

# **TRF**

## **ROBOSPARK 2021**

### **Final Task**

#### **Authentication API with JWT**

#### **(Web Development)**

**Date: 13/10/21**

**Project Mentor: Aditya Dhotarkar.**

## **Group Members:**

<b>PRN No.</b>	<b>Name</b>	<b>A.Y</b>
12010641	Siddhant Pawar	S.Y
12010374	Vedant Bhosle	S.Y
12010405	Aniket Kulkarni	S.Y

## **Project Github Link:**

[https://github.com/RoboSpark-2021/robospark-2021-FT-Authentication API with JWT](https://github.com/RoboSpark-2021/robospark-2021-FT-Authentication-API-with-JWT)

## **Project Workflow:**

- Project consists of two phase, Phase1 and Phase 2.
- Phase 1 consists of sub questions as mentioned below:
  - i) What is JWT?
  - ii) Create a Node.js server and connect your database.
  - iii) Create user model and route.
  - iv) Implement register and login functionality.
  - v) Create middleware for authentication.
- Sub question one and fifth were done by all members.
- While sub question two was done by Siddhant Pawar, third by Vedant Bhosle, fourth by Aniket Kulkarni.
- Code was done on common platform.

## Problems Faced:

- As our project primarily focuses on backend, code part was restricted.
- There was problem regarding generating 'JWT token'.
- Implementation of login page and connecting it was difficult task.
- There were multiple problems regarding cookies.

## JWT:

**JSON Web Token** (JWT) is a means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is digitally signed using JSON Web Signature (JWS) and/or encrypted using JSON Web Encryption (JWE).

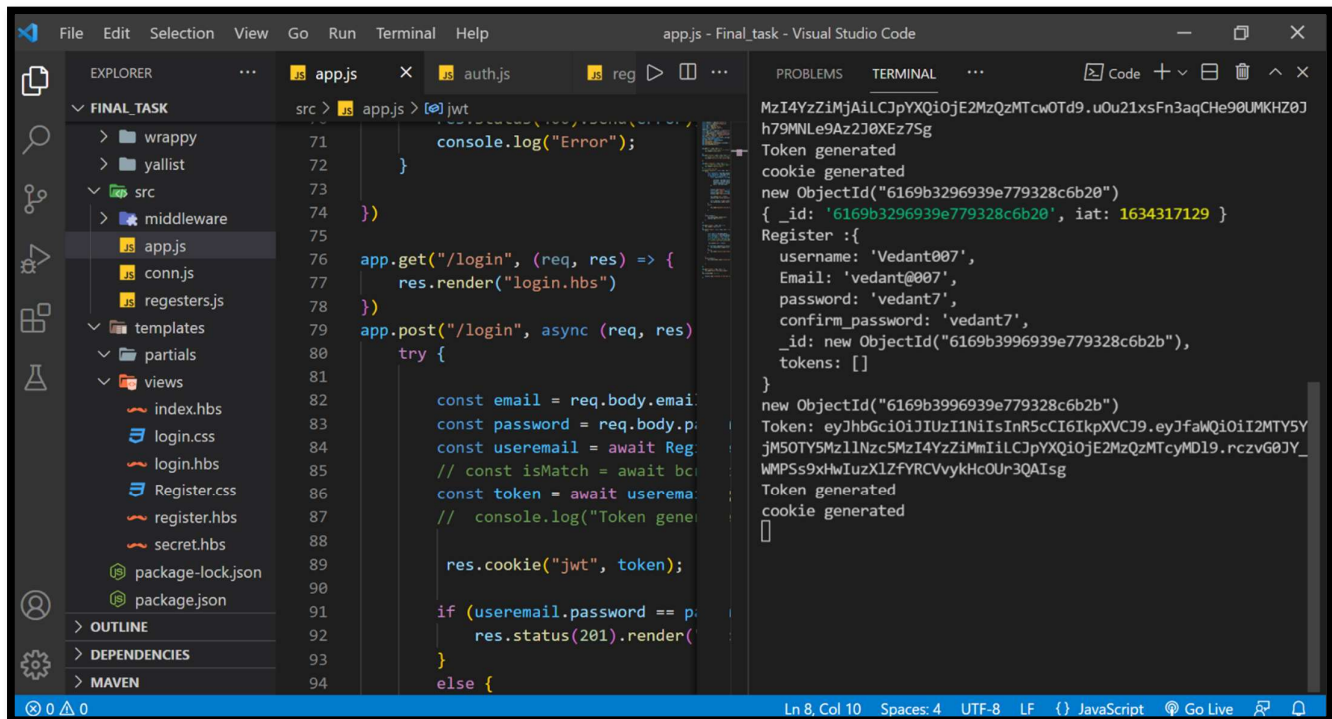
## Alternative Solutions:

Instead of using JWT we can do following points mentioned below:

- Use of **OAuth2**. It is an authentication protocol that allows you to approve one application interacting with another on your behalf without giving away your password.
- Use of **Passport**. Passport is **authentication middleware for Node.js**. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application.
- **Spring Security**. It is a Java/Java EE framework that provides authentication, authorization and other security features for enterprise applications.
- **Auth0**. It is a flexible, drop-in **solution to add authentication and authorization services to your applications**.

