="stylesheet"
href="https://cdn.materialdesignicons.com/4.8.95/css/materialdesignicons.m
/>
<link
rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min
/>
<link rel="stylesheet" href="/css/login.css" />
</head>
<body>
<main>

<h1 class="login-title">Log in</h1>
<form onsubmit="event.preventDefault(); validate(this);">

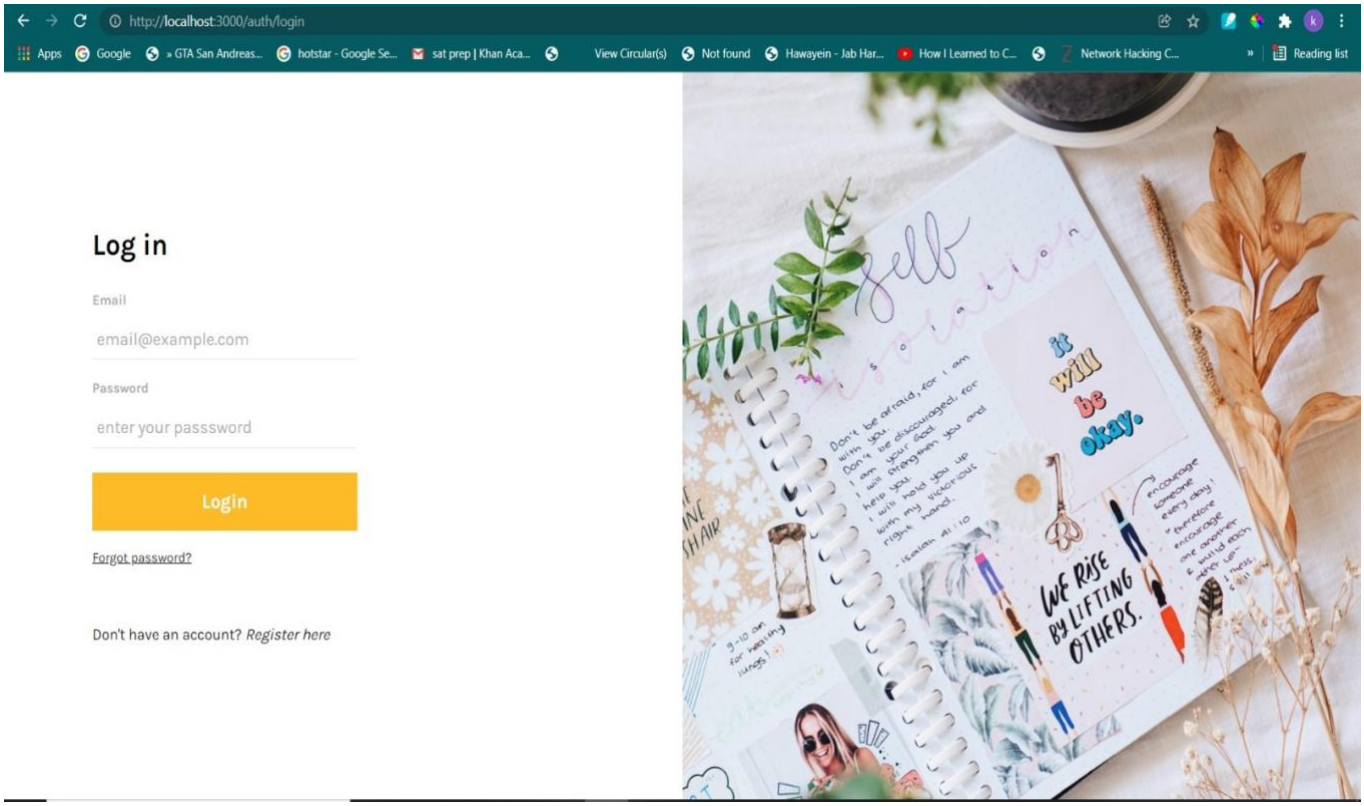
<label for="email">Email</label>
id="password"
class="form-control"
placeholder="enter your passsword"
maxlength="15"
required
/>

