PROJECT TITLE:

LOGIN AUTHENTICATION: AUTHENTICATE USERS WITH A USERNAME AND PASSWORD TO ACCESS THE APPLICATION'S FEATURES.

Table of Contents

Acknowledgement

We extend our deepest gratitude to Dr. Jabhera Matogoro, whose unwavering dedication, invaluable guidance, and insightful mentorship have been instrumental throughout every phase of the development of the Login Authentication system. With his encouragement, and constructive feedback have not only improved our understanding but have also inspired us to push the boundaries of our capabilities.

Furthermore, we wish to express our sincere appreciation to our dedicated colleagues whose unwavering support, collaborative spirit, and collective efforts have significantly contributed to the success of this project. Their diverse perspectives, expertise in React JS, Spring Boot, and MySQL technologies, and willingness to collaborate have fostered an environment conducive to innovation and growth.

Additionally, we would like to extend our gratitude to all individuals who have directly or indirectly contributed to this project, including friends, family, and mentors. Their encouragement, patience, and unwavering belief in our abilities have been a constant source of motivation throughout this journey.

Once again, thank you, Dr. Matogoro, for your exceptional guidance and mentor ship. We are grateful for the opportunity to work with you and for the knowledge and experience gained throughout this project.

INTRODUCTION

Project Overview

The project aims to address a critical challenge faced by many student management systems: the security vulnerabilities inherent in their login mechanisms. At the frontend of this project is the development of a robust Login Authentication System designed to fortify the authentication process, particularly in educational institutions. Leveraging advanced technologies such as React JS for the frontend, Spring Boot for the backend, and MySQL for the database, this system represents a significant step forward in enhancing both the efficiency and security of student admissions processes. By automating routine tasks and implementing stringent security measures, the system not only streamlines operations but also safeguards sensitive data against unauthorized access and potential breaches.

The project not only focuses on enhancing the login mechanism's security but also emphasizes the integration of cryptography to fortify user authentication. By employing hashing algorithms to encrypt passwords stored in the database, the system ensures that sensitive user credentials remain protected even in the event of a breach. Additionally, the project implements a token-based authentication system, wherein users receive unique tokens via email upon registration. These tokens serve as validation mechanisms, verifying the authenticity of user accounts and mitigating the risk of unauthorized access. Furthermore, each user is assigned a randomly generated number at the database level, adding an extra layer of security to the system. This approach enhances user privacy and resilience against common security threats, aligning with the project goals that bolstering security in student management systems while leveraging cutting-edge cryptographic techniques.

In this report, we delve into the key functionalities such as registering user, login mechanism, add users, see list of user and room allocation.

Background

The imperative for robust security measures in student management systems arises from the pervasive vulnerabilities that have plagued traditional systems, particularly concerning login mechanisms. Historically, many institutions have grappled with the repercussions of weak authentication protocols, leaving their systems vulnerable to a myriad of cyber threats. Instances of unauthorized access, data breaches, and compromised user credentials have underscored the critical need for a paradigm shift towards fortified security frameworks.

In response to these challenges, the project delves into the intricate landscape of cybersecurity, recognizing the indispensable role of cryptography in safeguarding sensitive data. Traditional systems, often reliant on rudimentary authentication methods, have proven insufficient in the face of increasingly sophisticated cyber threats. By harnessing cryptographic techniques such as hashing algorithms, the project aim to improve every foundation of user authentication, ensuring that passwords are securely encrypted before storage, thereby mitigating the risk of data compromise.

Furthermore, the project acknowledges the vulnerability inherent in traditional email-based authentication methods. Recognizing the susceptibility of email communication to interception and spoofing, the project seeks to augment traditional email verification with token-based authentication. This approach, wherein unique tokens are generated and sent to users via email, serves as a robust validation mechanism, bolstering the security of user accounts and thwarting potential unauthorized access attempts.

Moreover, the project adopts a proactive stance towards user privacy, recognizing the importance of implementing additional security measures beyond traditional authentication protocols. By assigning each user a randomly generated number at the database level, the project further fortifies the security posture of the system, mitigating the risk of data correlation and enhancing user privacy.

Problem Statement

The prevailing issues plaguing conventional login mechanisms within student management systems underscore a critical need for comprehensive solutions to address inherent security vulnerabilities. Weak authentication mechanisms, characterized by simplistic username-password combinations, pose a significant risk to the integrity and confidentiality of user credentials. These vulnerabilities leave systems susceptible to a spectrum of cyber threats, including brute force attacks, dictionary attacks, and credential stuffing, which can result in unauthorized access and potential data breaches. The architectural vulnerabilities inherent in many student management systems exacerbate these security concerns. Inadequate implementation of authentication protocols and flawed system design expose vulnerabilities such as SQL injection, cross-site scripting (XSS), and session hijacking, further compounding the risk of unauthorized access and data compromise.

Compounding these challenges is the stringent regulatory landscape governing the protection of student data, the proliferation of cyber threats necessitates a proactive stance towards user privacy and data security. By assigning each user a randomly generated number at the database

level, the project aims to augment traditional authentication mechanisms, enhancing user privacy and resilience against common security threats.

General Objective

The overall goal of this project is to develop a robust and secure Login Authentication system tailored specifically for student management systems. This system aims to fortify the authentication process, ensuring the integrity and confidentiality of user credentials while mitigating the risk of unauthorized access and data breaches. By adopting advanced cryptographic techniques and adhering to best practices in cybersecurity, the project seeks to establish a resilient security framework that enhances the overall security posture of student management systems.

Specific Objectives

1. Implement a Secure Authentication Mechanism: The project aims to implement a robust authentication mechanism that utilizes advanced cryptographic techniques to securely verify the identity of users. By leveraging hashing algorithms and salting techniques, passwords will be securely encrypted before storage, mitigating the risk of data compromise in the event of a breach.
2. Develop Backend Functionalities for Secure Authentication: The project will focus on developing backend functionalities using Spring Boot to facilitate secure authentication processes. This involves implementing stringent access controls, session management mechanisms, and encryption protocols to fortify the authentication process and prevent unauthorized access to sensitive user data.
3. Integrate MySQL Database for Secure Storage: The project will integrate a MySQL database to securely store user credentials and authentication tokens. By implementing encryption-at-rest and access control measures, the system will ensure the confidentiality and integrity of user data, safeguarding against potential data breaches and unauthorized access attempts.
4. Define User Roles and Access Permissions: The project will define distinct user roles, including administrators, staff, and students, each with specific access permissions and functionalities. Role-based access control mechanisms will be implemented to restrict access to sensitive features and data, ensuring that users only have access to the information and functionalities relevant to their role.
5. Enable User Registration, Login, and Account Activation: The project will enable user registration, login, and account activation functionalities, ensuring a seamless and secure user authentication process. Users will receive unique tokens via email upon registration, serving as validation mechanisms to activate their accounts securely.
6. Implement Role-Based Access Control: Role-based access control mechanisms will be implemented to restrict access to application features based on user roles. This involves defining access control policies, enforcing access restrictions, and auditing user activities to ensure compliance with security policies and regulatory requirements.

Structure of the document

The structure of this document is meticulously crafted to offer a comprehensive understanding of the Login Authentication system. Beginning with an introductory overview, it delves into the historical context and underlying challenges, articulating the project's overarching goal and specific objectives. An expanded section details each subsequent section's content and purpose, serving as a navigational guide. This is followed by explorations of related work, relevance, methodology, and technologies utilized, culminating in a detailed exposition of the system's implementation and operational mechanics. The document concludes with a concise summary encapsulating key insights and references for further exploration, providing readers with a holistic view of the project's scope, significance, and contributions to enhancing security in student management systems.

Related Work

This section of the report presents a comprehensive analysis of existing research and developments in the realm of secure authentication systems. It shows effort to contextualize the project within the broader landscape of cybersecurity and authentication mechanisms, drawing insights from seminal works and cutting-edge advancements in the field.

In this section, we delve into various frameworks, technologies, and protocols that have been employed to fortify user authentication processes. Among these are industry-standard frameworks such as OAuth, JWT, and OAuth 2.0, which have emerged as pillars of secure authentication and authorization. Through a detailed examination of these frameworks, we aim to show their relevance and applicability to our project, offering insights into their underlying principles, methodologies, and best practices.

Moreover, the section explores recent advancements in cryptographic techniques and authentication mechanisms, including multi-factor authentication (MFA), biometric authentication, and blockchain-based authentication systems.

Methodology and Sustainability

The development of the Login Authentication system follows a systematic approach, incorporating best practices in software development and cybersecurity. The use of React JS for the front end, Spring Boot for the backend, and MySQL for the database ensures a robust and scalable architecture that can accommodate future enhancements and updates. Additionally, the implementation of role-based access control and secure authentication mechanisms contributes to the long-term sustainability and security of the system.

Technologies Used

1. Front End: React JS

2. Back End: Spring Boot

3. Database: MySQL

4. Authentication Mechanism: JWT (JSON Web Tokens)

IMPLEMENTATION

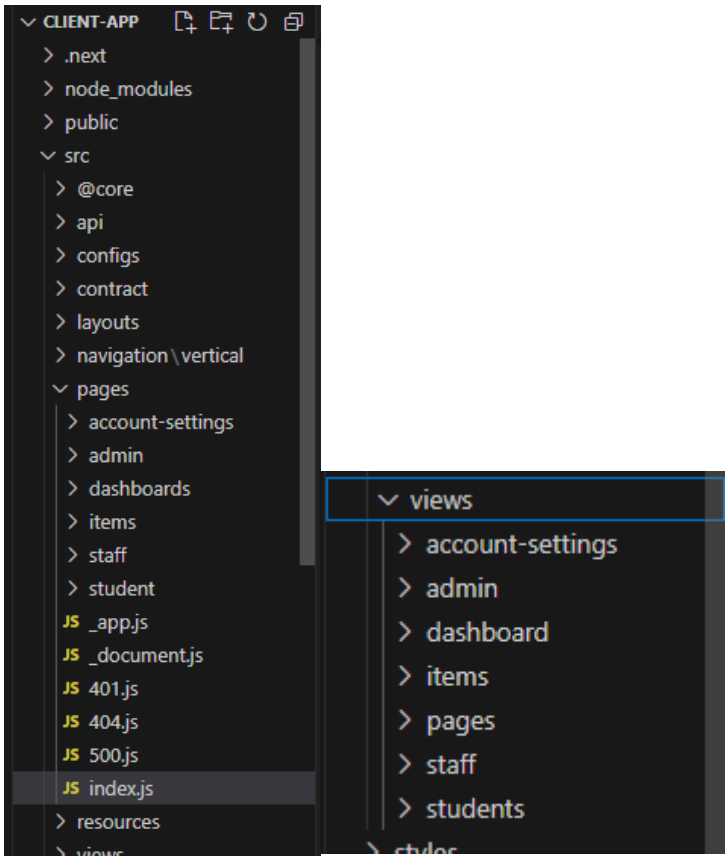
The implementation of the Login Authentication system involves the development of front end and back end components using React JS and Spring Boot, respectively. The system architecture comprises client-side and server-side components that interact seamlessly to authenticate users securely and efficiently.

The Implementation of program is based Two part:

- ✓ Font End (Client Side)
- ✓ Back End (Server Side)

FRONT END

The front end of the system is developed using React JS, a popular JavaScript library for building user interfaces. The front end components include login, registration, account activation, and user dashboard functionalities. React JS provides a robust and responsive user interface, ensuring an intuitive user experience for accessing application features securely.