# Code Snippet ss:

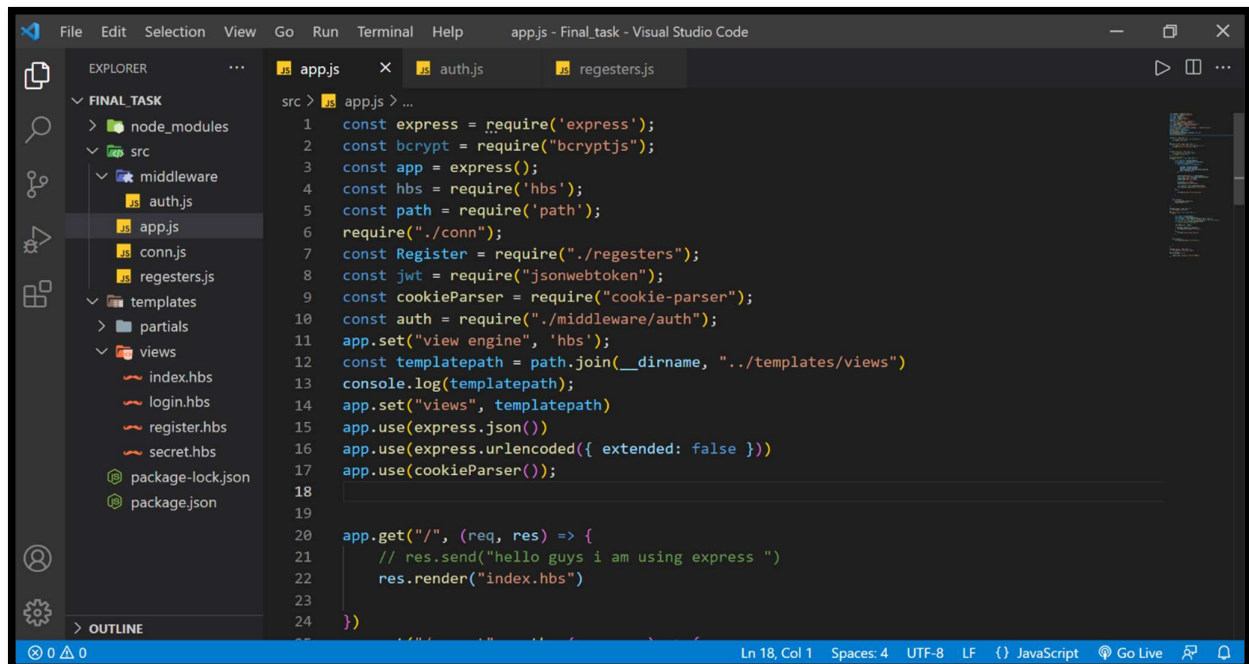
## App.js



The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying a project structure under 'FINAL\_TASK'. The 'src' directory contains 'app.js', 'conn.js', and 'regesters.js'. The 'templates' directory contains 'partials' and 'views'. The 'views' directory contains 'index.hbs', 'login.hbs', 'Register.hbs', 'register.hbs', and 'secret.hbs'. The 'app.js' file is open in the editor, showing a login route and a POST handler. The terminal on the right displays the output of the application, including token generation and cookie setting.

```
src > app.js > jwt
71 console.log("Error");
72 }
73
74 })
75
76 app.get("/login", (req, res) => {
77   res.render("login.hbs")
78 })
79 app.post("/login", async (req, res)
80   try {
81
82     const email = req.body.email;
83     const password = req.body.p
84     const useremail = await Reg
85     // const isMatch = await bcr
86     const token = await userema
87     // console.log("Token gene
88
89     res.cookie("jwt", token);
90
91
92     if (useremail.password == p
93       res.status(201).render(
94     }
95   } else {
```

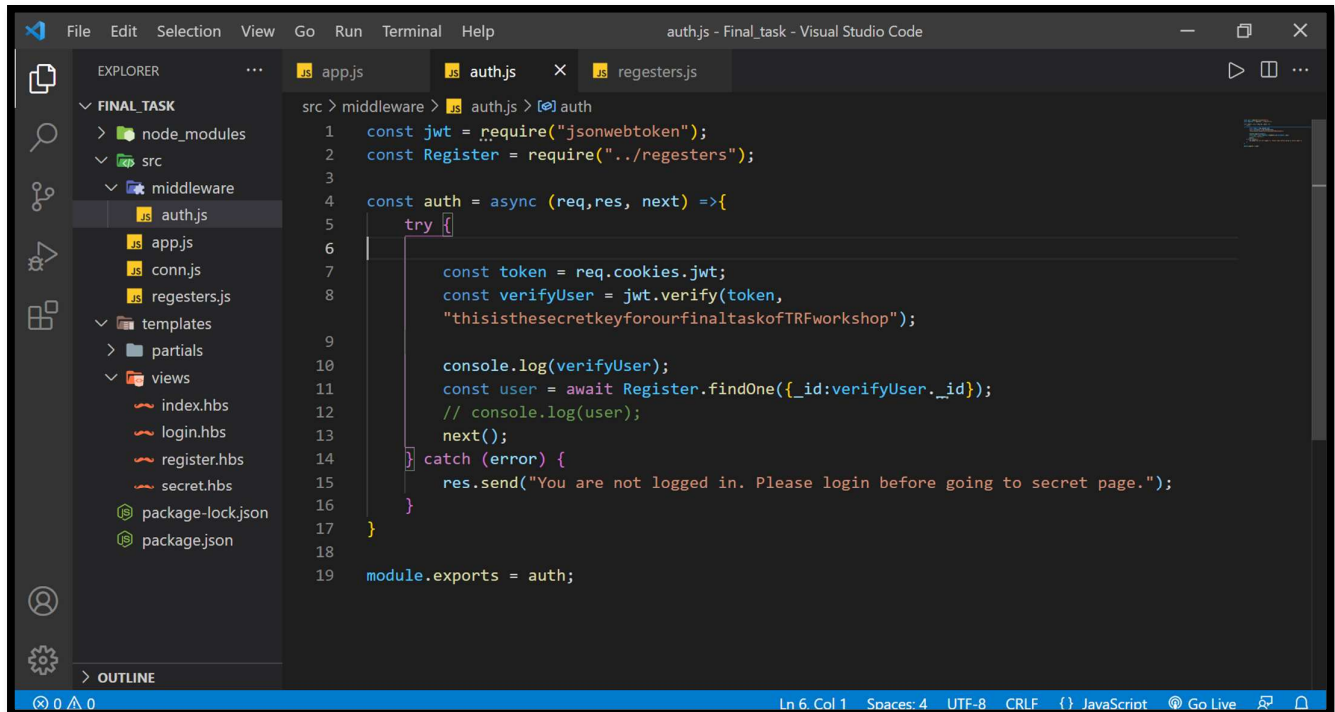
```
MzI4YzZiMjAiIjCjYXQjE2MzQzMWOTd9.uOu21xsFn3aqChe90UMKH20J
h79MNLe9Az2J0XEz7Sg
Token generated
cookie generated
new ObjectId("6169b3296939e779328c6b20")
{ _id: '6169b3296939e779328c6b20', iat: 1634317129 }
Register :{
  username: 'Vedant007',
  Email: 'vedant007',
  password: 'vedant7',
  confirm_password: 'vedant7',
  _id: new ObjectId("6169b3996939e779328c6b2b"),
  tokens: []
}
new ObjectId("6169b3996939e779328c6b2b")
Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfawQjE2MTY5Y_
jM5OTY5MzllNzc5MzI4YzZiMmIiLCJpYXQjE2MzQzMWOTd9.rczvG0JY_
WMP5s9xHwIuzX1ZfYRCVvykHc0Ur3QAIsG
Token generated
cookie generated
```



The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying a project structure under 'FINAL\_TASK'. The 'node\_modules' directory is expanded, showing 'src', 'middleware', 'app.js', 'conn.js', and 'regesters.js'. The 'app.js' file is open in the editor, showing the main application setup and routes.