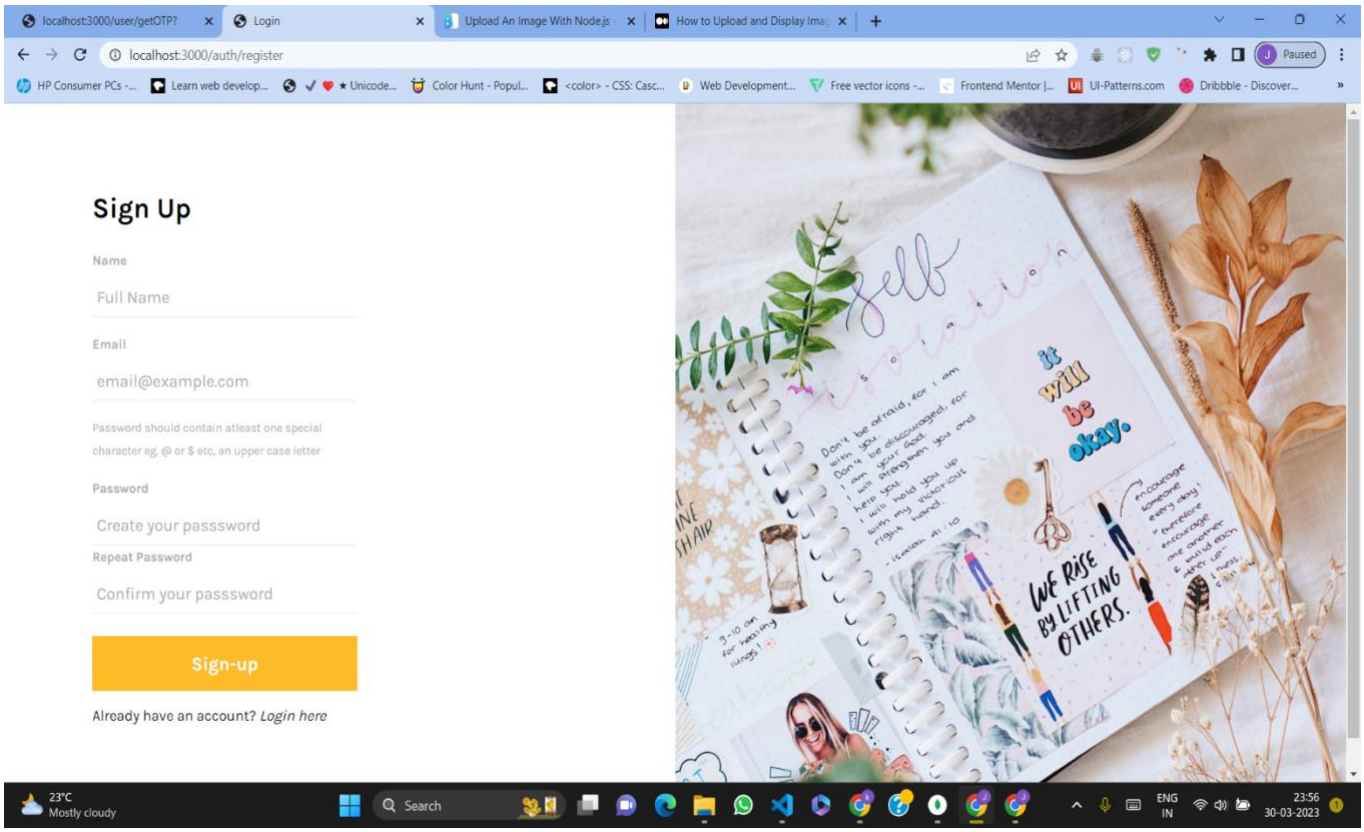
<button
name="login"
id="login"
class="btn btn-block login-btn"
type="submit"
value="Login"
>
Login
</button>
</form>
<a href="/auth/forgot" class="forgot-password-link"
>Forgot password?

