

Steganography Based Email OTP Verification System



Team Members.

Kunal Chandra

19BCY10028

K

G

Gopal Dandotiya

19BCY10047

Shivank Sharma

19BCY10058

S

J

Jasleen Kaur

19BCY10150

Guide: Dr. Soma Saha

1. Introduction



INTRODUCTION

- The idea of our project is to build a stronger and more modern email OTP verification system that would replace the traditional email OTP verification system using Steganography.
- In the current OTP system, anyone around the user can peek into his device and get access to the OTP, and can use the OTP to harm the individual's reputation.
- A hacker can also break into the network and access the OTP in plain text.

OTP SMS

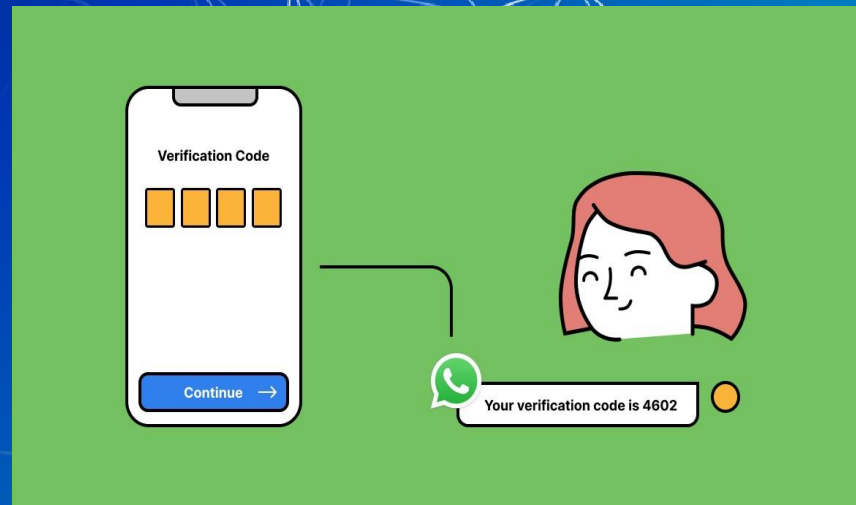


2. Existing work with Limitations

Existing Work

- The existing work is known in the original system, One Time Password, or OTP comes through an SMS/Email.
- In the persisting 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 the privacy of the user here, is being breached.

3. Proposed Work



PROPOSED WORK

- Our project will mainly target email-based OTP verification systems.
- The application will start with generating an OTP consisting of 4–6 characters, then it will be transferred to the encryption and decryption module where the OTP will be encrypted into an image.
- This image will then be sent to the user through email using a NodeJS library named **NodeMailer**.
- User can then download the image and upload it into the uploading section.
- 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.



4.

The novelty of the Project

- 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 steganographically 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.

5. Real Time Usage

The background of the slide is a deep blue gradient that transitions to a lighter cyan at the bottom right. Overlaid on this background is a complex, abstract network of thin white lines connecting small white dots. These dots and lines form various geometric shapes, including triangles, quadrilaterals, and larger, more intricate polygons, creating a sense of a dynamic, interconnected structure.

REAL TIME USAGE

- Our web app can be used as a third-party application by various organizations to send OTPs through email to their users.
- Our project will be an enhanced version of the traditional email OTP system as we have encrypted the OTP in an image hence we will be providing them with improved security using the Steganography-based email OTP verification system.




6. Hardware & Software Requirements

Hardware Requirements

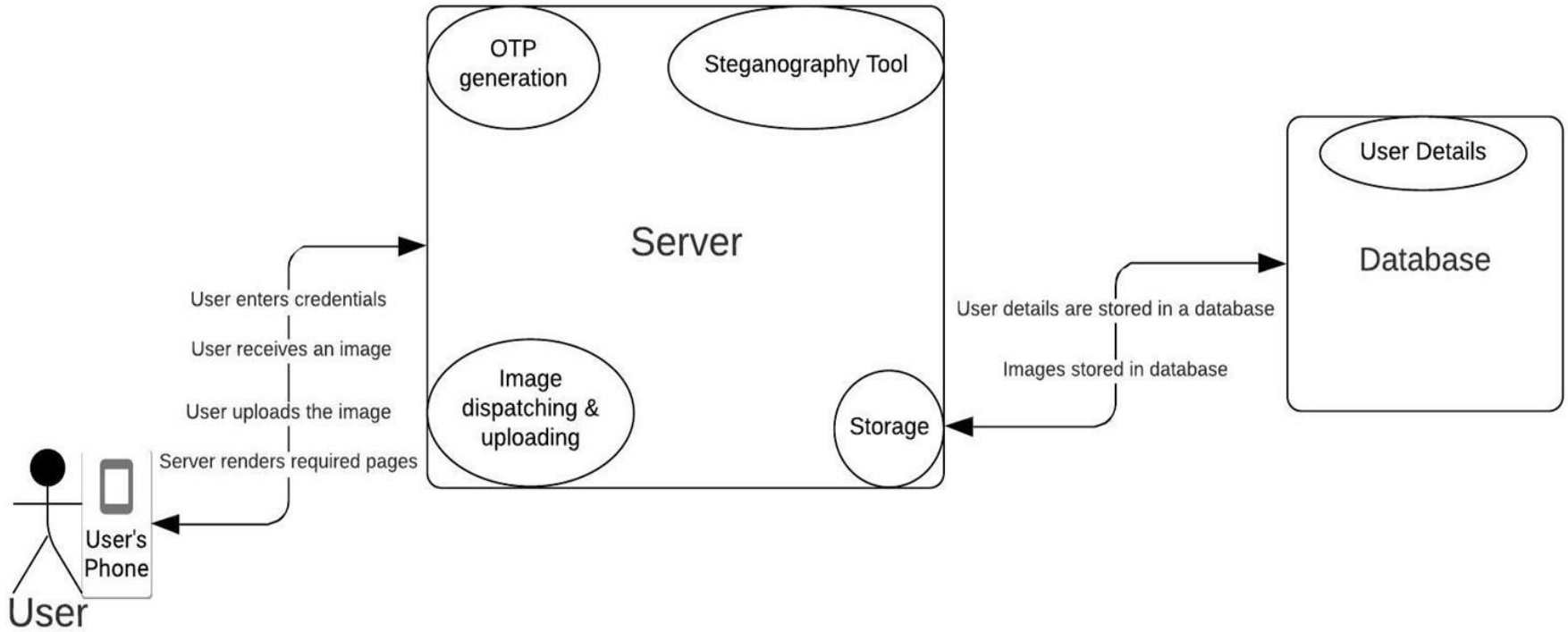
- ❑ x86 64-bit CPU (Intel / AMD architecture)
- ❑ 4 GB RAM.
- ❑ 5 GB free disk space