System structure

Login Page

```
JS index.js ×
src > pages > JS index.js > LoginPage > handleSubmit
61  const LoginPage = () => {
62
63    // ** Hook
64    const theme = useTheme()
65    const router = useRouter()
66    // const authStore = useAuthStore();
67
68    // ** State
69    const [openAlert, setOpenAlert] = useState(false)
70    const [errormsg, setErrorg] = useState(null)
71    const [values, setValues] = useState({
72      email: '',
73      password: '',
74      showPassword: false
75    })
76
77
78    const handleChange = prop => event => {
79      setValues({ ...values, [prop]: event.target.value })
80    }
81
```

```
Run ···  ← →  client-app
JS index.js ×
src > pages > JS index.js > LoginPage > handleSubmit
61  const LoginPage = () => {
62
63    // ** Hook
64    const theme = useTheme()
65    const router = useRouter()
66    // const authStore = useAuthStore();
67
68    // ** State
69    const [openAlert, setOpenAlert] = useState(false)
70    const [errormsg, setErrorg] = useState(null)
71    const [values, setValues] = useState({
72      email: '',
73      password: '',
74      showPassword: false
75    })
76
77
78    const handleChange = prop => event => {
79      setValues({ ...values, [prop]: event.target.value })
80    }
81
82    const handleClickShowPassword = () => {
83      setValues({ ...values, showPassword: !values.showPassword })
84    }
85
86    const handleMouseDownPassword = event => {
87      event.preventDefault()
88    }
89
90    function extractUserRoleFromToken(token) {
91      const decodedToken = JSON.parse(atob(token.split(".")[1]));
Ln 136, Col 11 (326 selected)  Spaces: 2  UTF-8  LF  {
```

Run ··· client-app

index.js

src > pages > index.js > LoginPage > handleSubmit

```
61  const LoginPage = () => {
89
90    function extractUserRoleFromToken(token) {
91      const decodedToken = JSON.parse(atob(token.split(".")[1]));
92      return decodedToken.role;
93    }
94
95    const handleSubmit = async (e) => {
96      e.preventDefault();
97      setErrorg(null);
98      if (!values.email || !values.password) {
99        setErrorg("Please enter both email and password.");
100       setOpenAlert(true);
101       return;
102     }
103     try {
104
105       // send the login request to the server
106         const response = await axiosInstance.post(
107           "auth/authenticate", // the endpoint
108           values, // the request body
109           { withCredentials: true },
110         );
111
112          // get the token from the response
113           const accessToken = response.data.access_token;
114           const refreshToken = response.data.refresh_token;
115
116           // set the token in local storage
117           localStorage.setItem("access_token", accessToken);
118           localStorage.setItem("refresh_token", refreshToken);
119
```

Ln 136, Col 11 (326 selected) Spaces: 2 UTF-8 LF {} JavaSc

Run ··· client-app

index.js

src > pages > index.js > LoginPage > handleSubmit

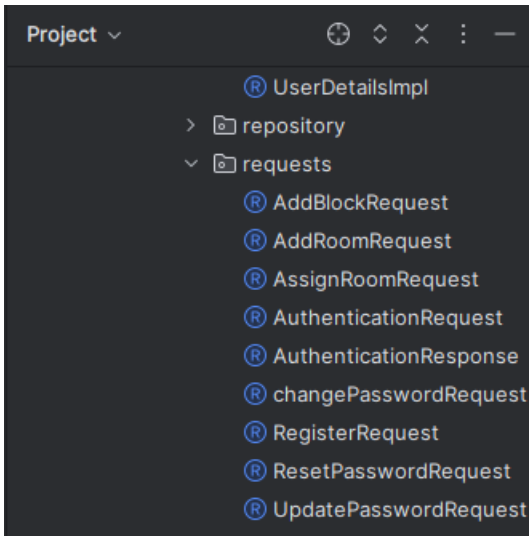
```
61  const LoginPage = () => {
95    const handleSubmit = async (e) => {
119
120           // update the authorization header
121           axiosInstance.defaults.headers.common[
122             "Authorization"
123           ] = `Bearer ${accessToken}`;
124
125           // extract the user role from the token
126           const userRole = extractUserRoleFromToken(accessToken);
127           localStorage.setItem("role", userRole);
128
129 console.log(userRole);
130           // call the stores login method this will update the stores state
131           // authStore.login(userRole);
132           setOpenAlert(true);
133           setErrorg("Login success");
134
135           // redirect to the home page
136           if(userRole == "ROLE_USER"){
137             await router.push("/dashboards/student_dashboard");
138           } else if (userRole == "ROLE_ADMIN"){
139             await router.push("/dashboards/admin_dashboard");
140           } else if(userRole == "ROLE_STAFF"){
141             await router.push("/dashboards/employee_dashboard");
142           }
143
144
145         // const token = await AuthService.login(values);
146         // localStorage.setItem('token', token);
147       } catch (error) {
148         console.error('Authentication failed:', error);
```

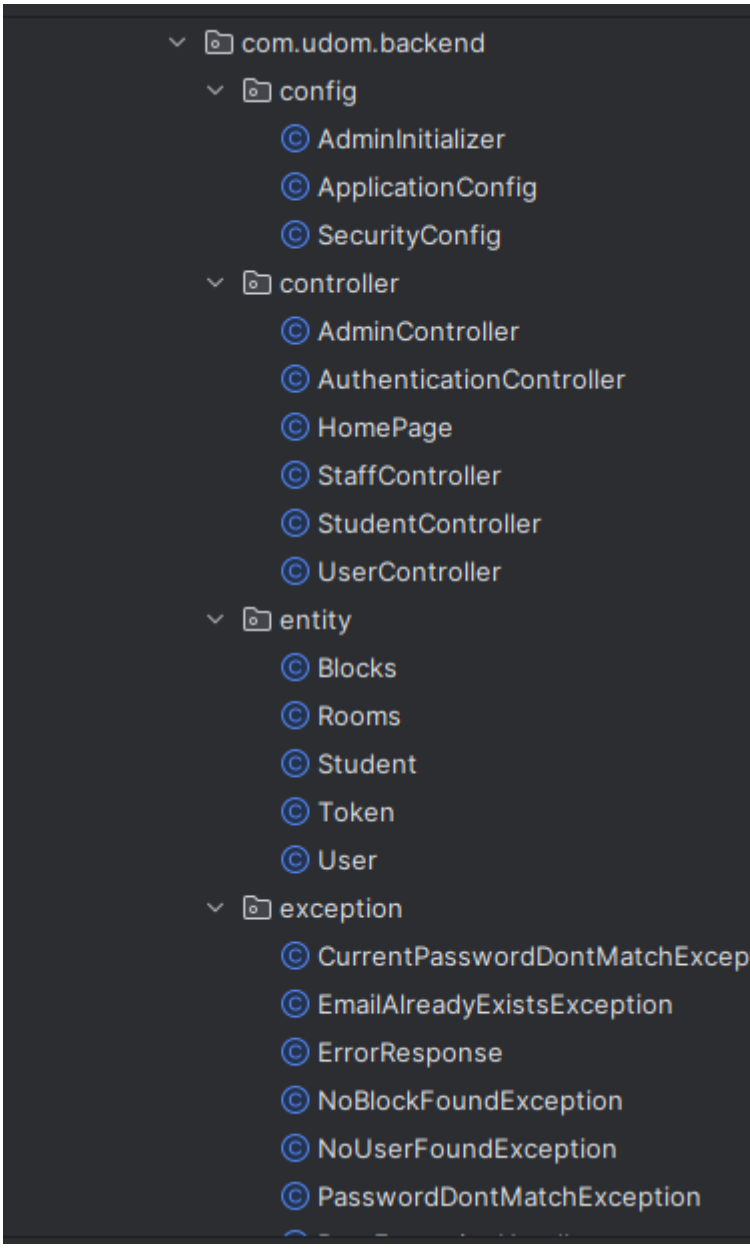
```
Run ...  client-app
JS index.js ×
src > pages > JS index.js > LoginPage > handleSubmit
61  const LoginPage = () => {
246             </Typography>
247             <Typography variant='body2'>Please sign-in to your account and start the adventure</Typography>
248           </Box>
249           <form noValidate autoComplete='off' onSubmit={e => e.preventDefault()}>
250             <TextField autoFocus fullWidth id='email' label='email' sx={{ marginBottom: 4 }} value={values.email} onCha
251             <FormControl fullWidth>
252               <InputLabel htmlFor='auth-login-password'>Password</InputLabel>
253               <OutlinedInput
254                 label='Password'
255                 value={values.password}
256                 id='auth-login-password'
257                 onChange={handleChange('password')}
258                 type={values.showPassword ? 'text' : 'password'}
259                 endAdornment={
260                   <InputAdornment position='end'>
261                     <IconButton
262                       edge='end'
263                       onClick={handleClickShowPassword}
264                       onMouseDown={handleMouseDownPassword}
265                       aria-label='toggle password visibility'
266                     >
267                       {values.showPassword ? <EyeOutline /> : <EyeOffOutline />}
268                     </IconButton>
269                   </InputAdornment>
270                 }
271               />
272             </FormControl>
273             <Box sx={{ mt: 3}} >
274             {openAlert ? (
275               <Grid item xs={12} sx={{ mb: 3 }}>
276                 <Alert
```

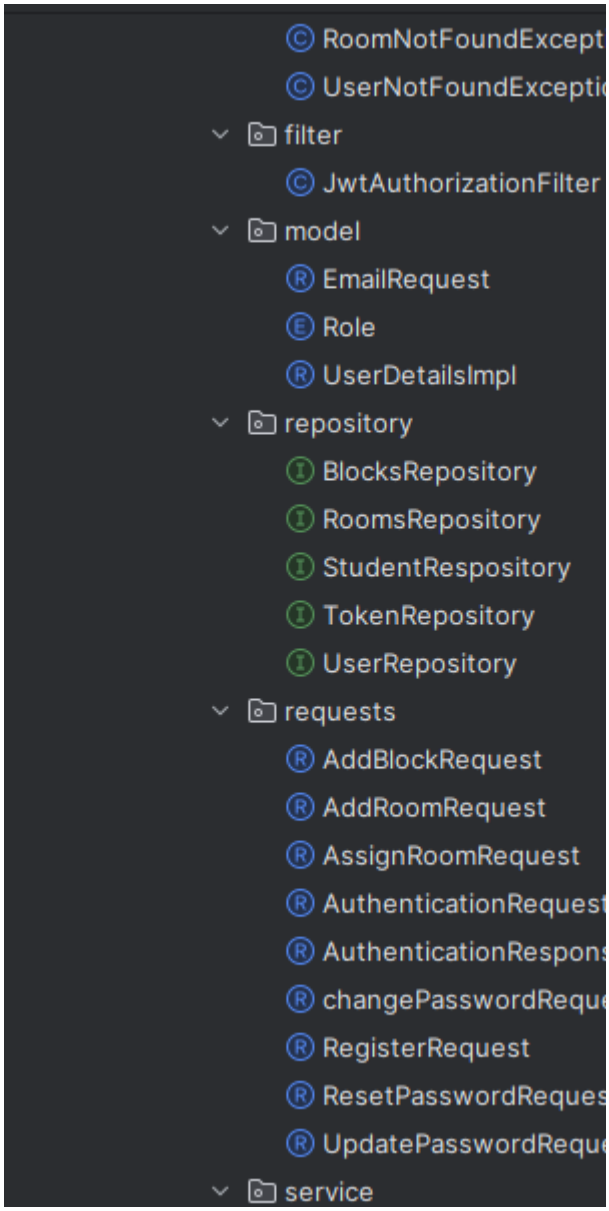
BACK END

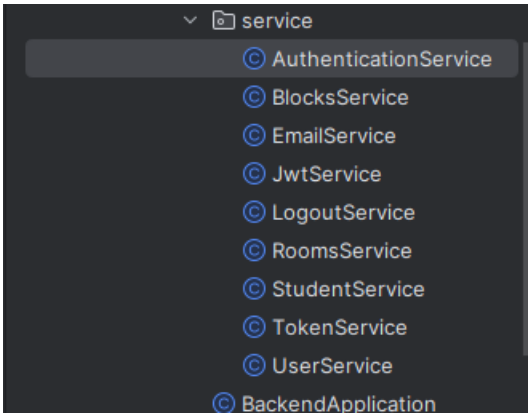
The back end of the system is built using Spring Boot, a lightweight framework for building Java-based applications. The back-end components include authentication, authorization, user management, and database interaction functionalities. Spring Boot provides a scalable and efficient framework for implementing secure authentication mechanisms and managing user data securely.