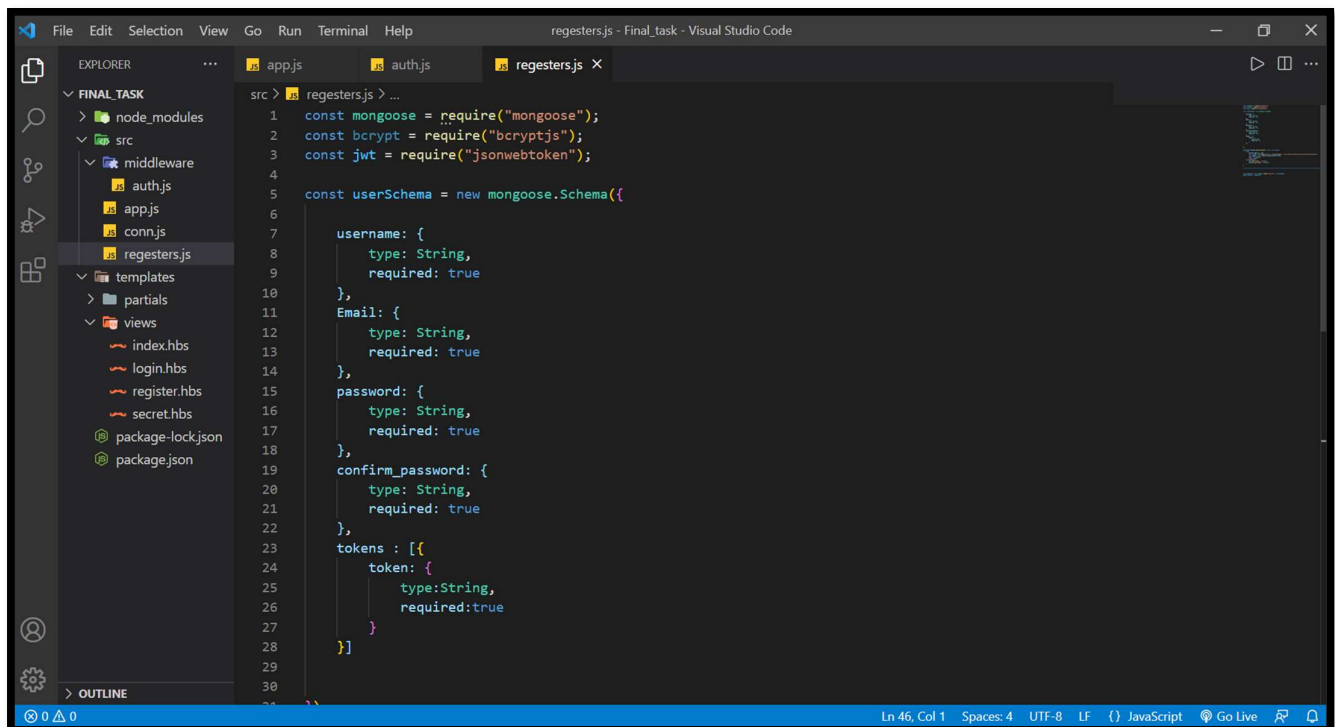
```
src > app.js > ...
1 const express = require('express');
2 const bcrypt = require("bcryptjs");
3 const app = express();
4 const hbs = require('hbs');
5 const path = require('path');
6 require("./conn");
7 const Register = require("./regesters");
8 const jwt = require("jsonwebtoken");
9 const cookieParser = require("cookie-parser");
10 const auth = require("./middleware/auth");
11 app.set("view engine", 'hbs');
12 const templatepath = path.join(__dirname, "../templates/views")
13 console.log(templatepath);
14 app.set("views", templatepath)
15 app.use(express.json())
16 app.use(express.urlencoded({ extended: false }))
17 app.use(cookieParser());
18
19
20 app.get("/", (req, res) => {
21   // res.send("hello guys i am using express ")
22   res.render("index.hbs")
23 }
24 )
```

# Auth.js



```
src > middleware > auth.js > auth
1  const jwt = require("jsonwebtoken");
2  const Register = require("../regesters");
3
4  const auth = async (req,res, next) =>{
5    try {
6
7      const token = req.cookies.jwt;
8      const verifyUser = jwt.verify(token,
9        "thisisthesecretkeyforourfinaltaskofTRFworkshop");
10
11     console.log(verifyUser);
12     const user = await Register.findOne({_id:verifyUser._id});
13     // console.log(user);
14     next();
15   } catch (error) {
16     res.send("You are not logged in. Please login before going to secret page.");
17   }
18
19   module.exports = auth;
```

# Register.js



```
regesters.js - Final_task - Visual Studio Code
src > regesters.js > ...
1  const mongoose = require("mongoose");
2  const bcrypt = require("bcryptjs");
3  const jwt = require("jsonwebtoken");
4
5  const userSchema = new mongoose.Schema({
6
7    username: {
8      type: String,
9      required: true
10    },
11    Email: {
12      type: String,
13      required: true
14    },
15    password: {
16      type: String,
17      required: true
18    },
19    confirm_password: {
20      type: String,
21      required: true
22    },
23    tokens : [{
24      token: {
25        type:String,
26        required:true
27      }
28    }]
29
30  })
```

The screenshot shows the Visual Studio Code editor with the 'regesters.js' file open. The Explorer sidebar on the left shows the project structure: 'FINAL\_TASK' containing 'node\_modules', 'src' (with 'middleware', 'templates', and 'views' subfolders), and 'package-lock.json' and 'package.json'. The 'regesters.js' file is selected in the Explorer. The main editor area shows the following code:

```
src > regesters.js > ...
25     type:String,
26     required:true
27   }
28 ]
29
30
31 })
32
33 userSchema.methods.generateToken = async function(){
34   try{
35     console.log(this._id);
36     const token = jwt.sign({_id:this._id.toString()}, "thisisthesecretkeyforourfinaltaskofTRFworkshop");
37     this.tokens = this.tokens.concat({token:token})
38     await this.save();
39     return token;
40   }catch(error){
41     res.send("ERROR: "+ error);
42     console.log("ERROR: "+ error);
43   }
44 }
45
46
47
48
49
50 const Register = new mongoose.model("Register", userSchema);
51 module.exports = Register;
```

The status bar at the bottom indicates 'Ln 46, Col 1', 'Spaces: 4', 'UTF-8', 'LF', 'JavaScript', and 'Go Live'.