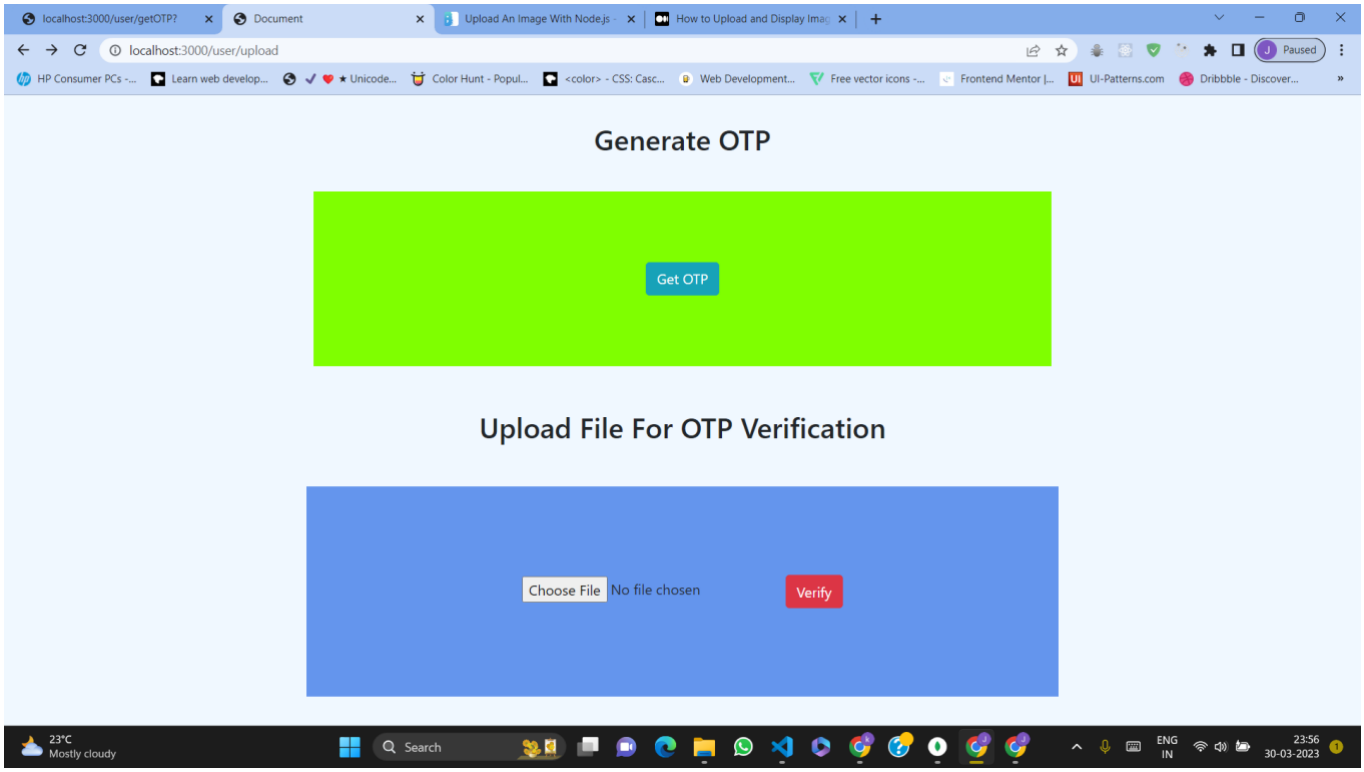
Don't have an account?
<a href="/auth/register" class="text-reset"
>Register here