Software Requirements

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



7. Overall System Architecture Design



8. Literature Review



LITERATURE REVIEW

- 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 due to this 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.



9.

Modules Description

(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.

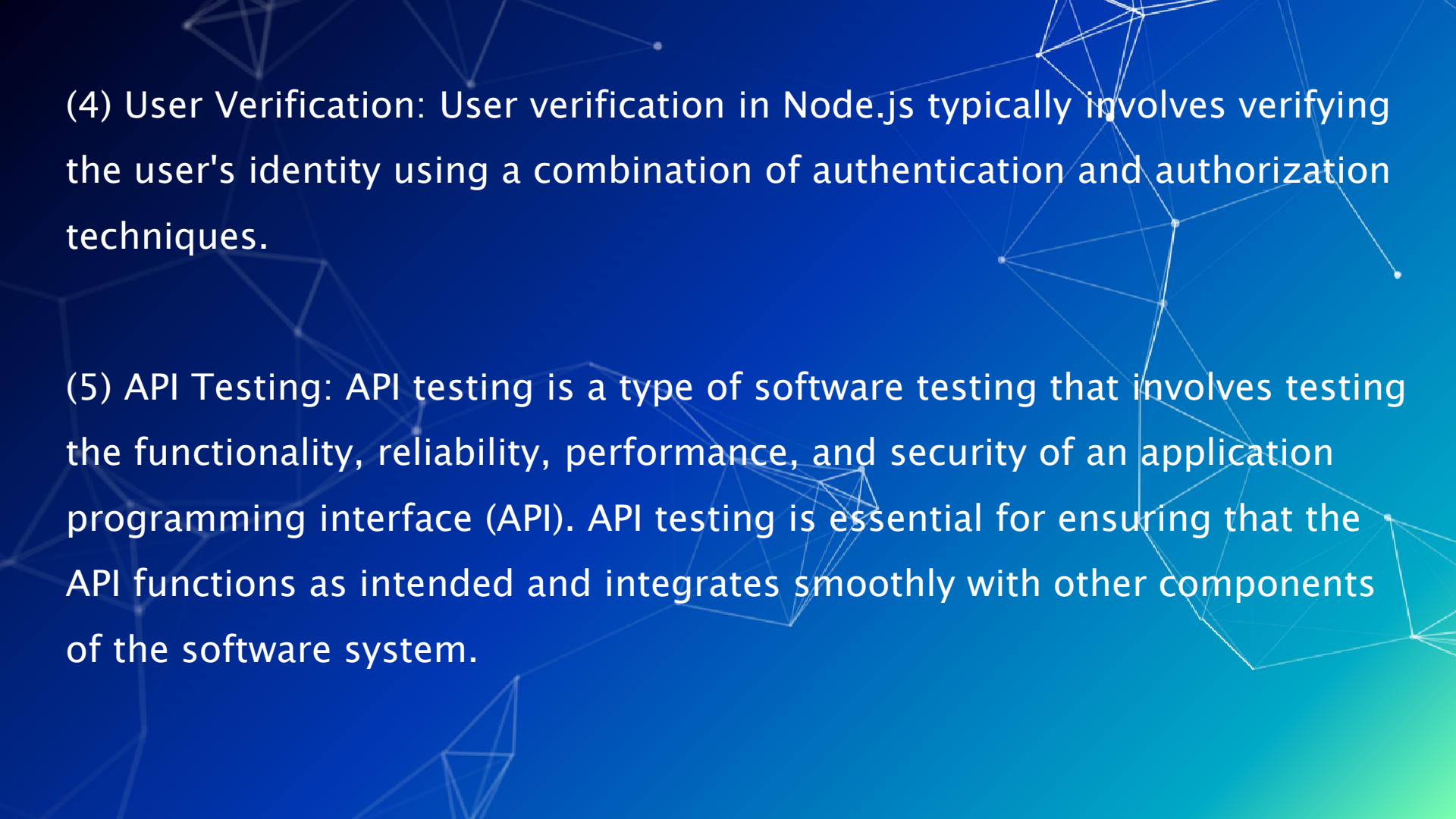
The module includes parameters or settings to specify the database type, hostname or IP address, port number, authentication credentials, encryption options, and other configuration options.