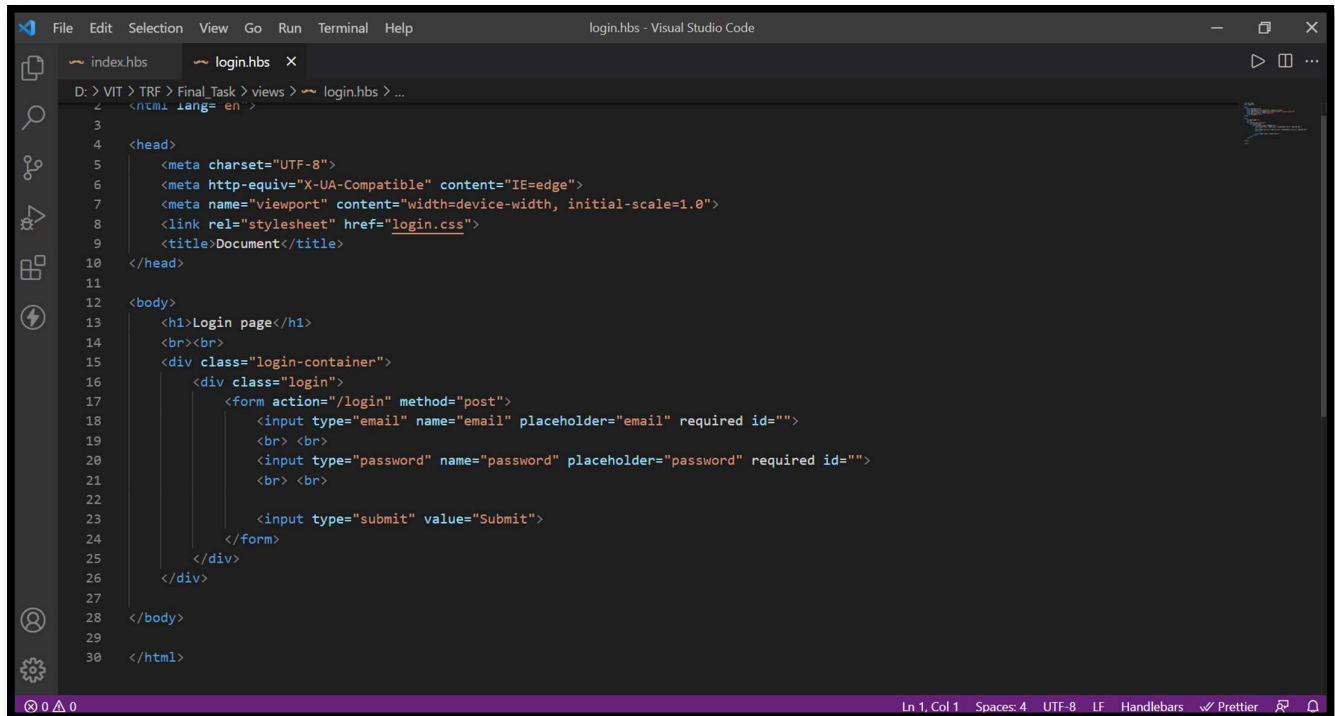
## Conn.js

The screenshot shows the Visual Studio Code editor with the 'conn.js' file open. The Explorer sidebar on the left shows the project structure: 'FINAL\_TASK' containing 'node\_modules', 'src' (with 'middleware', 'templates', and 'views' subfolders), and 'package-lock.json' and 'package.json'. The 'conn.js' file is selected in the Explorer. The main editor area shows the following code:

```
src > conn.js > then() callback
1 const mongoose = require('mongoose')
2 mongoose.connect("mongodb://localhost:27017/userdetail", {
3   useNewUrlParser: true, useUnifiedTopology: true
4 }).then(() =>|
5   console.log("connection succesfull.....")
6 ).catch((err) =>
7   console.log("not succesfull")
8 )
```

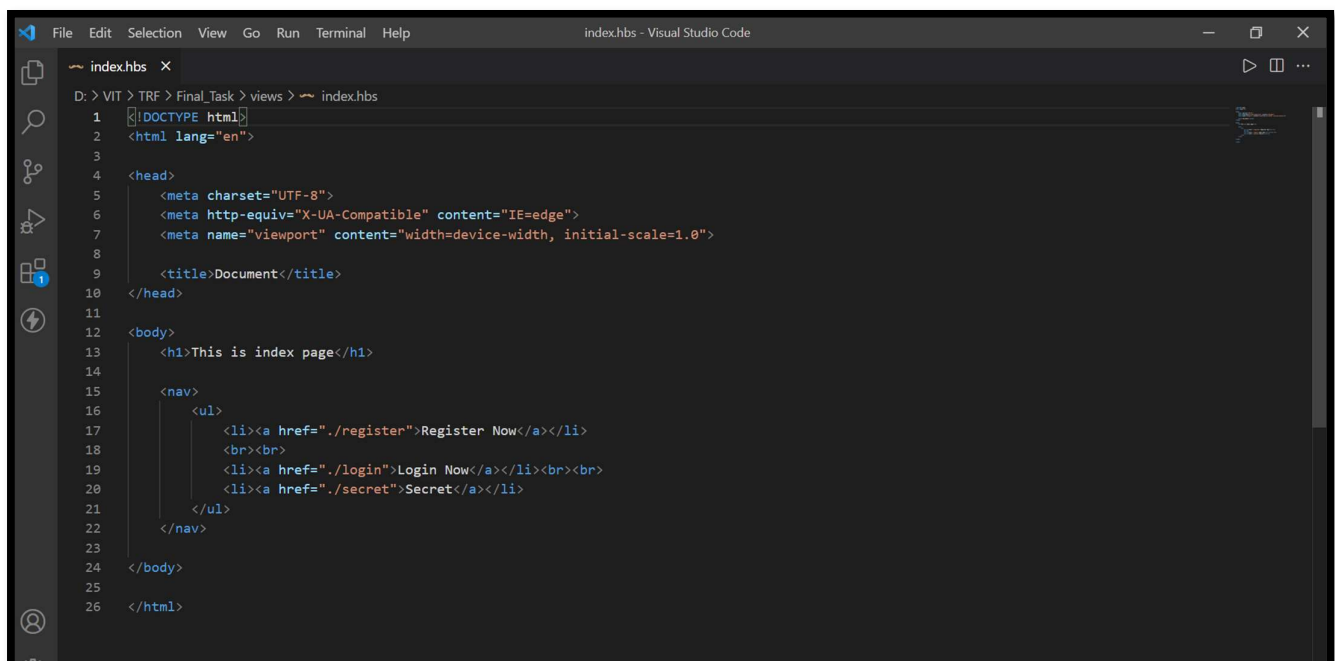
The status bar at the bottom indicates 'Ln 4, Col 14', 'Spaces: 4', 'UTF-8', 'LF', 'JavaScript', and 'Go Live'.