Here are some pages that are used in back which shows some mechanism of security include:






RoomNotFoundExcept
UserNotFoundExcepti
filter
JwtAuthorizationFilter
model
EmailRequest
Role
UserDetailsImpl
repository
BlocksRepository
RoomsRepository
StudentRespository
TokenRepository
UserRepository
requests
AddBlockRequest
AddRoomRequest
AssignRoomRequest
AuthenticationRequest
AuthenticationRespons
changePasswordReque
RegisterRequest
ResetPasswordReques
UpdatePasswordReque
service


service
AuthenticationService
BlocksService
EmailService
JwtService
LogoutService
RoomsService
StudentService
TokenService
UserService
BackendApplication

Authentication Page: This page used to provide authentication when user register into the system by provide token that sent to email account also it used to hash password when stored into the data base

```java
.java   StaffController.java   UserController.java   BackendApplication.java   AuthenticationService.java
3 usages
public AuthenticationResponse authenticate(AuthenticationRequest request) {
    try {
        // Attempts to authenticate the user with the provided email and password
        authenticationManager.authenticate(
          new UsernamePasswordAuthenticationToken(
            request.email(),
            request.password()
          )
        );
    } catch (InternalAuthenticationServiceException e) {
        // If the authentication fails, throws an exception with a message indicating invalid credentials
        log.error("error while authenticating user with request {}", request);
        throw new BadCredentialsException("Invalid credentials");
    }

    // If the authentication is successful, retrieves the user from the database and generates a JWT toke
    User user = userService.validateCredentials(request.email(), request.password());
    log.info("User {} successfully authenticated with role {}", user.getFirstName(), user.getRole());

    String accessToken = jwtService.generateAccessToken(user);
    String refreshToken = jwtService.generateRefreshToken(user.getEmail());

    tokenService.revokeAllUserTokens(user);
    tokenService.saveUserToken(user, accessToken);

    // Returns an authentication response containing the JWT token
    return new AuthenticationResponse(accessToken, refreshToken);
```

Security Config: This is used to configure the setting of the system that will be used in many of operation in the system such as get and post

```java
AuthenticationService.java   JwtService.java   SecurityConfig.java   ApplicationConfig.java
@RequiredArgsConstructor
public class SecurityConfig {
    private final JwtAuthorizationFilter jwtAuthFilter; // this is the filter that will be used
    private final AuthenticationProvider authenticationProvider; // this is the provider that wi
    private final LogoutHandler logoutHandler; // this is the handler that will be used to log o

    no usages
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {...}

    no usages
    @Bean(name = "corsConfigurationSource")
    public CorsConfigurationSource corsConfiguration() {

        final CorsConfiguration corsConfiguration = new CorsConfiguration();


        corsConfiguration.setAllowedOriginPatterns(List.of( e1: "*"));

        corsConfiguration.setAllowedMethods(
                List.of(
                        "GET",
                        "POST",
                        "PUT",
                        "DELETE"
                )
        );
```

JWTAuthrization Filter: This page used to Check the validity of token generated to the user and allowed public endpoint and private that are defined

```
AuthenticationService.java   JwtService.java   SecurityConfig.java   JwtAuthorizationFilter.java

public class JwtAuthorizationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserService userService;
    private final TokenService tokenService;

    // The Authorization header is in the form of Bearer <token>. The prefix Bearer is removed t
    1 usage
    private static final String AUTHORIZATION_HEADER = "Authorization";

    // The prefix Bearer is removed to extract the token.
    1 usage
    private static final String BEARER_PREFIX = "Bearer ";

    // The list of public endpoints that do not require authentication
    1 usage
    private static final List<String> PUBLIC_ENDPOINTS = List.of(
        "/api/v1/auth/register",
        "/api/v1/auth/refresh-token",
        "/api/v1/auth/enable-user",
        "/api/v1/auth/authenticate",
        "/api/v1/auth/forgot-password",
        "/api/v1/auth/reset-password",
            "/api/v1/auth/change-password"
    );
```

```
uthenticationService.java   JwtService.java   SecurityConfig.java   JwtAuthorizationFilter.java

    @Override
    protected void doFilterInternal(
        @NonNull HttpServletRequest request,
        @NonNull HttpServletResponse response,
        @NonNull FilterChain filterChain
    ) throws ServletException, IOException {
        // Check if the request is for a public endpoint
        if (isPublicEndpoint(request)) {
            // If the request is for a public endpoint, skip the filter
            log.info("Skipping the filter for the following request URL {}", request.getServletP
            filterChain.doFilter(request, response);
            return;
        }

        // Check if the Authorization header is missing or does not contain a valid JWT
        String authHeader = request.getHeader(AUTHORIZATION_HEADER);
        if (authHeader == null || !authHeader.startsWith(BEARER_PREFIX)) {
            // Handle the case where the Authorization header is missing or does not contain a v
            handleMissingToken(response, request);
            return;
        }

        // Extract JWT from the Authorization header
        String jwt = authHeader.substring( beginIndex: 7);
        // Extract username from the JWT
        String username = extractUsernameFromJwt(jwt);

        // If username is null, it indicates an issue with the JWT
```

CONNECTION BETWEEN FRONT END AND BACK END

The front end and back-end components communicate via RESTful APIs, enabling seamless interaction between client-side and server-side components. The connection between the front end and back end is established using Axios, a promise-based HTTP client for JavaScript. Axios facilitates data exchange between the front end and back end, ensuring secure communication and efficient data transfer.

```
EXPLORER
CLIENT-APP
  .next
  node_modules
  public
  src
    @core
    api
      axiosInstance.js
      interceptor.js
    configs
    contract
    layouts
    navigation
    pages
      account-settings
      admin
      dashboards
      items
      staff
      student
      _app.js
      _document.js
      401.js
      404.js
```

axiosInstance.js

src > api > axiosInstance.js > axiosInstance

```
import axios from "axios";
import React, { Component } from 'react'


const API_URl = "http://localhost:3001";
// set the base URL from the .env file
const baseURL = API_URl + "/api/v1";

// create an axios instance
const axiosInstance = axios.create({
  baseURL,
});

// set the Authorization header for every request
let token;
if (typeof localStorage !== 'undefined') {
  token = localStorage.getItem("access_token");

  if (token != null) {
    axiosInstance.defaults.headers.common["Authorization"] = `Bearer ${token}`;
  }
}



export default axiosInstance;
```

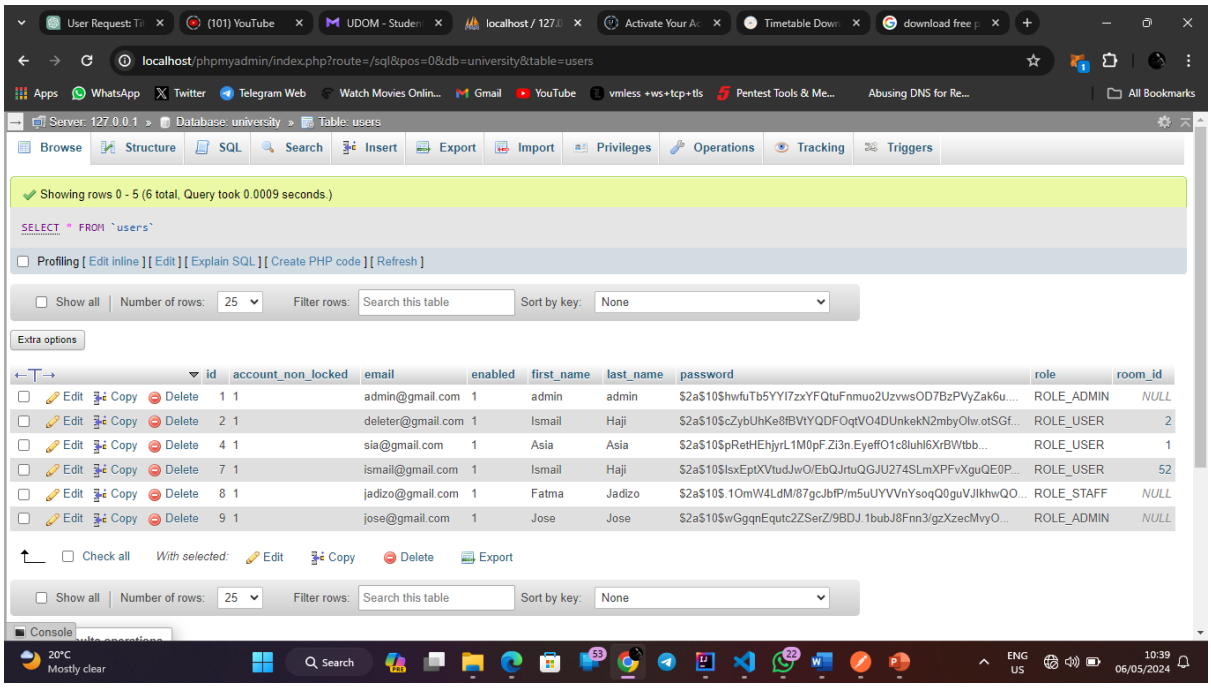
The connection to the MySQL database

JwtService.java | TokenService.java | application.yml | SecurityConfig.j

```yaml
server:
  port: 3001

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/university
    username: root
    password:
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    show-sql: true
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL8Dialect
        format_sql: true
    database: mysql
#     database-platform: PostgreSQLDialect
```