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.

(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.



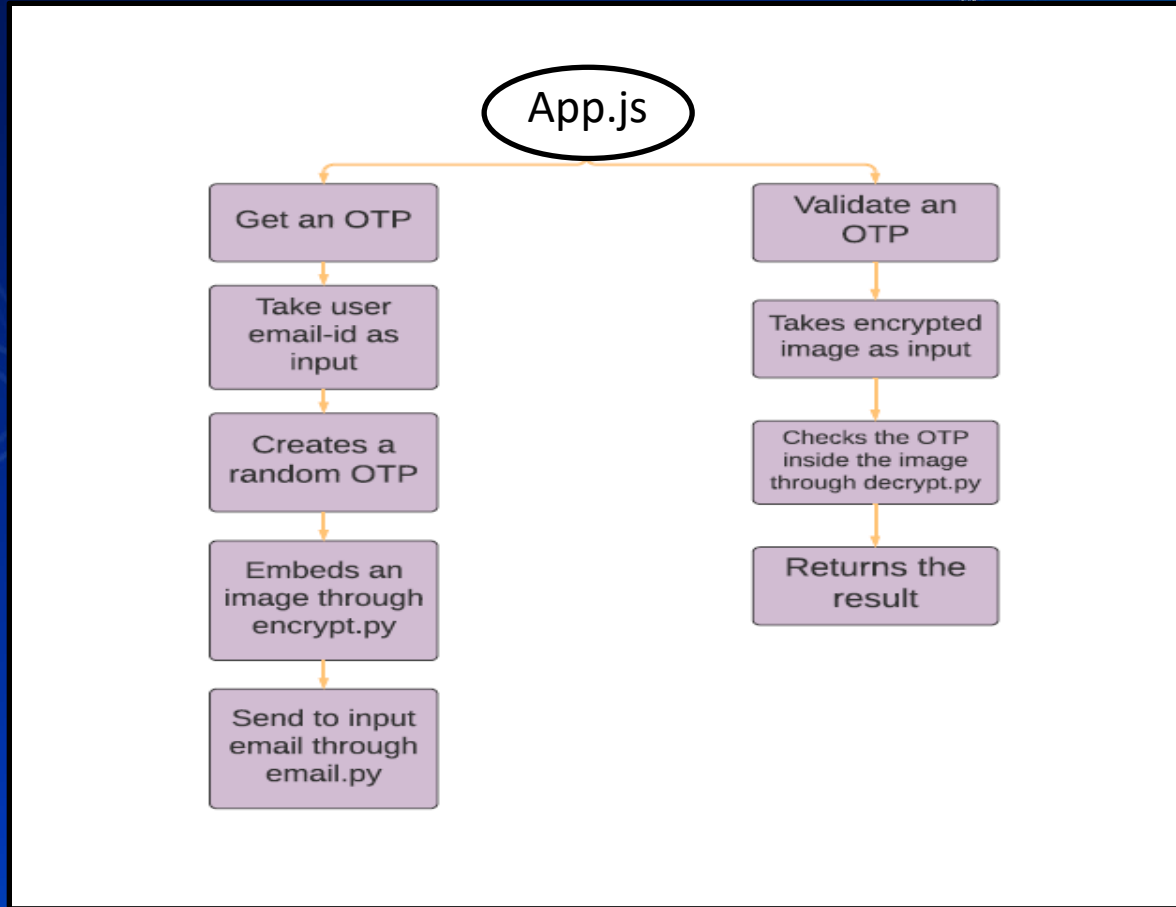
(4) User Verification: User verification in Node.js typically involves verifying the user's identity using a combination of authentication and authorization techniques.

(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.



10. Module Workflow Explanation

MODULES Work Flow



11. Implementation & Coding



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

This screenshot shows the Visual Studio Code interface with the REST Client extension. The active file is `login` in the `login - capstone_project` workspace. The request is a POST to `http://localhost:3000/auth/login`. The response status is 200 OK, with a size of 0 Bytes and a time of 108 ms. The response body is a JSON object: `{ "message": "User logged in successfully", "success": true }`. The JSON content is displayed in the left pane, and the response is shown in the right pane. The terminal at the bottom shows the Node.js server running on port 3000.

```
POST http://localhost:3000/auth/login Send
```

Status: 200 OK Size: 0 Bytes Time: 108 ms

Query Headers Auth Body Tests Pre Run New

Json Xml Text Form Form-encode GraphQL Binary

Json Content Format

```
1 {
2   "email": "abc@gmail.com",
3   "password": "123456789"
4 }
```

Response Headers Cookies Results Docs

```
1 {
2   "message": "User logged in successfully",
3   "success": true
4 }
```

Copy

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
```

This screenshot shows the Visual Studio Code interface with the REST Client extension. The active file is `login` in the `login - capstone_project` workspace. The request is a POST to `http://localhost:3000/auth/login`. The response status is 201 Created, with a size of 55 Bytes and a time of 89 ms. The response body is a JSON object: `{ "message": "Invalid email or password", "success": false }`. The JSON content is displayed in the left pane, and the response is shown in the right pane. The terminal at the bottom shows the Node.js server running on port 3000.