# Login.hbs



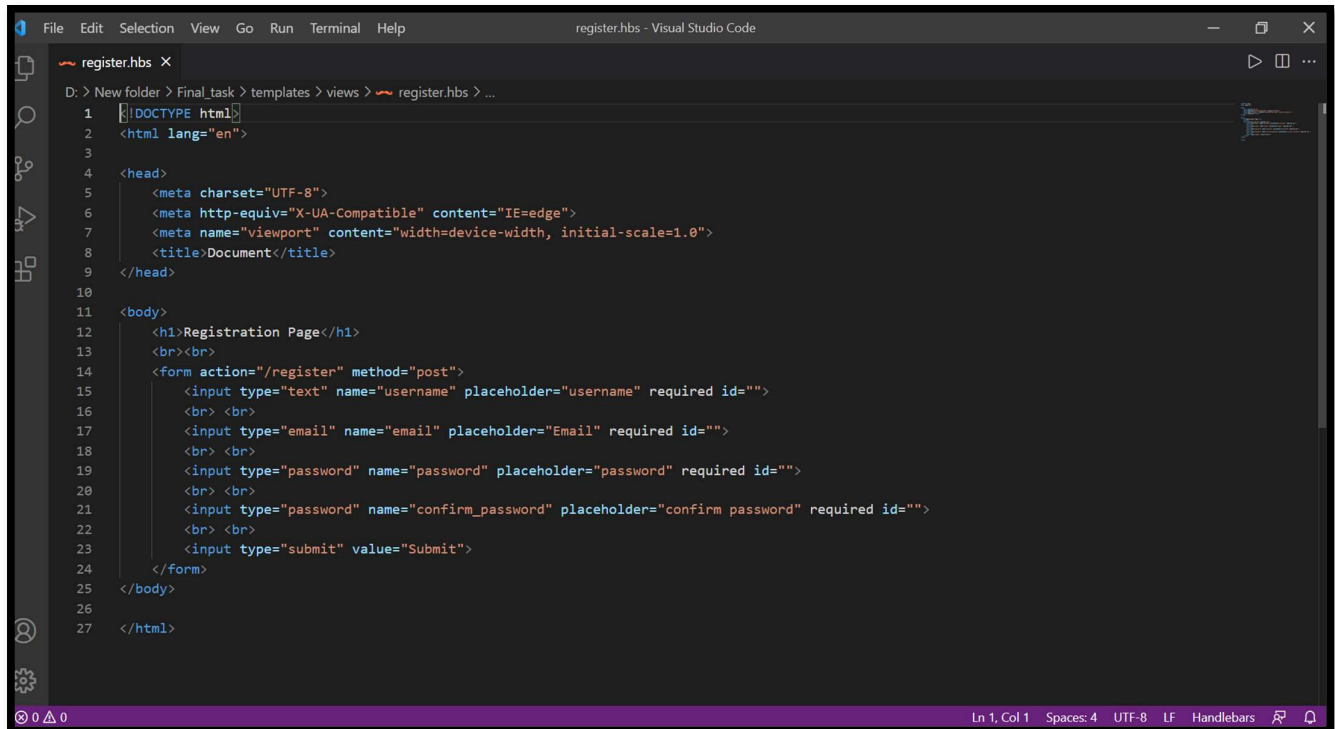
```
File Edit Selection View Go Run Terminal Help login.hbs - Visual Studio Code
index.hbs login.hbs X
D:\> VIT > TRF > Final_Task > views > login.hbs > ...
1 <html lang="en">
2
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <link rel="stylesheet" href="login.css">
9   <title>Document</title>
10 </head>
11
12 <body>
13   <h1>Login page</h1>
14   <br><br>
15   <div class="login-container">
16     <div class="login">
17       <form action="/login" method="post">
18         <input type="email" name="email" placeholder="email" required id="">
19         <br> <br>
20         <input type="password" name="password" placeholder="password" required id="">
21         <br> <br>
22
23         <input type="submit" value="Submit">
24       </form>
25     </div>
26   </div>
27
28 </body>
29
30 </html>
Ln 1, Col 1 Spaces: 4 UTF-8 LF Handlebars Prettier
```

# Index.hbs



```
File Edit Selection View Go Run Terminal Help index.hbs - Visual Studio Code
index.hbs X
D:\> VIT > TRF > Final_Task > views > index.hbs
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8
9   <title>Document</title>
10 </head>
11
12 <body>
13   <h1>This is index page</h1>
14
15   <nav>
16     <ul>
17       <li><a href="/register">Register Now</a></li>
18       <br><br>
19       <li><a href="/login">Login Now</a></li><br><br>
20       <li><a href="/secret">Secret</a></li>
21     </ul>
22   </nav>
23
24 </body>
25
26 </html>
```