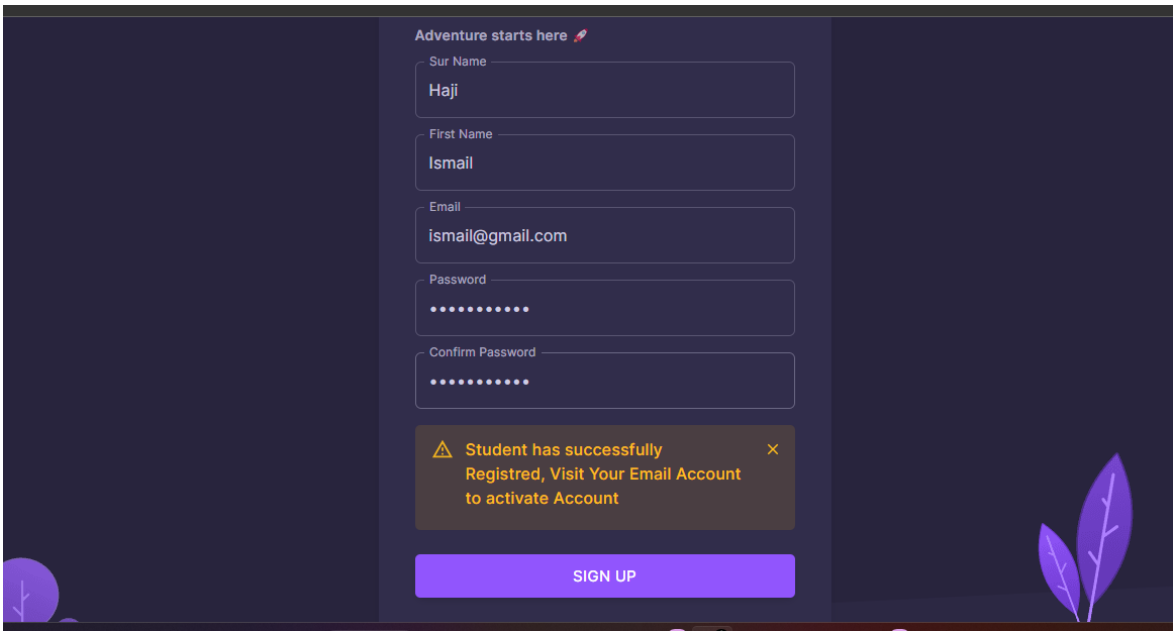
```yaml
jwt:
  # different expiration time for each token
  expiration:
    access-token: 86400000          # 1 hour
    refresh-token: 604800000       # 7 days
    reset-password: 900000         # 15 minutes
    enable-account: 900000

  # The jwt secret key is used to sign the token.
  secret-key: 404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970
```

User as seen in database with password encrypted

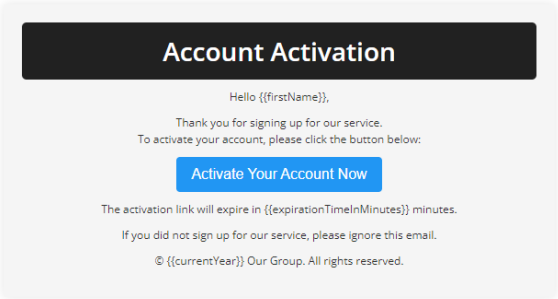


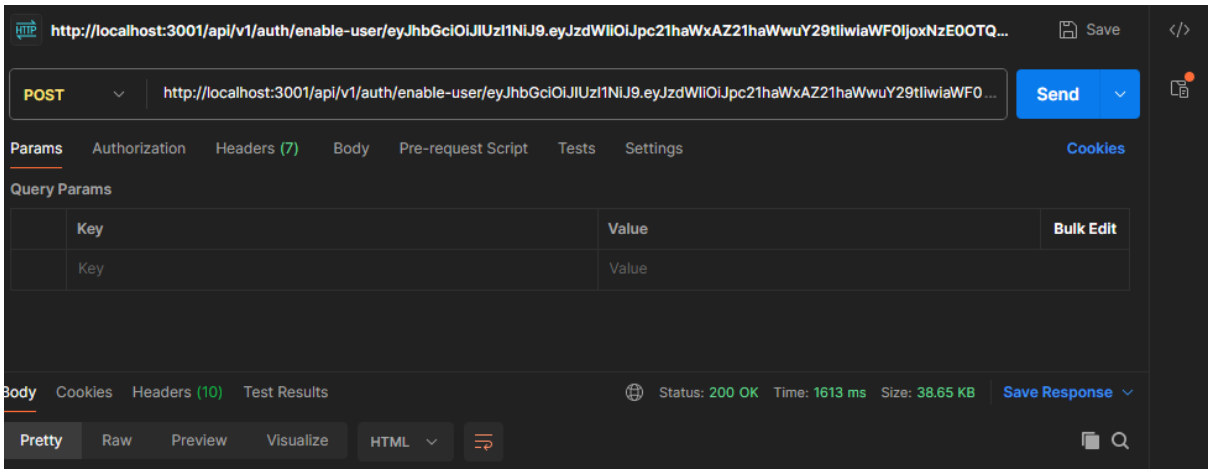
HOW DOES THE SYSTEM WORK

1. **User Registration:** Users can register an account using their email address and password. Upon registration, users receive an email verification link to activate their account.



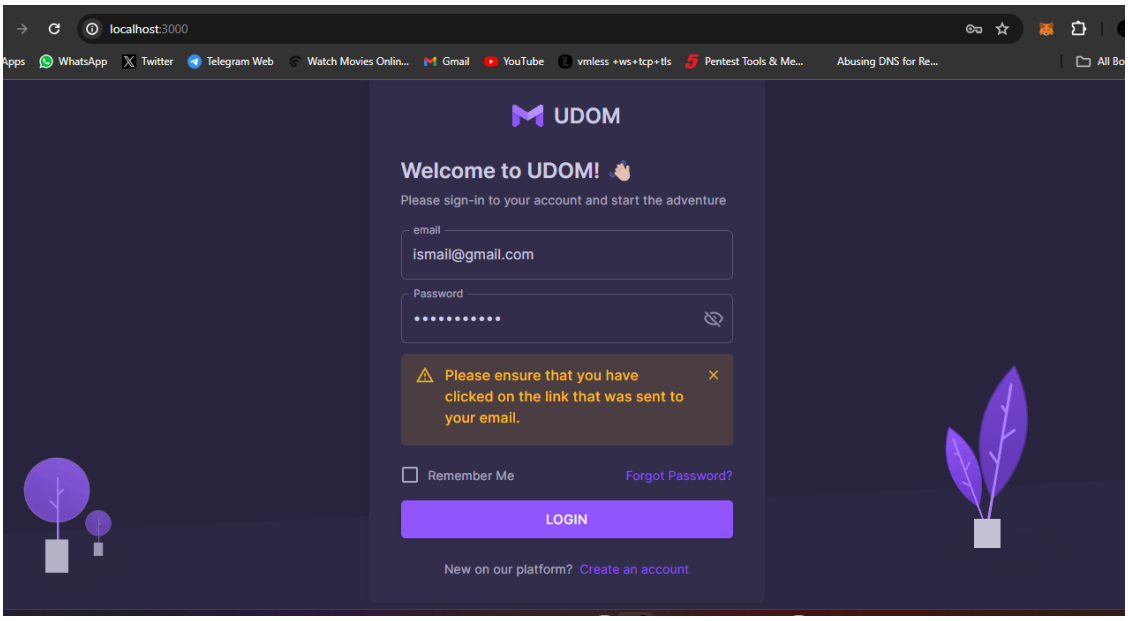
2. **Account Activation:** Users activate their account by clicking the email verification link sent to their email address. Once activated, users can log in to access the application's features.



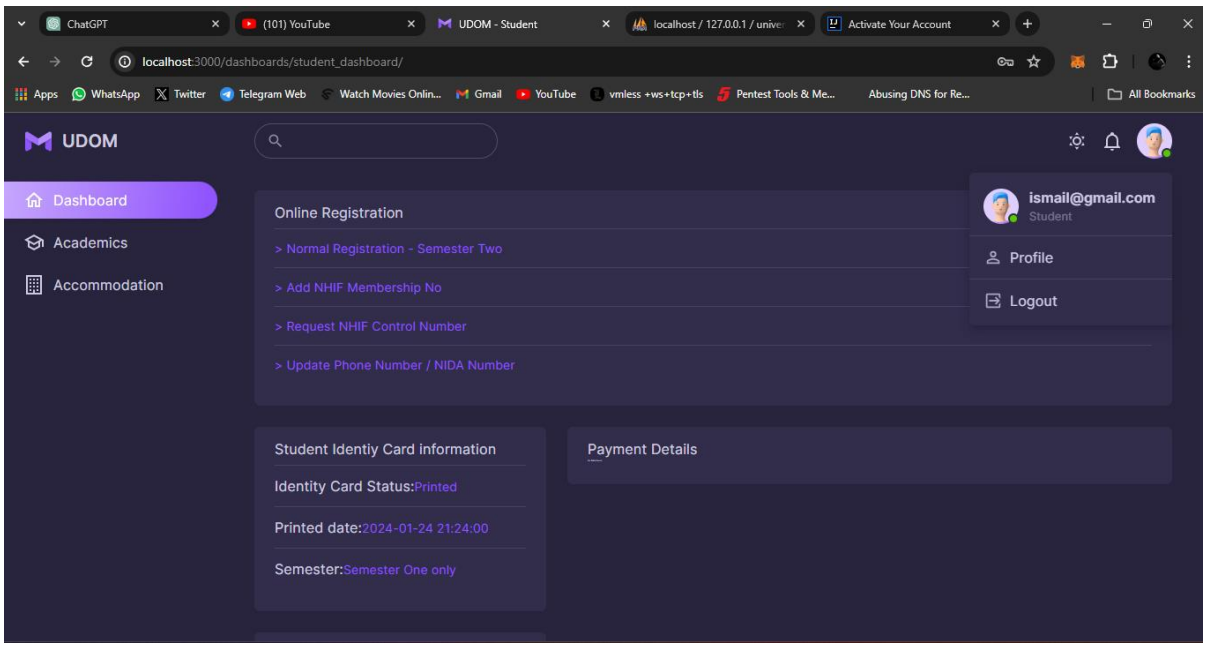


3**. Login:** User can login in the system after activate account if not he won't able to login

Role-based Access Control: The system defines three user roles: admin, staff, and user, each with specific access permissions and functionalities. Role-based access control ensures that users have appropriate access permissions based on their roles within the application.

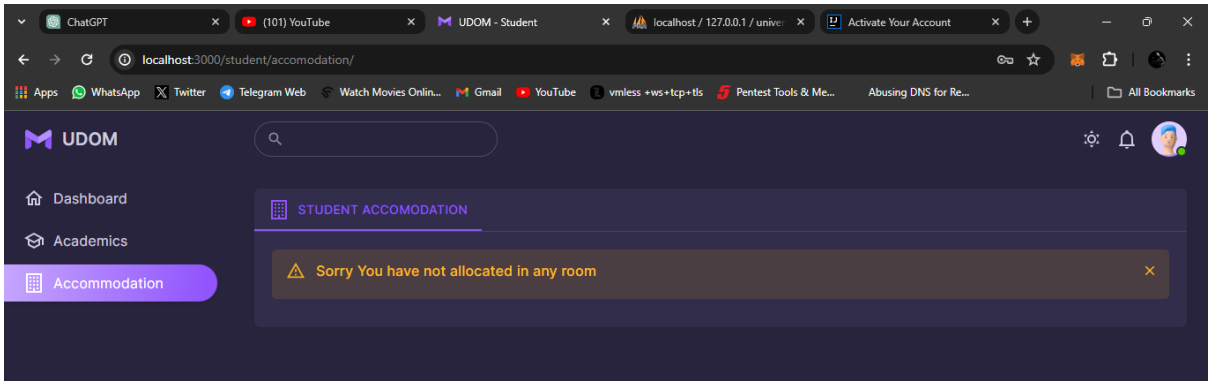


4. **User Dashboard**: Upon successful login, users are redirected to their respective dashboards based on their roles. The dashboard provides access to application features and functionalities based on the user's role.