```
POST http://localhost:3000/auth/login Send
```

Status: 201 Created Size: 55 Bytes Time: 89 ms

Query Headers Auth Body Tests Pre Run New

Json Xml Text Form Form-encode GraphQL Binary

Json Content Format

```
1 {
2   "email": "abc@gmail.com",
3   "password": "12345678"
4 }
```

Response Headers Cookies Results Docs

```
1 {
2   "message": "Invalid email or password",
3   "success": false
4 }
```

Copy

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
```


getSecret - capstone_project - Visual Studio Code

GET http://localhost:3000/user/secret Send

Query Headers Auth Body Tests Pre Run New

Query Parameters

parameter value

Status: 200 OK Size: 49 Bytes Time: 7 ms

Response Headers Cookies Results Docs

```
1 {
2   "message": "reached secret route",
3   "success": true
4 }
```

Copy

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,wjs,json
[nodemon] starting `node app.js`
Server listening on port 3000
connected to database
```

Go Live



getSecret - capstone_project - Visual Studio Code

GET http://localhost:3000/user/secret Send

Query Headers Auth Body Tests Pre Run New

Query Parameters

parameter value

Status: 500 Internal Server Error Size: 126 Bytes Time: 6 ms

Response Headers Cookies Results Docs

```
1 {
2   "message": "user not authorized, internal server error",
3   "error": {
4     "name": "JsonWebTokenError",
5     "message": "jwt must be provided"
6   }
7 }
```

Copy

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,wjs,json
[nodemon] starting `node app.js`
Server listening on port 3000
connected to database
```

Go Live

Removed All Cookies...

login - capstone_project - Visual Studio Code

JS app.jsJS db.jsTC registerTC loginJS userRoutes.jsJS userController.js.env

POSThttp://localhost:3000/auth/loginSend

QueryHeaders3AuthBody1TestsPre RunNew

JsonXmlTextFormForm-encodeGraphQLBinary

Json ContentFormat

```
1 {
2   "email": "abc@gmail.com",
3   "password": "123456789"
4 }
```

Status: 200 OKSize: 0 BytesTime: 89 ms

ResponseHeaders7Cookies1ResultsDocs

Request Url

```
http://localhost:3000/auth/login
```

Response Cookies

Clear All

Name	Value
secret	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiaWVWjQGdtYWlsLmNvbSIsImhhdCI6MTY3NTE1NDQzM30.FoKi1ts0B7a3GgFJKWpr9gl85mT0xol5yV1vUSVq0Ak

TERMINALDEBUG CONSOLEPROBLEMSOUTPUT

```
[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
█
```