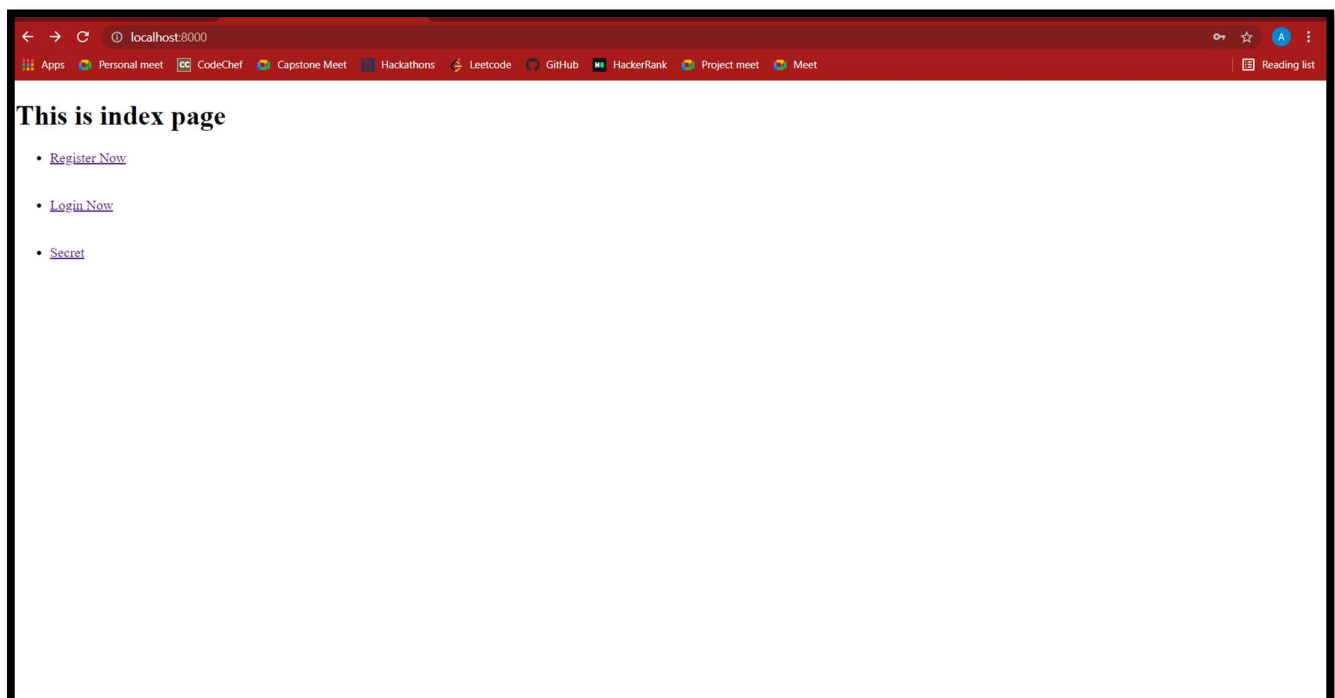
# Register.hbs



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>Document</title>
9 </head>
10
11 <body>
12   <h1>Registration Page</h1>
13   <br><br>
14   <form action="/register" method="post">
15     <input type="text" name="username" placeholder="username" required id="">
16     <br> <br>
17     <input type="email" name="email" placeholder="Email" required id="">
18     <br> <br>
19     <input type="password" name="password" placeholder="password" required id="">