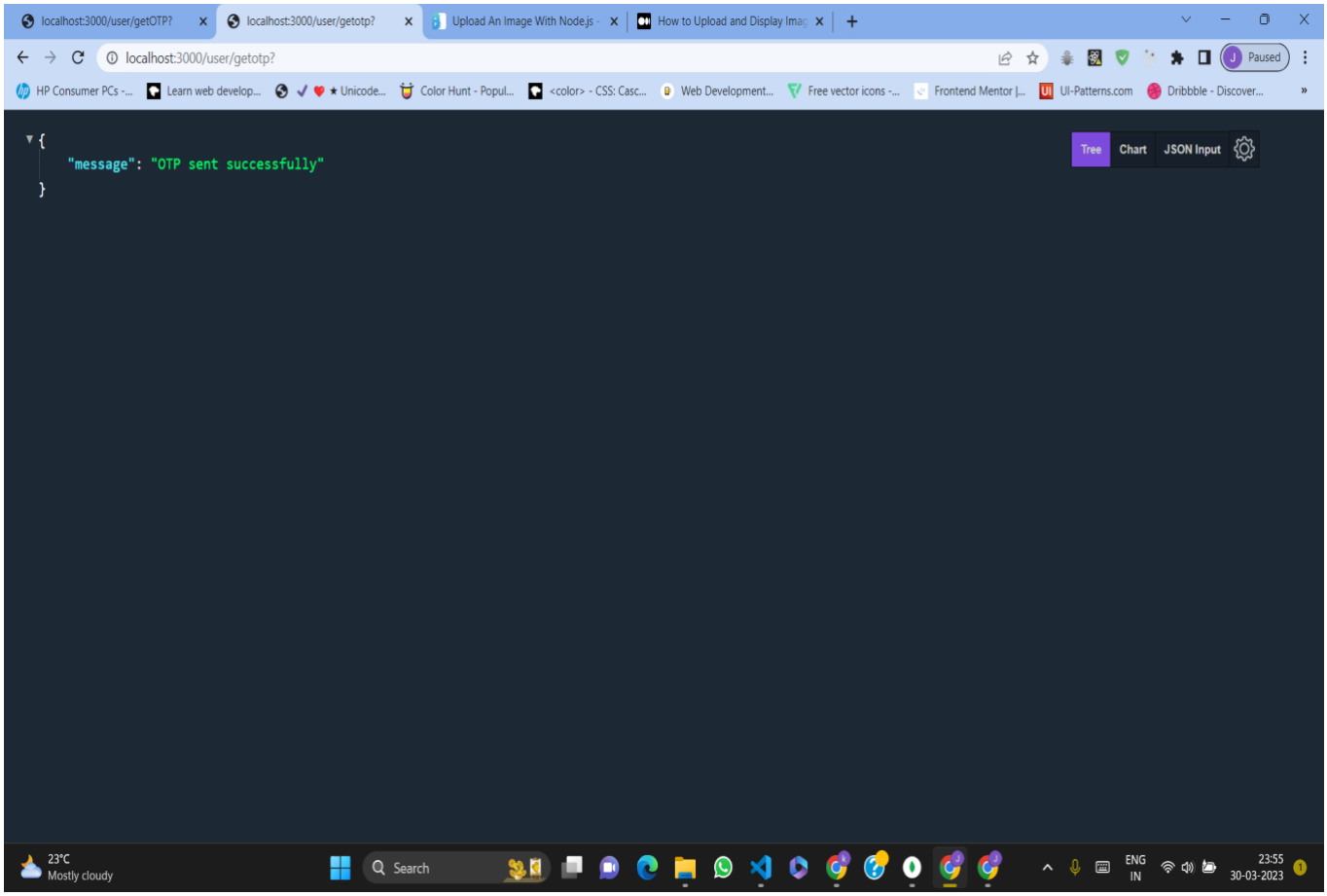
TERMINAL DEBUG CONSOLE PROBLEMS OUTPUT
$ nodemon app.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server listening on port 3000
Connected to database
81°F
Partly sunny
Search
ENG
IN
14:20
31-01-2023