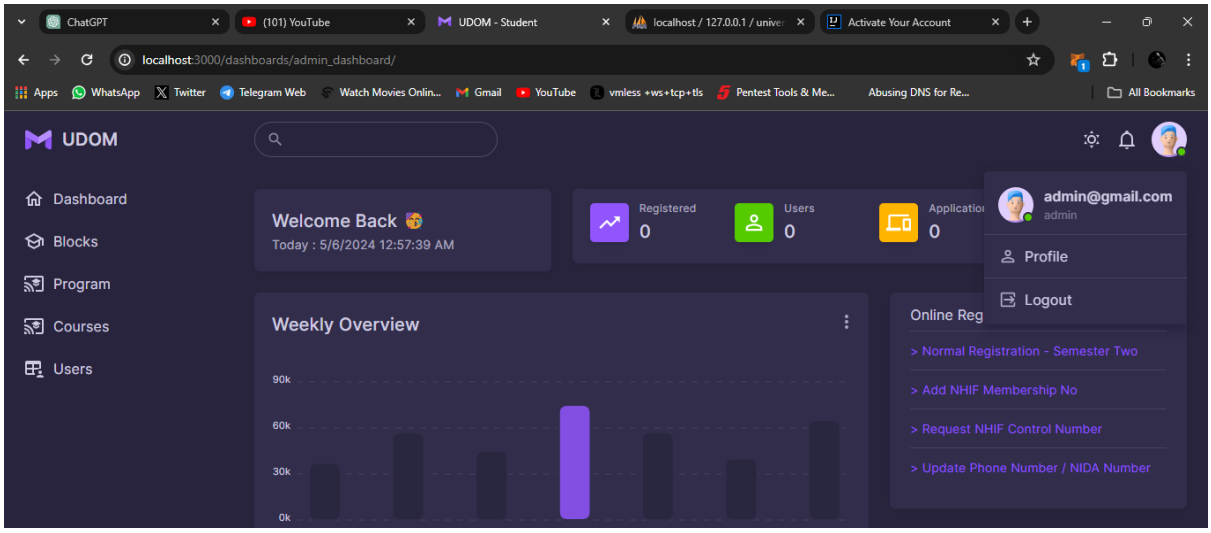
NORMAL USER (STUDENT)

This is normal user when into to account he will be able to see his room allocated

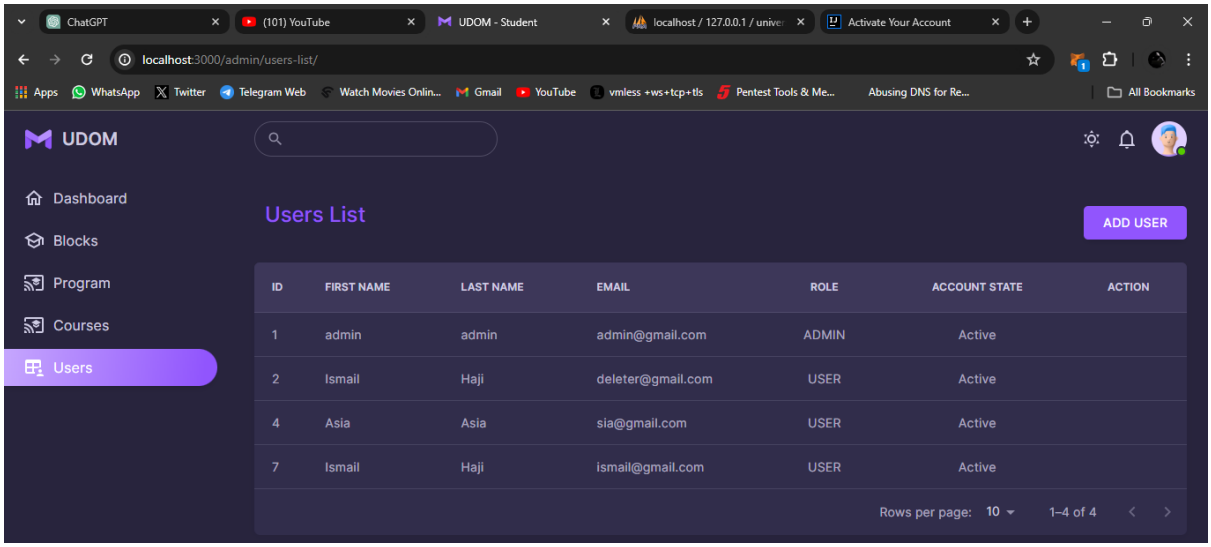


ADMIN USER (ADMIN)

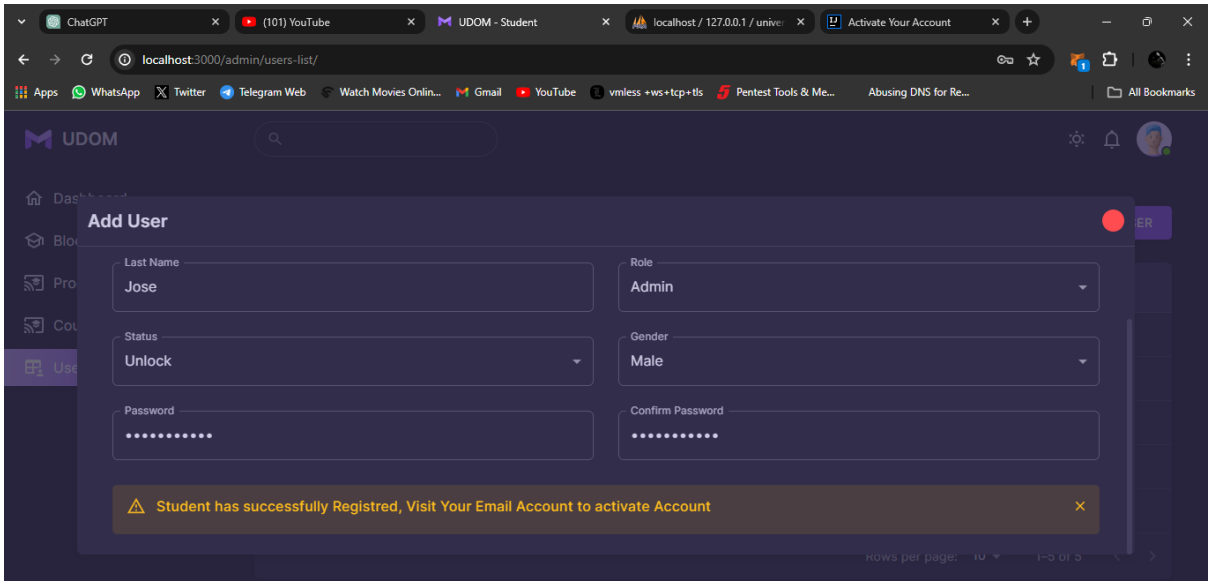
This one admin account into his account he can add users, block and rooms for student