20     <br> <br>
21     <input type="password" name="confirm_password" placeholder="confirm password" required id="">
22     <br> <br>
23     <input type="submit" value="Submit">
24   </form>
25 </body>
26
27 </html>
```

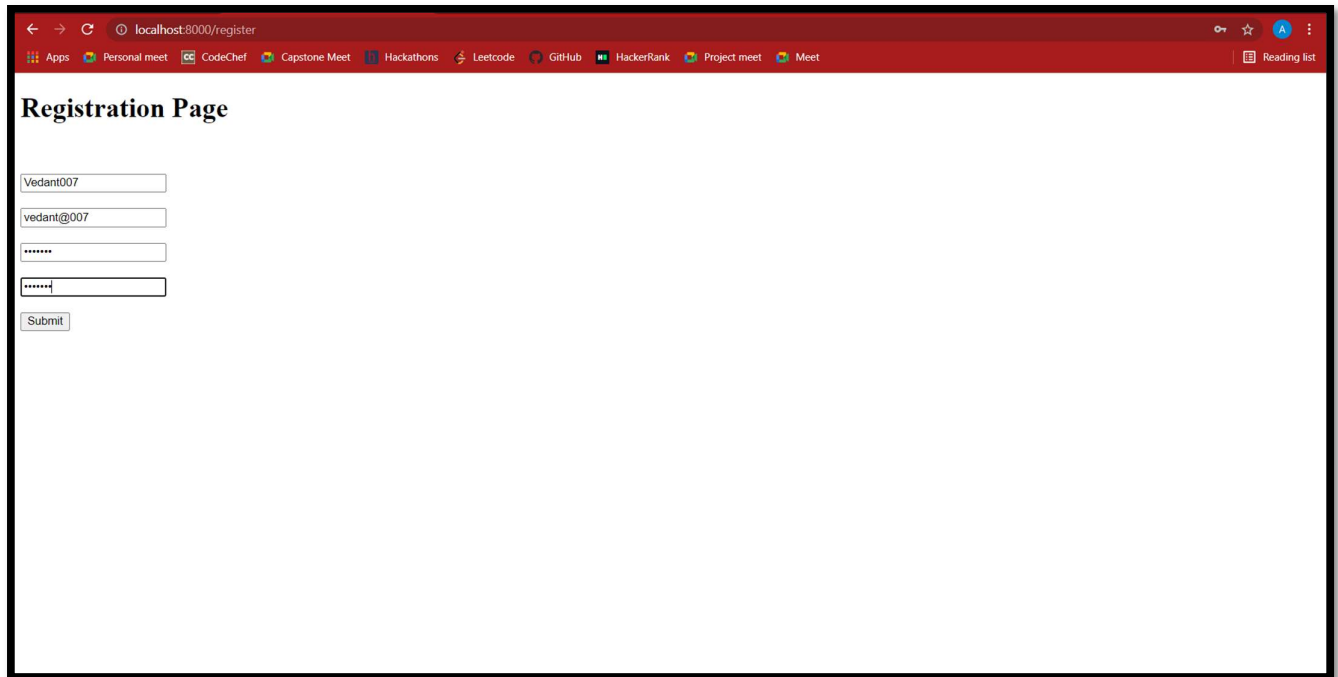
## Output Snippets:

### Index Page:





## Registration Page:



localhost:8000/register

Apps Personal meet CodeChef Capstone Meet Hackathons Leetcode GitHub HackerRank Project meet Meet Reading list

### Registration Page

Vedant007