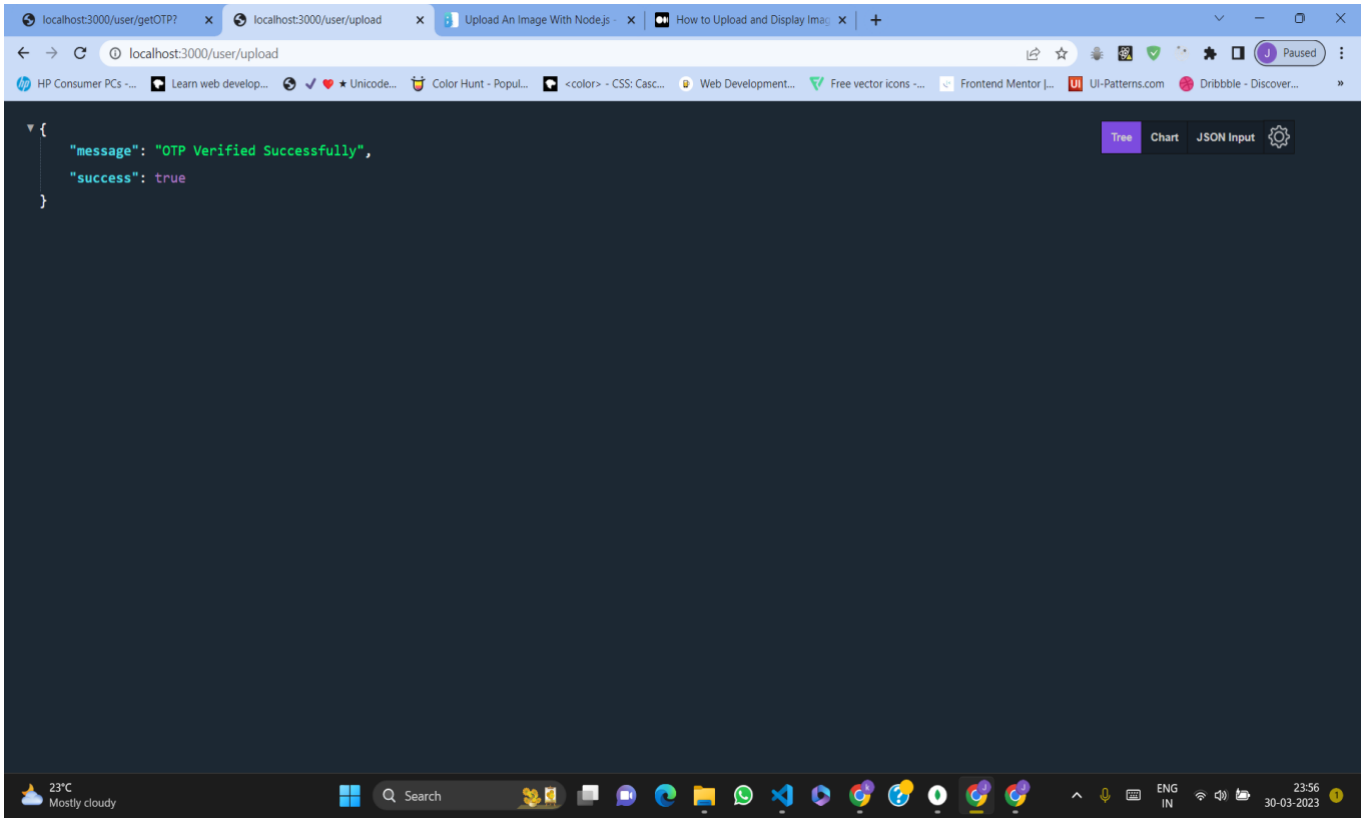
TESTING AND VALIDATION

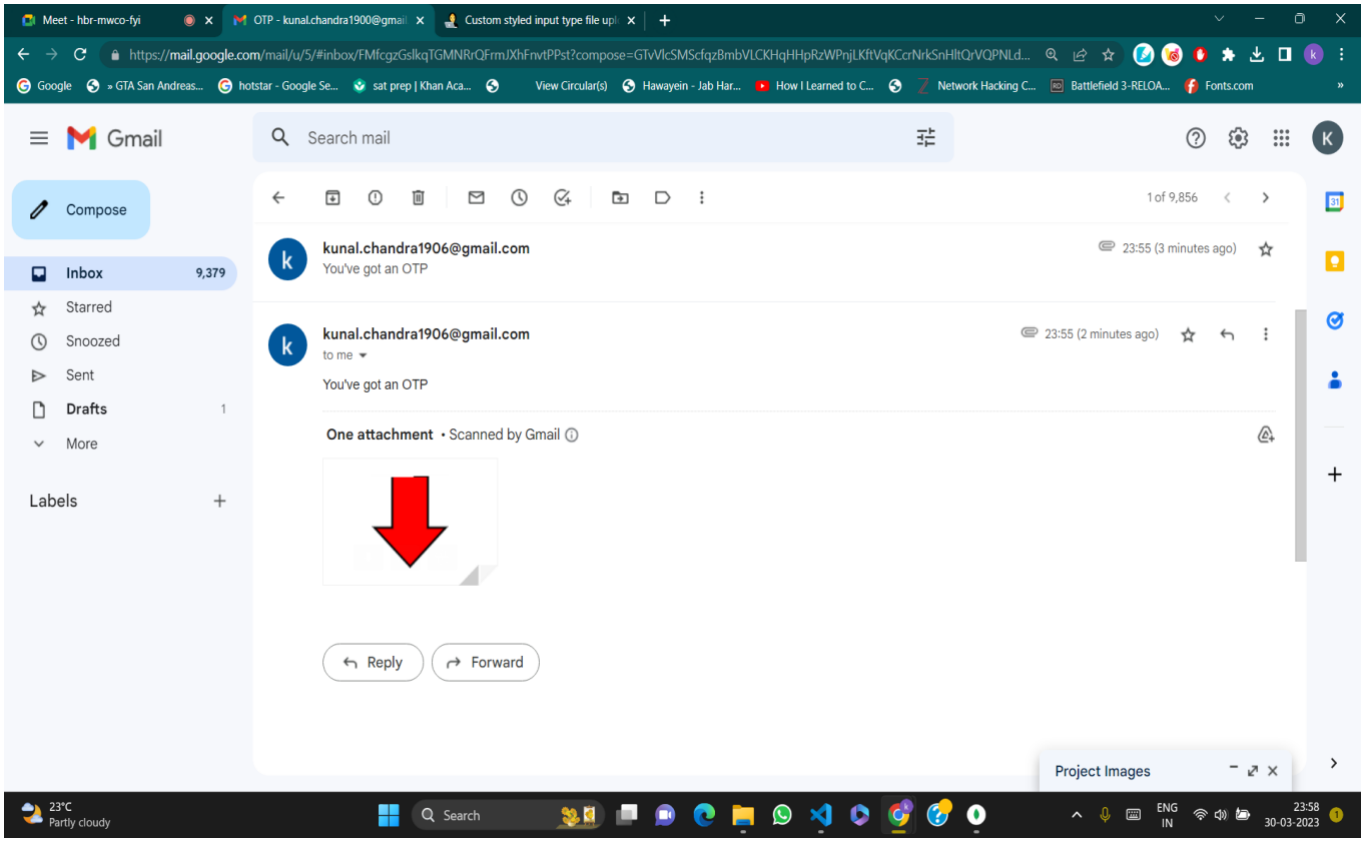





localhost:3000/user/getOTP?
Document
Upload An Image With Node.js
How to Upload and Display Imag
localhost:3000/user/upload
Paused
Generate OTP
Get OTP
Upload File For OTP Verification
Choose File
No file chosen
Verify
23°C
Mostly cloudy
Search
ENG
IN
23:56
30-03-2023


localhost:3000/user/getOTP?
localhost:3000/user/getotp?
Upload An Image With Node.js
How to Upload and Display Imag
localhost:3000/user/getotp?
Paused
{
"message": "OTP sent successfully"
}
Tree
Chart
JSON Input
23°C
Mostly cloudy
Search
ENG
IN
23:55
30-03-2023


localhost:3000/user/getOTP?
localhost:3000/user/upload
Upload An Image With Node.js
How to Upload and Display Imag
localhost:3000/user/upload
Paused
HP Consumer PCs -...
Learn web develop...
Unicode...
Color Hunt - Popul...
<color> - CSS: Casc...
Web Development...
Free vector icons -...
Frontend Mentor |...
UI-Patterns.com
Dribbble - Discover...
{
"message": "OTP Verified Successfully",
"success": true
}
Tree
Chart
JSON Input
23°C
Mostly cloudy
Search
ENG
IN
23:56
30-03-2023