+ - ^ x

bash

bash

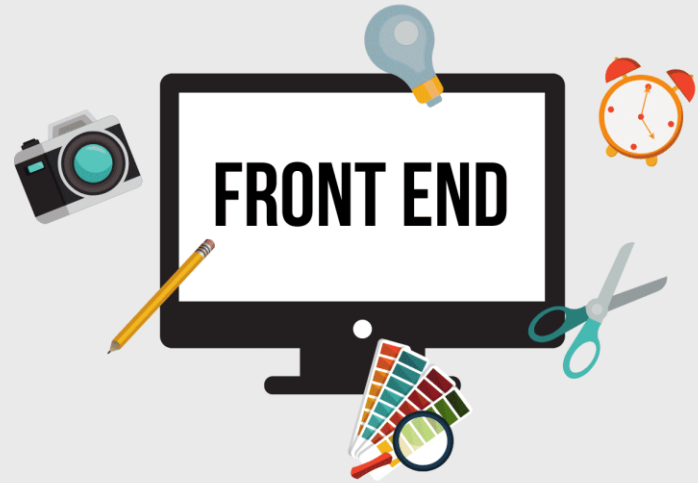
81°FPartly sunny

Search

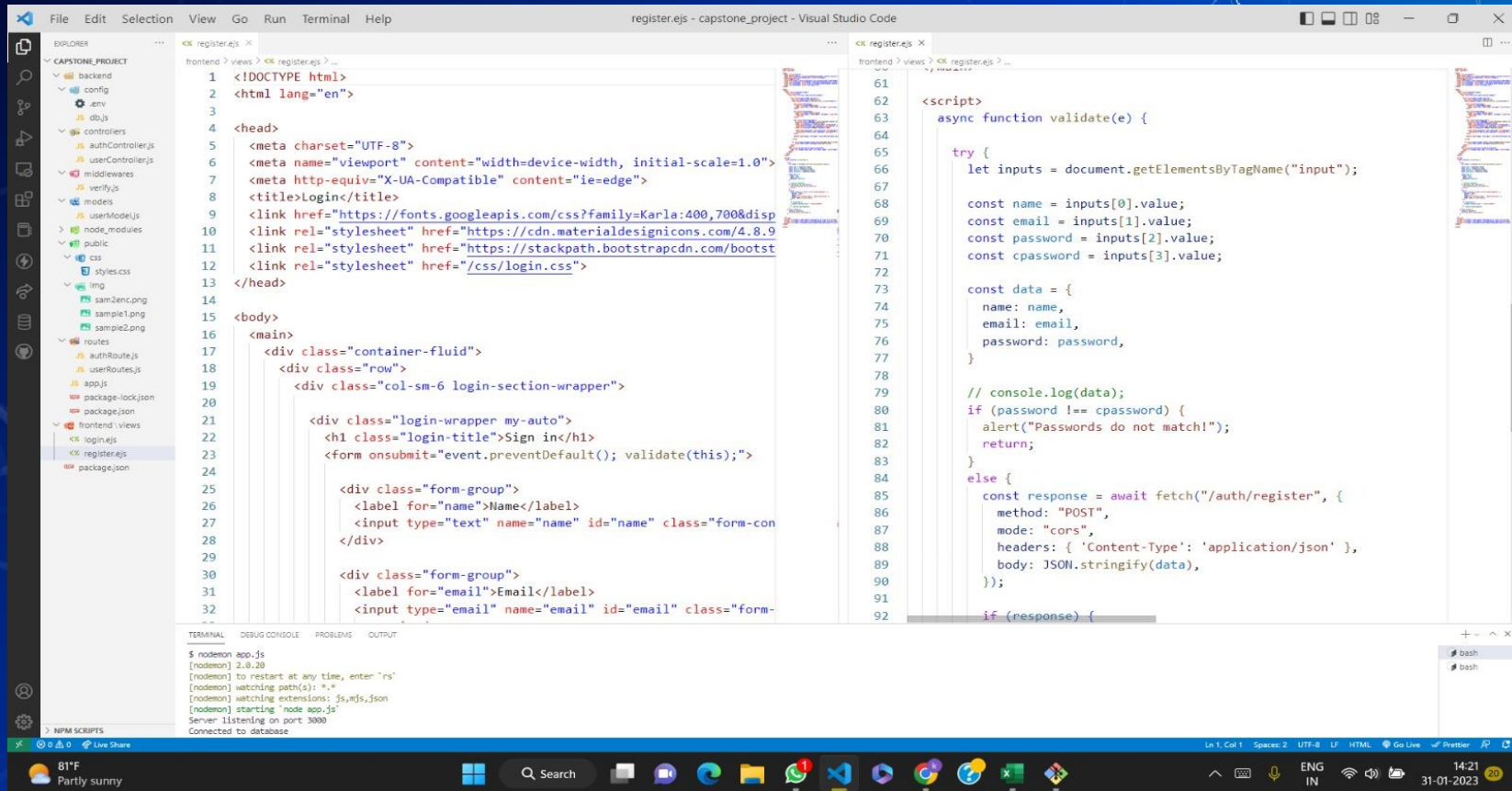
Go Live

14:1031-01-2023

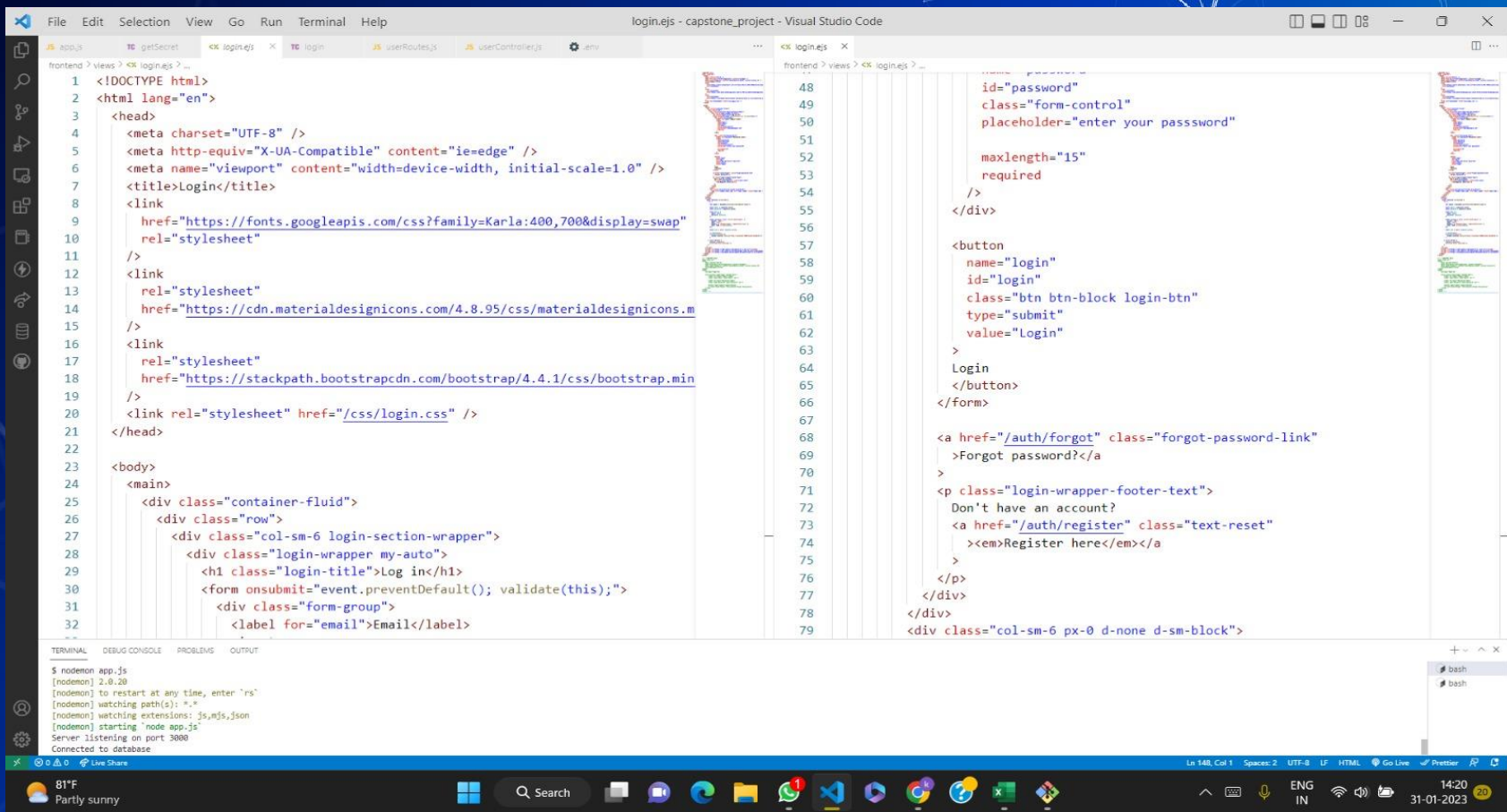
8. Front end



Signup- Code



Login- Code



```
File Edit Selection View Go Run Terminal Help login.ejs - capstone_project - Visual Studio Code

frontend > views > login.ejs
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <meta http-equiv="X-UA-Compatible" content="ie=edge" />
6   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7   <title>Login</title>
8   <link
9     href="https://fonts.googleapis.com/css?family=Karla:400,700&display=swap"
10    rel="stylesheet"
11  />
12  <link
13    rel="stylesheet"
14    href="https://cdn.materialdesignicons.com/4.8.95/css/materialdesignicons.min"
15  />
16  <link
17    rel="stylesheet"
18    href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min"
19  />
20  <link rel="stylesheet" href="/css/login.css" />