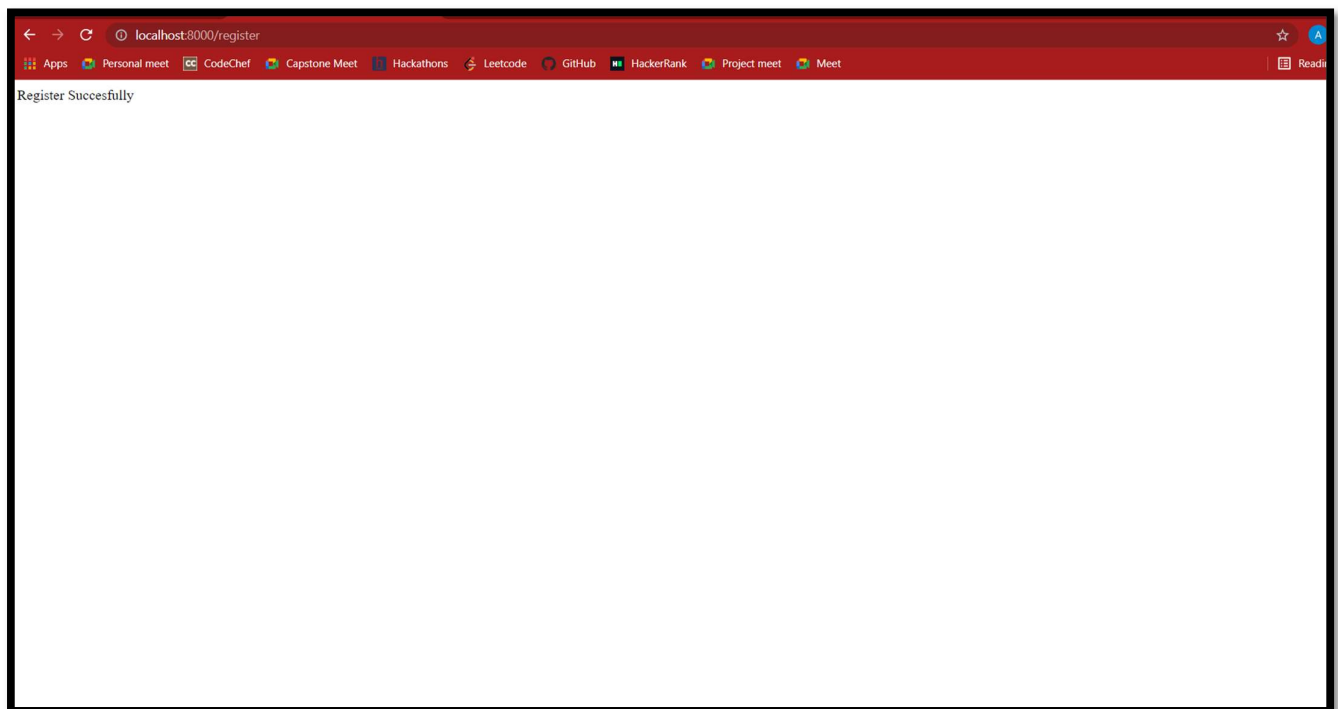
vedant@007

\*\*\*\*\*

\*\*\*\*\*

Submit

## Registration Successful:

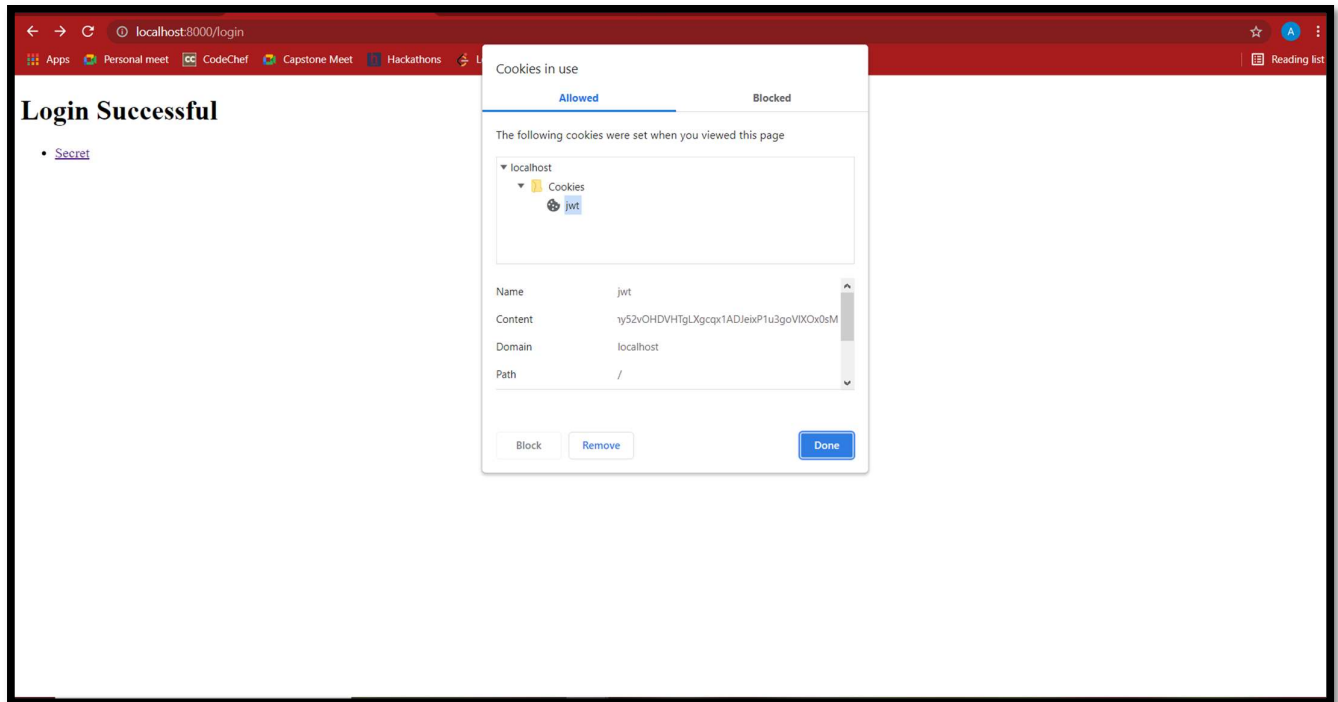


localhost:8000/register

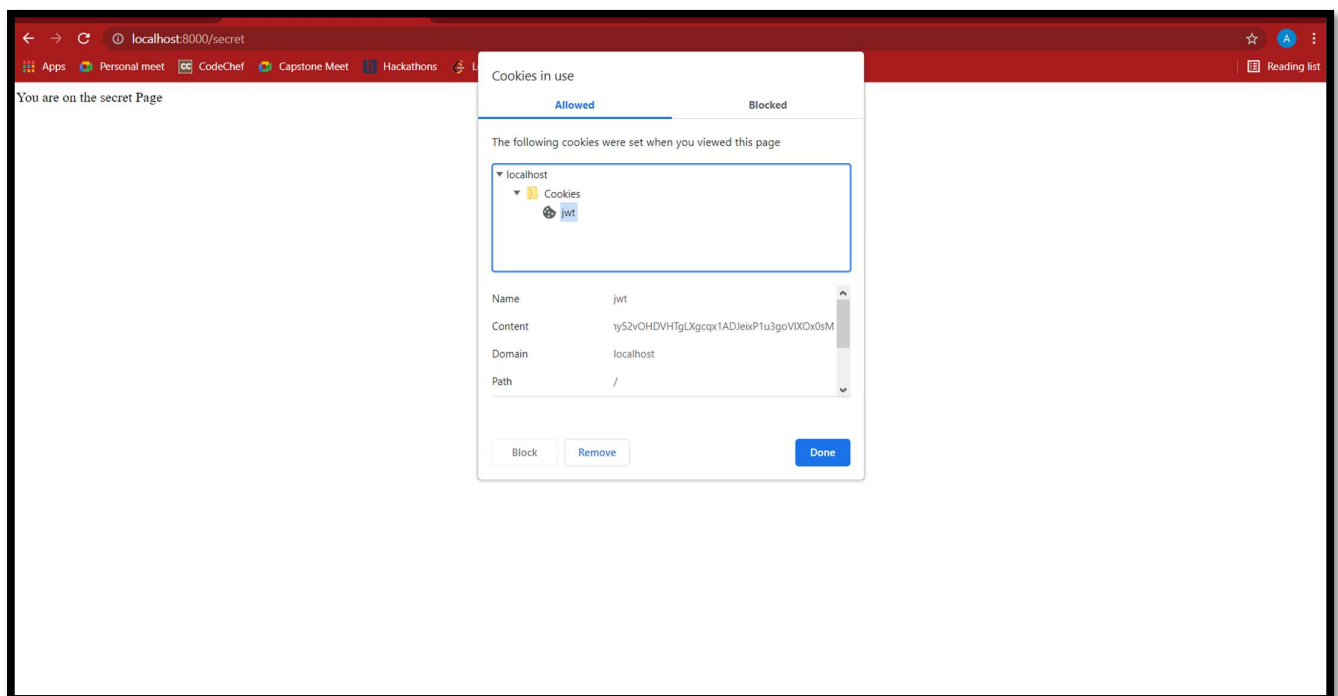
Apps Personal meet CodeChef Capstone Meet Hackathons Leetcode GitHub HackerRank Project meet Meet Reading list

Register Successfully

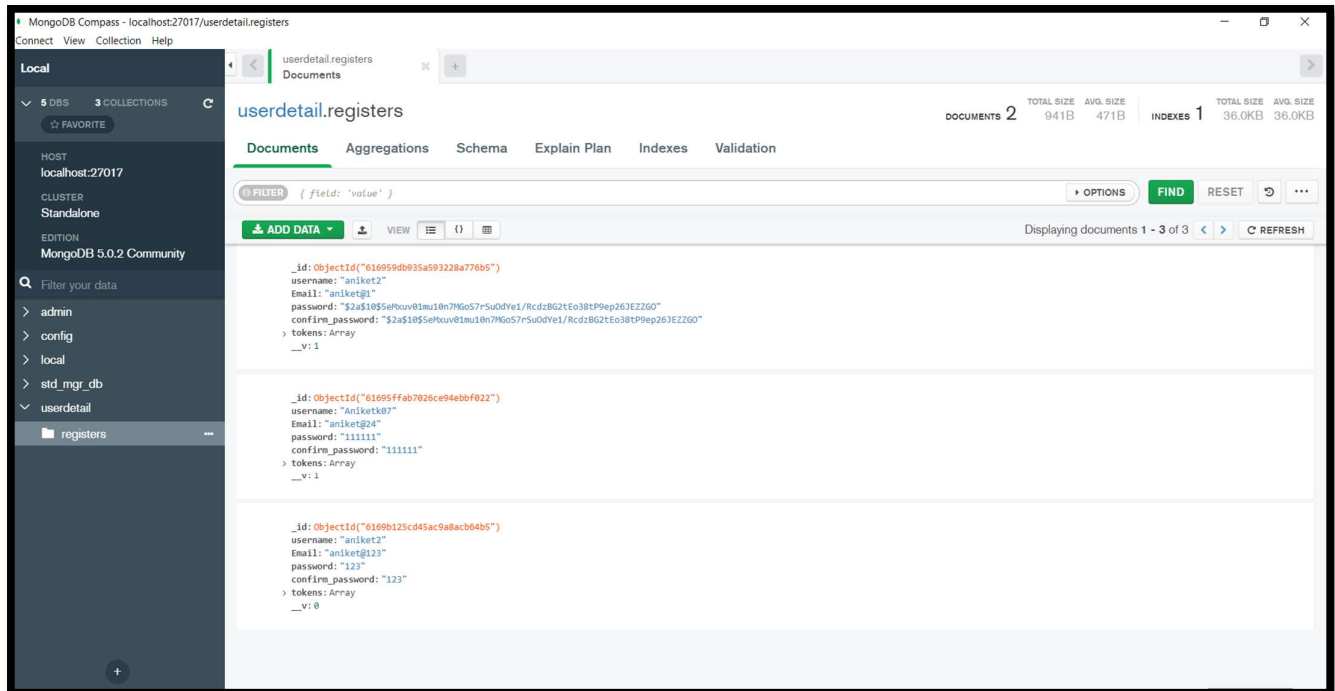
## Login Successful & JWT:



## Login to internal Page:



# MongoDB Page:



-----END OF TASK-----