Meet - hbr-mwco-fyi
OTP - kunal.chandra1900@gmail
Custom styled input type file upl
https://mail.google.com/mail/u/5/#inbox/FMfcgzGslkqTGMNRrQFrmJXhFnvtPPst?compose=GTvVlcSMScfqzBmbVLCKHqHHpRzWPnjLKftVqKCcrNrkSnHltQrVQPNLd...
Gmail
Search mail
Compose
Inbox 9,379
Starred
Snoozed
Sent
Drafts 1
More
Labels
1 of 9,856
kunal.chandra1906@gmail.com
You've got an OTP
23:55 (3 minutes ago)
kunal.chandra1906@gmail.com
to me
You've got an OTP
23:55 (2 minutes ago)
One attachment • Scanned by Gmail
Reply
Forward
Project Images
23°C
Partly cloudy
Search
ENG
IN
23:58
30-03-2023

PROJECT OUTCOME AND APPLICABILITY

OUTCOME

A few years back when security was becoming vulnerable and only keeping passwords was not a safe technique, some people came up with the idea of using OTPs to enhance security.

It was a wise option back then, and cyberspace became safer.

But as of now, we came across that this is an old-fashioned system and needs to be replaced with a better one, this is what our project is all about.

PROJECT APPLICABILITY ON REAL-WORLD APPLICATIONS

Our project will be helpful in preventing some forms of identity theft by making sure that even if someone captures the Username and password and then also they won't be able to get the authority to gain access to the user's account.

The security of our project will be stronger than the traditional OTP system as we have encrypted the OTP in the image.

CONCLUSION

OTP service is the easiest way of verifying user information and authenticating online transactions, used by businesses to boost data security and prevent phishing attacks, an OTP service provider ensures timely delivery of OTPs to the right customers via SMSs and voice calls.

That’s why the OTP system must be strongly secure, so in order to make it more protected we have introduced an enhanced OTP algorithm with the combination of traditional OTP system with steganography.

FUTURE ENHANCEMENTS

A simple password doesn't cut it for most systems, especially ones with higher risks or sensitivity attached to them. One of the most effective ways of ensuring authentication is with "Multi-Factor Authentication". It is a method of access control in which a user is granted access only after successfully presenting at least two separate pieces of evidence to an authentication mechanism within the following categories:

- The library we are using to decode, we using the pre defined algorithms and not overriding any of the decoding functions, which makes it a bit generic as it can be used by anyone to decode the same.
- We would hope to optimize the overall time consumption and bring it down to about 4 to 5 minutes per verification.
- Also, for the testing phase we had to locally use a limited set of images so for future enhancement and more randomization we would be including a bigger lot of images to choose from.

- Knowledge Factor: Something you "know": a password or PIN, or an answer to a question
- Possession Factor: Something you "have": a token, credit card or mobile device
- Inherence Factor: Something you "are": biometric data, such as fingerprints

We plan to add this feature to make our project from a single method of authentication currently to this multi-factor mode of authentication.

1. Benefits of Multi-Factor Authentication -
2. Multi-factor authentication increases security with third parties and organizations.
3. It offers a variety of choices to meet your security needs.
4. MFA better controls who has access to your files.
5. MFA helps meet regulatory requirements.

6. It takes away password risks.

REFERENCES

1. Abdul-Wahab,S.A., Al-Alawi,S.M. and El-Zawahry, Patterns of $S0_2$ emission: a refinery case study, *Environmental modeling & software*, 2002, 17, 563-570.

2. Aggarwal A.L, Sivacoumar R. and Goyal SK Air Quality Prediction : influence of model parameters and sensitivity analysis, *Indian Journal of Environmental Protection*, 1997, 17(9), 650-655.

3. https://www.npmjs.com/package/uuidv4

4. https://mailtrap.io/blog/send-emails-with-nodejs/

5. https://www.geeksforgeeks.org/image-steganography-in- cryptography/

6. https://www.npmjs.com/package/bcrypt

XXXXX THE END XXXXX