21 </head>
22
23 <body>
24   <main>
25     <div class="container-fluid">
26       <div class="row">
27         <div class="col-sm-6 login-section-wrapper">
28           <div class="login-wrapper my-auto">
29             <h1 class="login-title">Log in</h1>
30             <form onsubmit="event.preventDefault(); validate(this);">
31               <div class="form-group">
32                 <label for="email">Email</label>
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
id="password"
class="form-control"
placeholder="enter your password"

maxlength="15"
required
/>
</div>

<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?</a>
</div>
<div>
  <p class="login-wrapper-footer-text">
    Don't have an account?
    <a href="/auth/register" class="text-reset">
      >Register here</a>
  </p>
</div>
</div>
<div class="col-sm-6 px-0 d-none d-sm-block">
```

```
TERMINAL DEBUG CONSOLE PROBLEMS OUTPUT
$ node 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

Sign Up

Name

Full Name

Email

email@example.com

Password should contain atleast one special character eg. @ or \$ etc, an upper case letter

Password

Create your password

Repeat Password

Confirm your password

Sign-up

Already have an account? [Login here](#)



localhost:3000/user/getOTP? x Login x Upload An Image With Nodejs - x How to Upload and Display Imag x +

localhost:3000/auth/login

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... »

Log in

Email

email@example.com

Password

enter your password

Login

Don't have an account? [Register here](#)

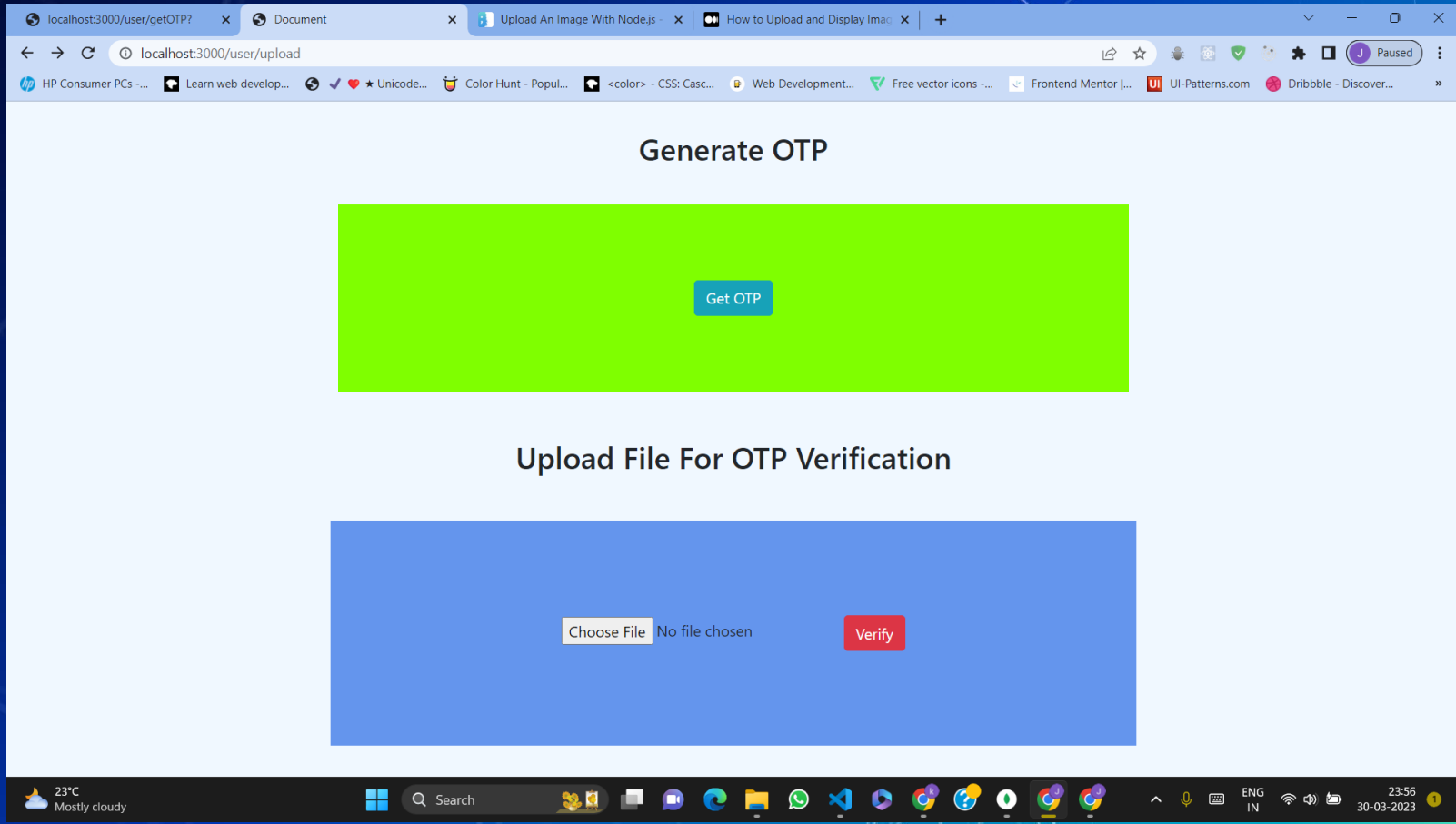
23°C Mostly cloudy

Search

📁 📧 📅 📌 📎 📏 📐 📑 📔 📕 📖 📗 📘 📙 📚

ENG IN 23:56 30-03-2023

Generate OTP



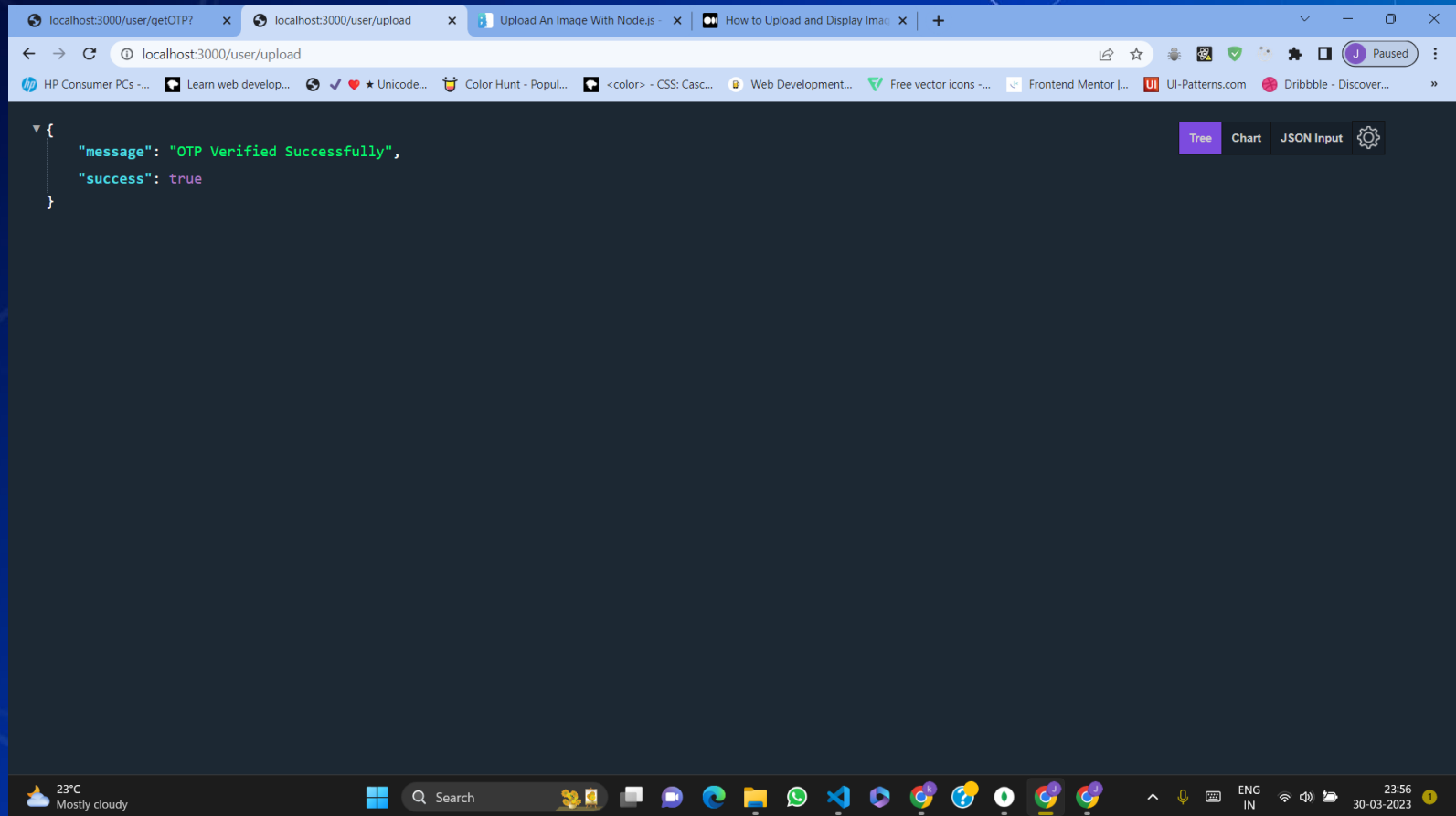
Sent Successfully

The screenshot shows a web browser window with the address bar displaying `localhost:3000/user/getotp?`. The browser's developer tools are open, showing a JSON response in the console:

```
{  
  "message": "OTP sent successfully"  
}
```

The browser's taskbar at the bottom shows the system clock as 23:55 on 30-03-2023, with a temperature of 23°C and weather status of 'Mostly cloudy'.

Verification Successful

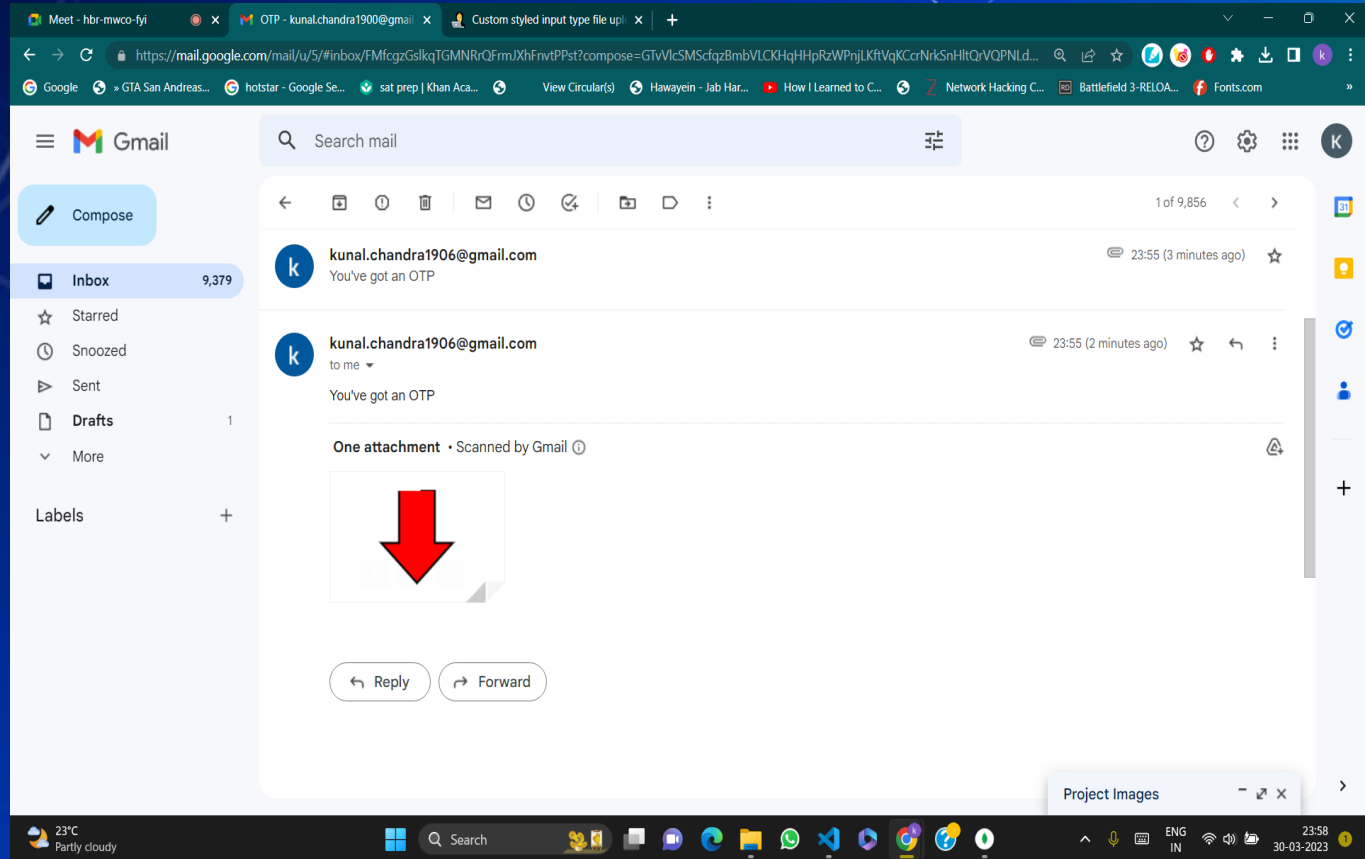


The screenshot shows a web browser window with the address bar displaying `localhost:3000/user/upload`. The page content shows a JSON response in a dark-themed editor:

```
{
  "message": "OTP Verified Successfully",
  "success": true
}
```

The browser's tab bar includes several open tabs: `localhost:3000/user/getOTP?`, `localhost:3000/user/upload`, `Upload An Image With Nodejs`, and `How to Upload and Display Image`. The browser's address bar and tab bar show various bookmarks and extensions. The Windows taskbar at the bottom displays the system clock as 23:56 on 30-03-2023, along with the weather (23°C, Mostly cloudy) and several application icons.

14. Email Notification



16. 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.

8.



REFERENCES

- <https://www.npmjs.com/package/uuidv4>
- <https://mailtrap.io/blog/send-emails-with-nodejs/>
- <https://www.geeksforgeeks.org/image-steganography-in-cryptography/>
- <https://www.npmjs.com/package/bcrypt>



Thank
you