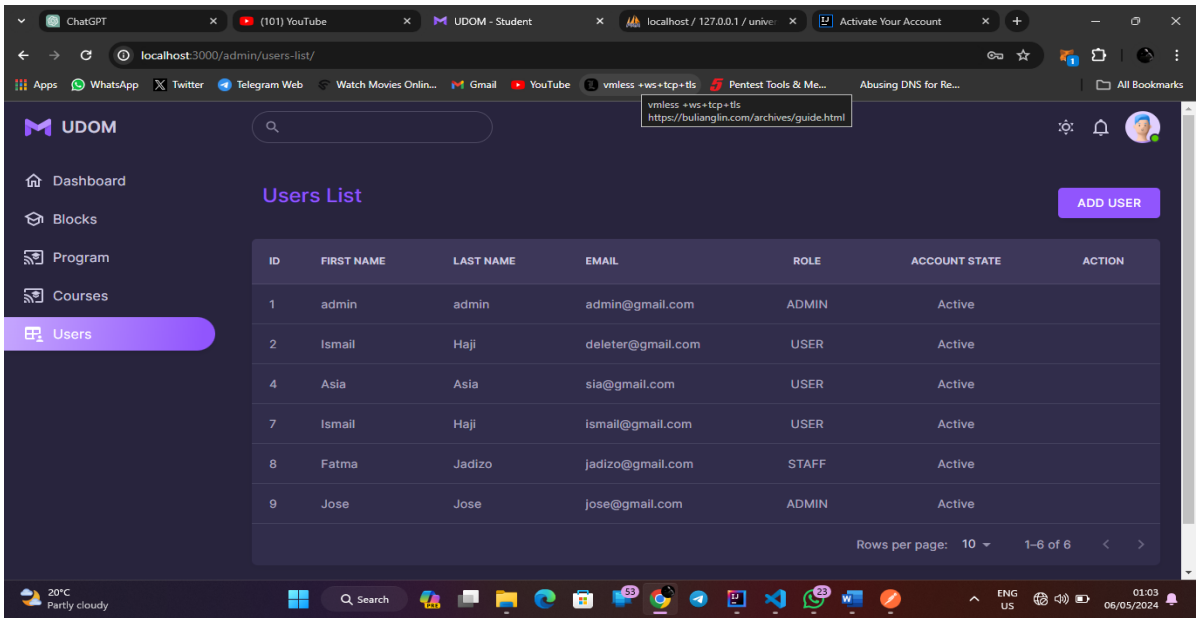


Here is the list of users in the system

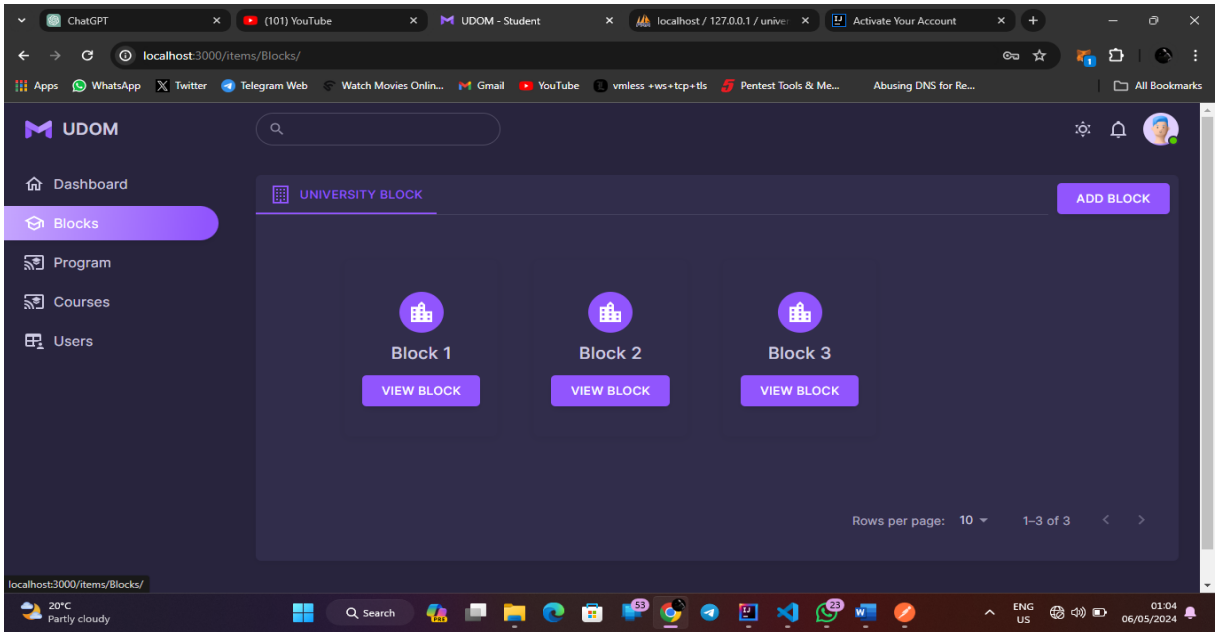


When user click add user, he will be able to add new user

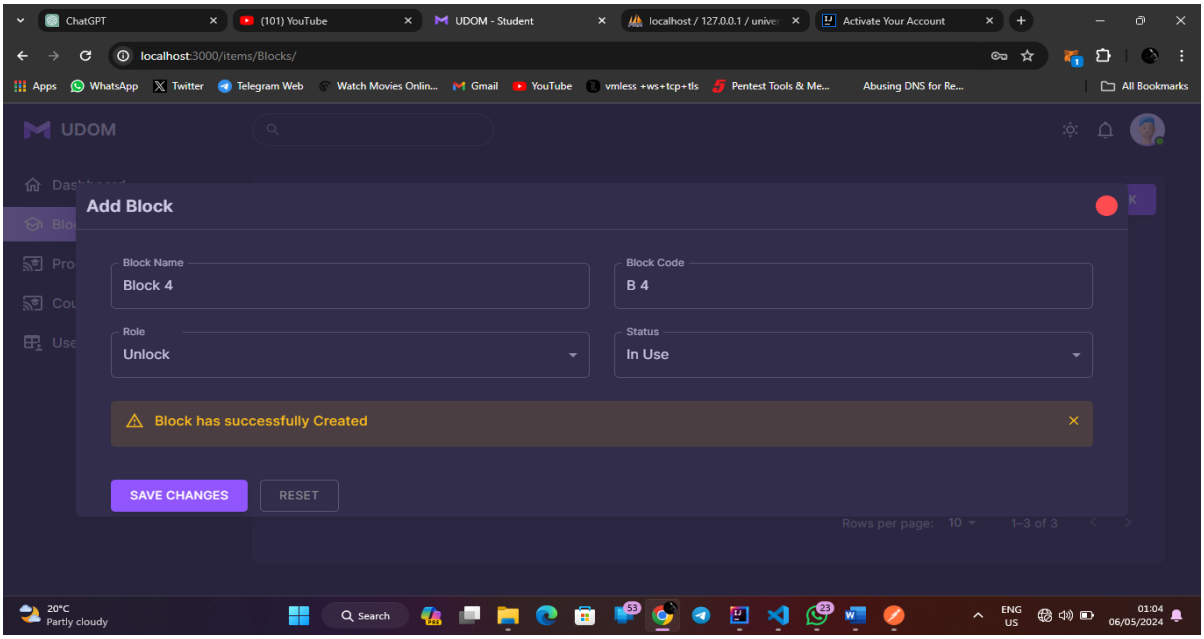


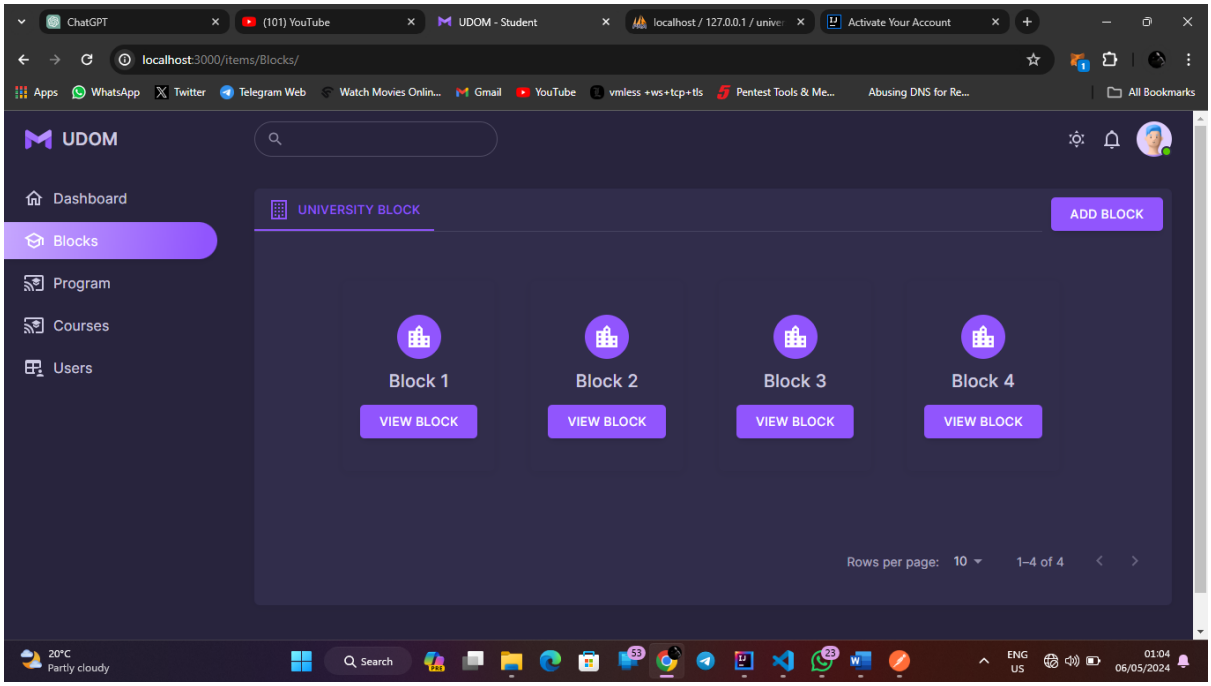


List of blocks

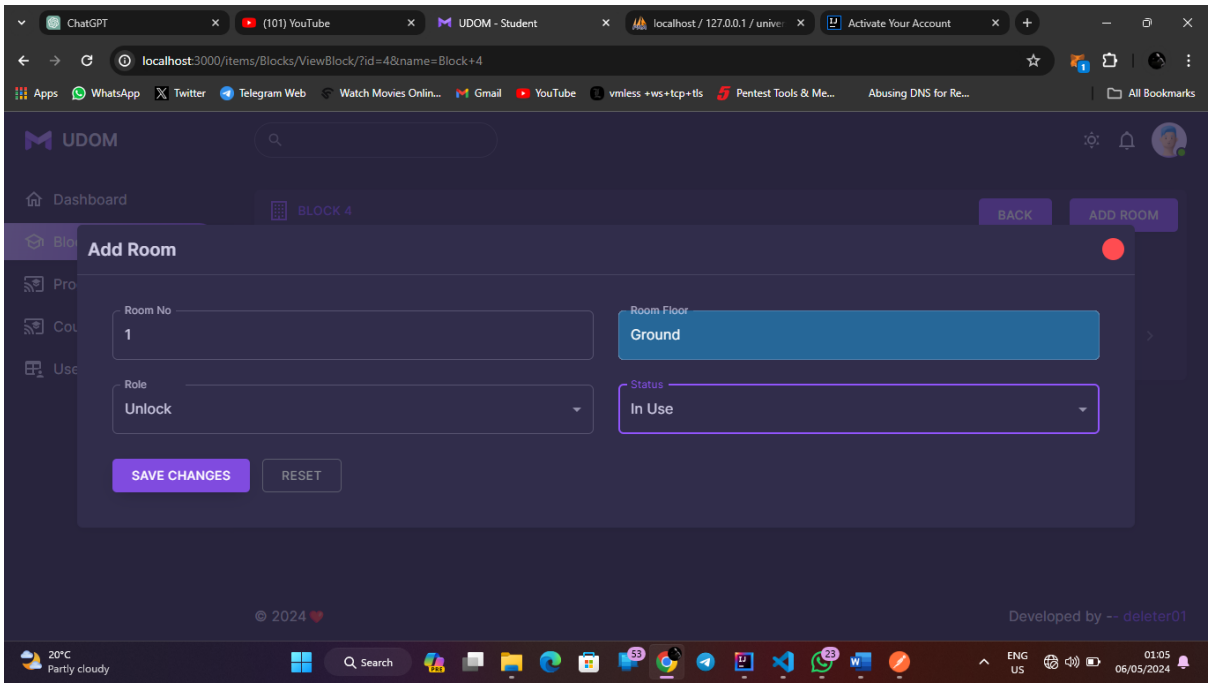


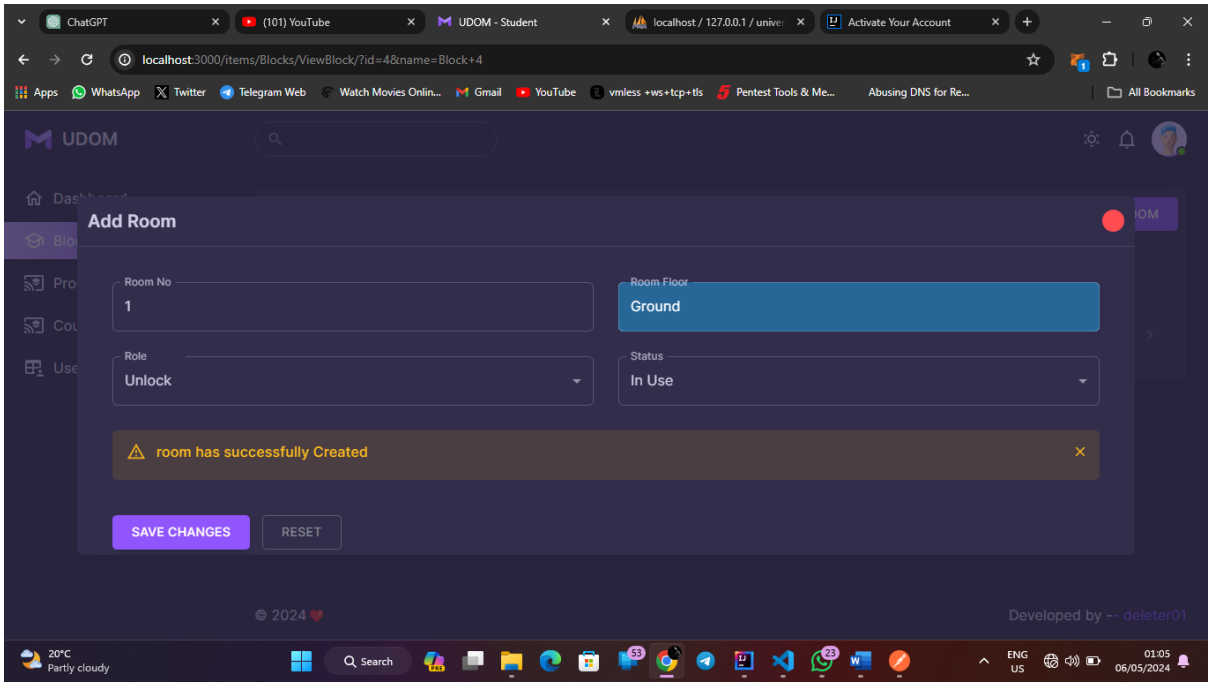
Adding blocks

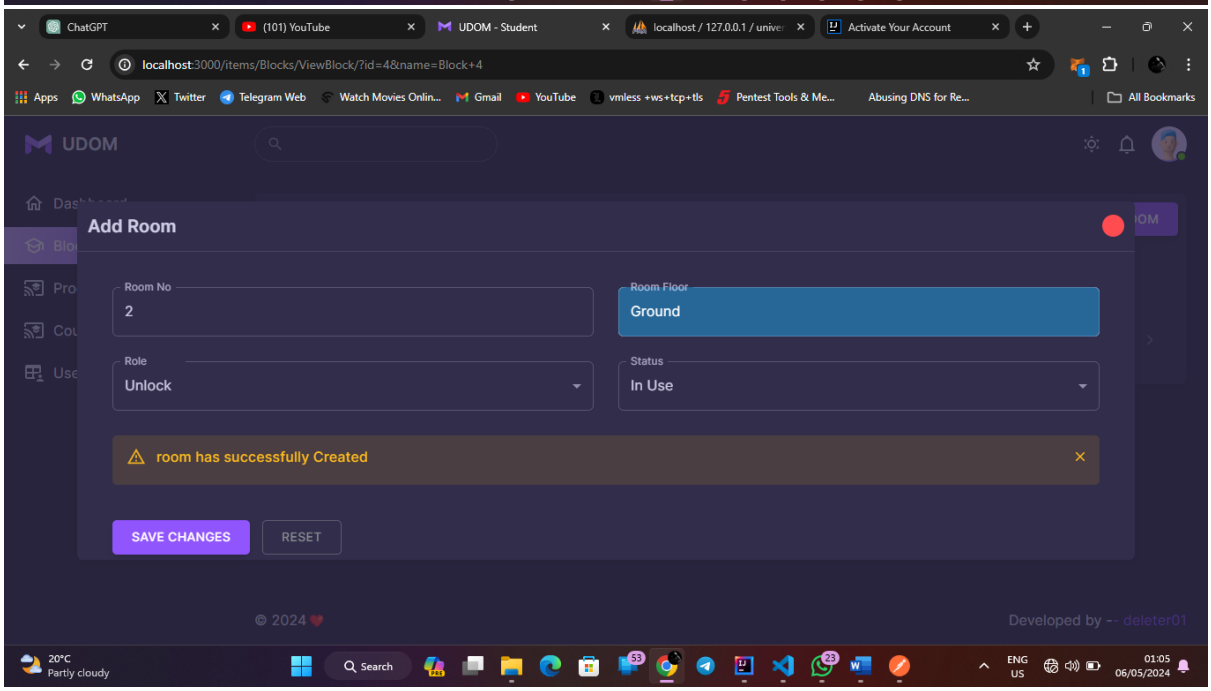


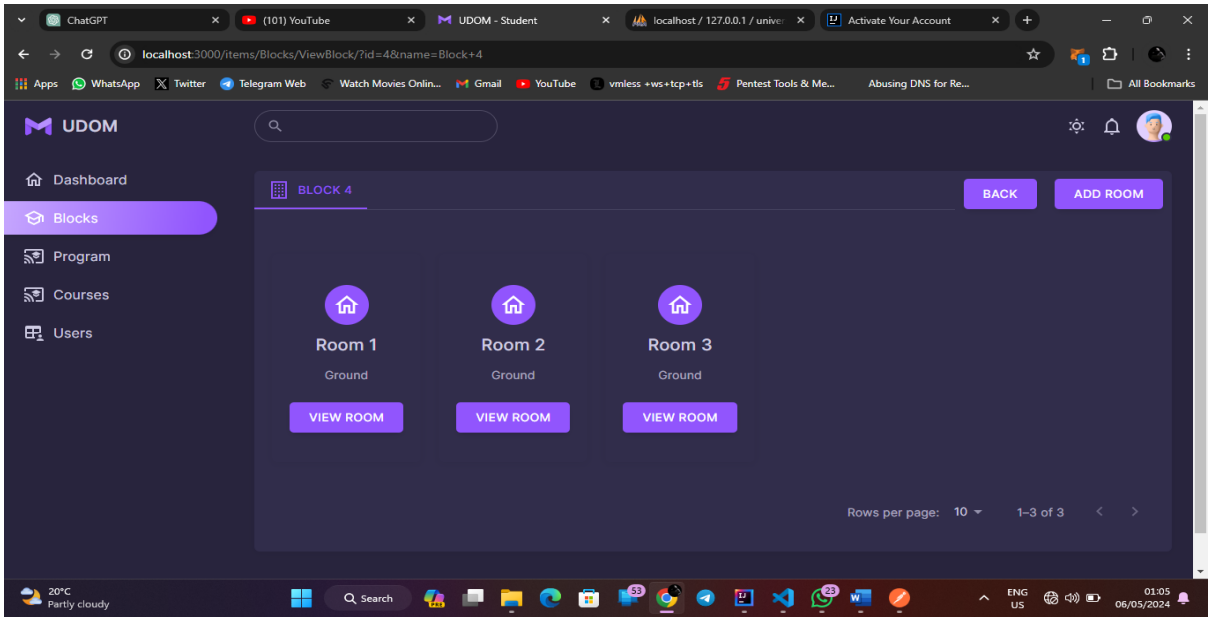

UDOM
Dashboard
Blocks
Program
Courses
Users
UNIVERSITY BLOCK
ADD BLOCK
Block 1
Block 2
Block 3
Block 4
VIEW BLOCK
VIEW BLOCK
VIEW BLOCK
VIEW BLOCK
Rows per page: 10
1–4 of 4

Adding Room


UDOM
Dashboard
BLOCK 4
BACK
ADD ROOM
Add Room
Room No
1
Room Floor
Ground
Role
Unlock
Status
In Use
SAVE CHANGES
RESET
© 2024
Developed by -- deleter01


ChatGPT
(101) YouTube
UDOM - Student
localhost / 127.0.0.1 / univer
Activate Your Account
localhost:3000/items/Blocks/ViewBlock/?id=4&name=Block+4
Apps
WhatsApp
Twitter
Telegram Web
Watch Movies Onlin...
Gmail
YouTube
vmless +ws+tcp+tls
Pentest Tools & Me...
Abusing DNS for Re...
All Bookmarks
UDOM
Add Room
Room No
1
Room Floor
Ground
Role
Unlock
Status
In Use
room has successfully Created
SAVE CHANGES
RESET
© 2024
Developed by -- deleter01
20°C
Partly cloudy
Search
ENG
US
01:05
06/05/2024

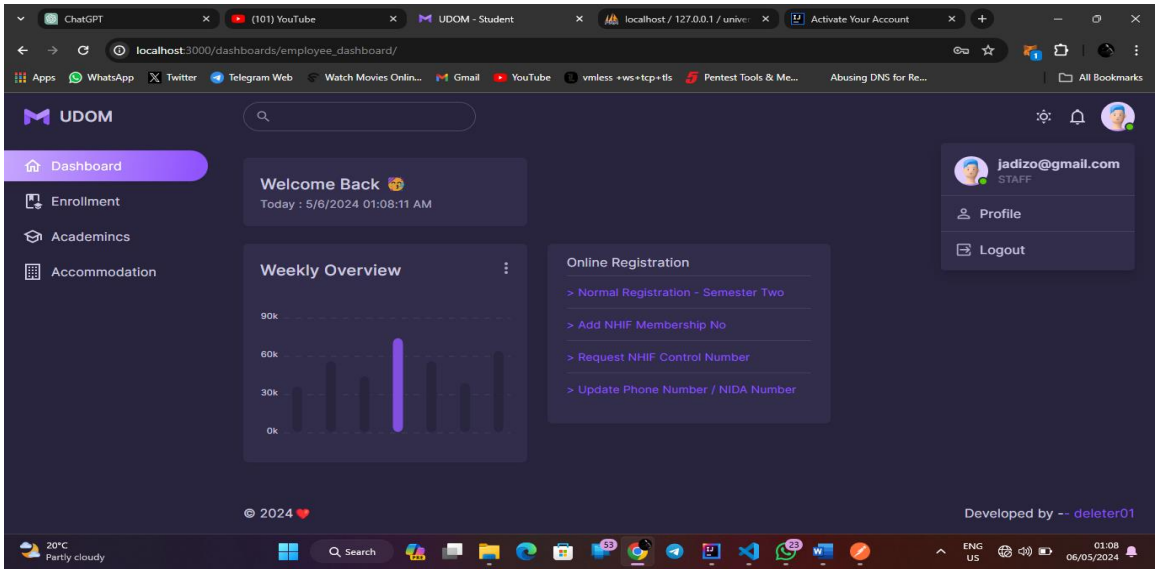

ChatGPT
(101) YouTube
UDOM - Student
localhost / 127.0.0.1 / univer
Activate Your Account
localhost:3000/items/Blocks/ViewBlock/?id=4&name=Block+4
Apps
WhatsApp
Twitter
Telegram Web
Watch Movies Onlin...
Gmail
YouTube
vmless +ws+tcp+tls
Pentest Tools & Me...
Abusing DNS for Re...
All Bookmarks
UDOM
Add Room
Room No
2
Room Floor
Ground
Role
Unlock
Status
In Use
room has successfully Created
SAVE CHANGES
RESET
© 2024
Developed by -- deleter01
20°C
Partly cloudy
Search
ENG
US
01:05
06/05/2024



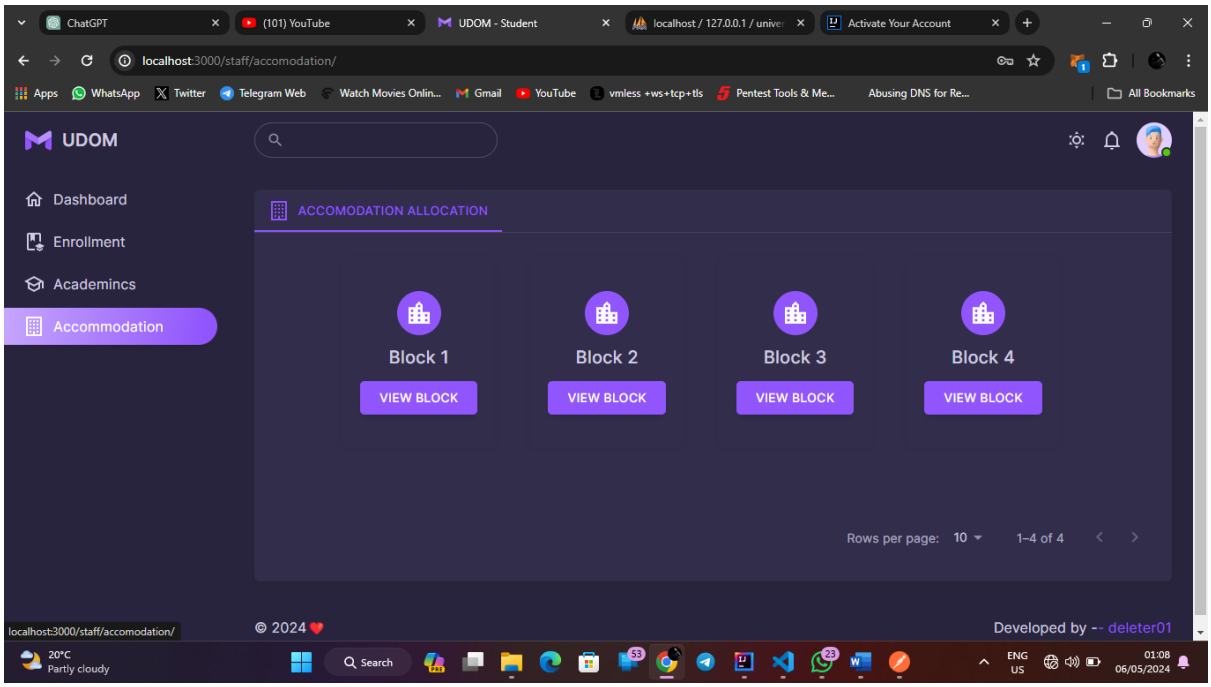
STAFF USER (DEAN OF STUDENT)

This is the staff user who can assign user to specific room.

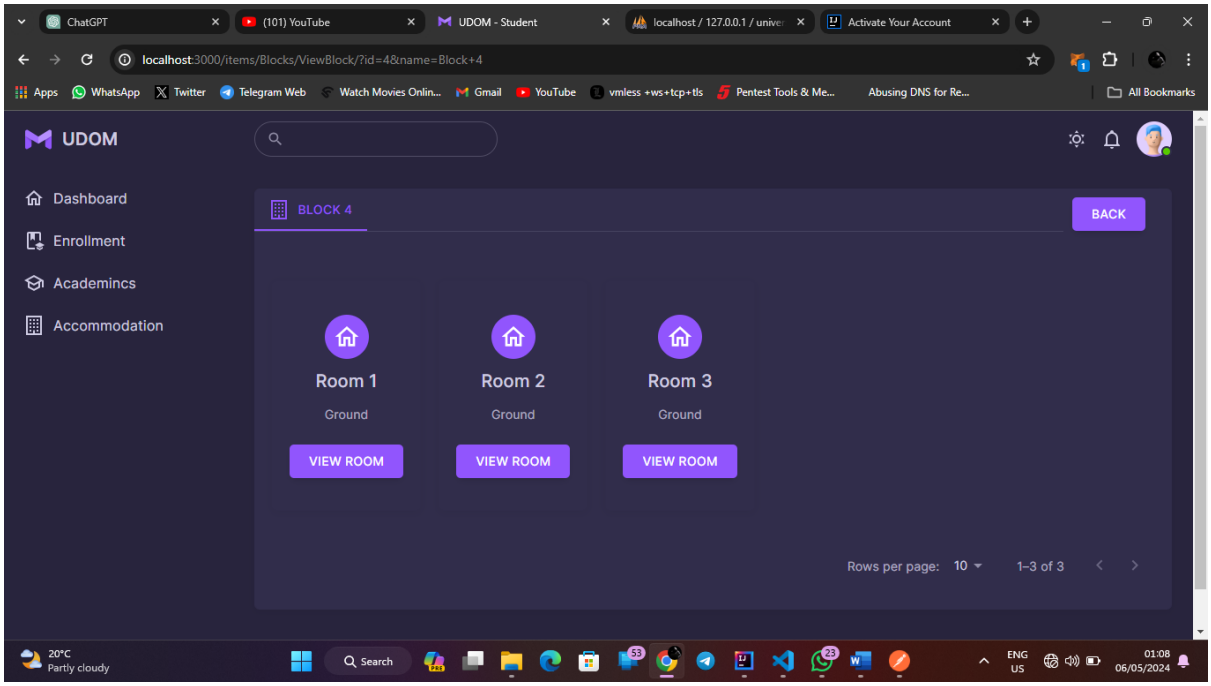
Login as staff and access his dashboard



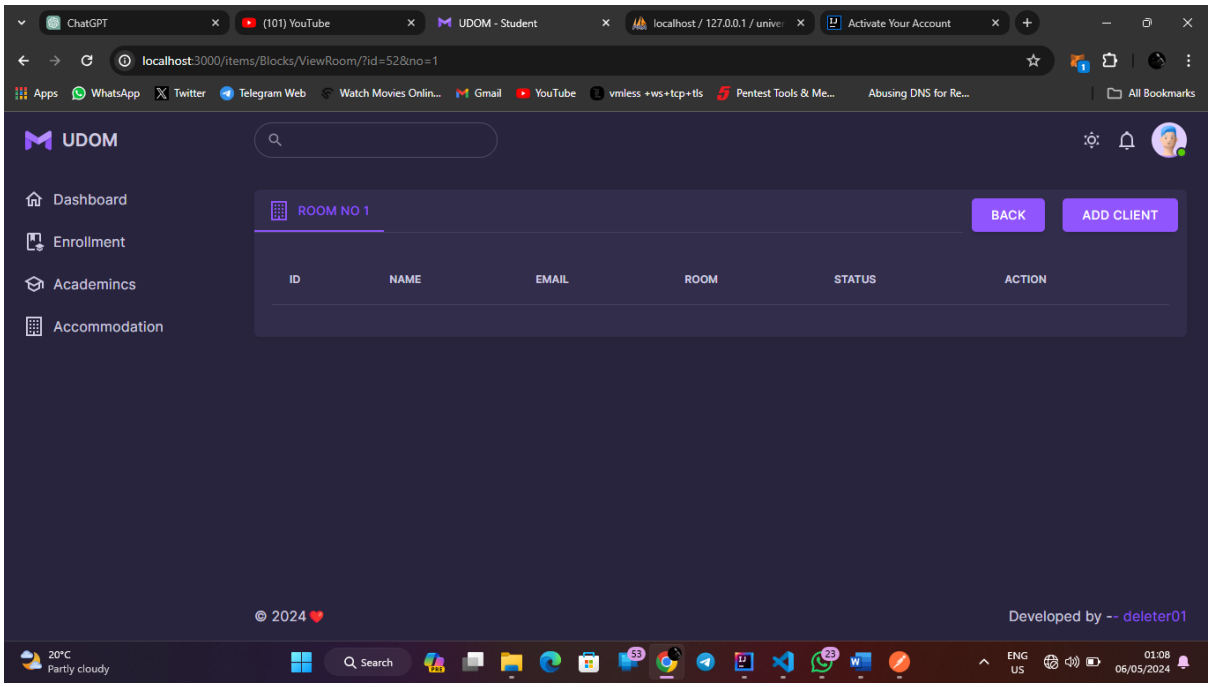
Step One: See list of blocks in the system



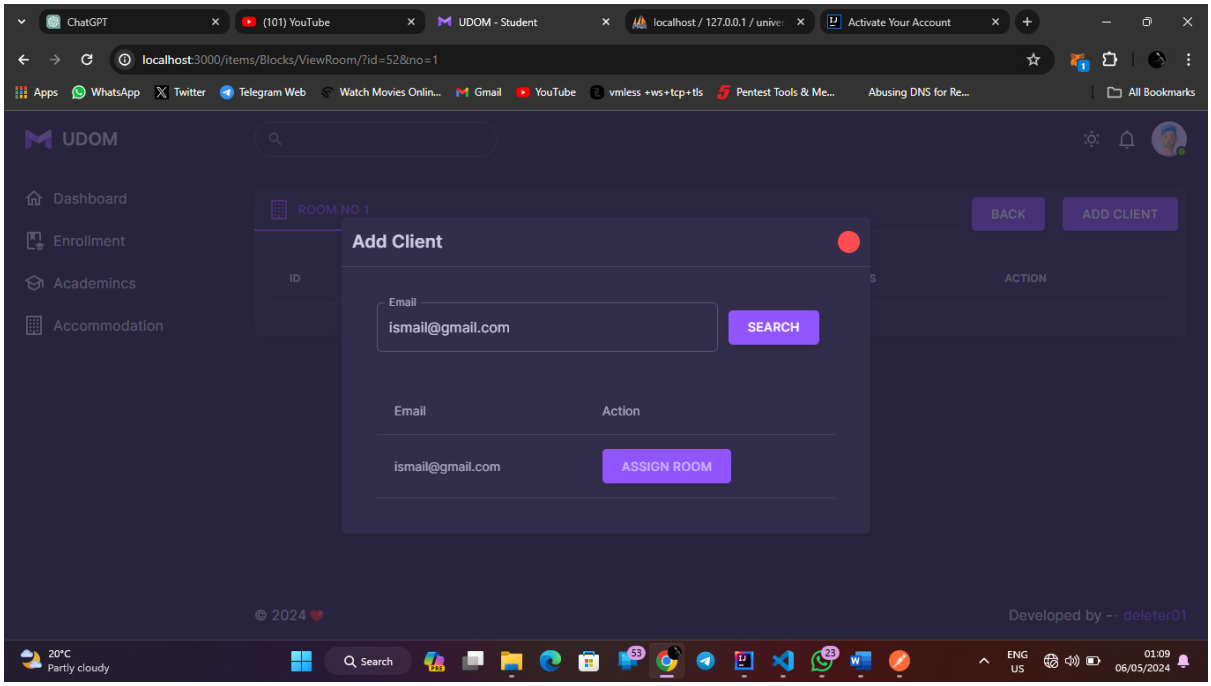
Step Two: See list of room respective to block



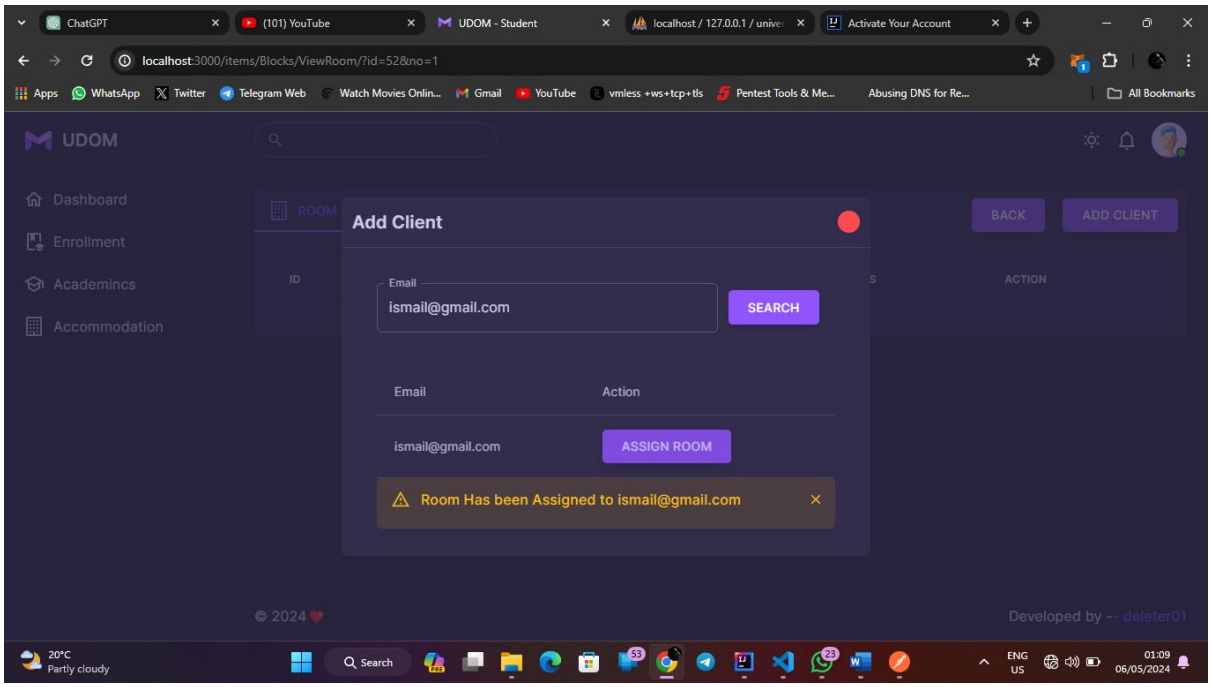
Step Three: See list of users in respective room

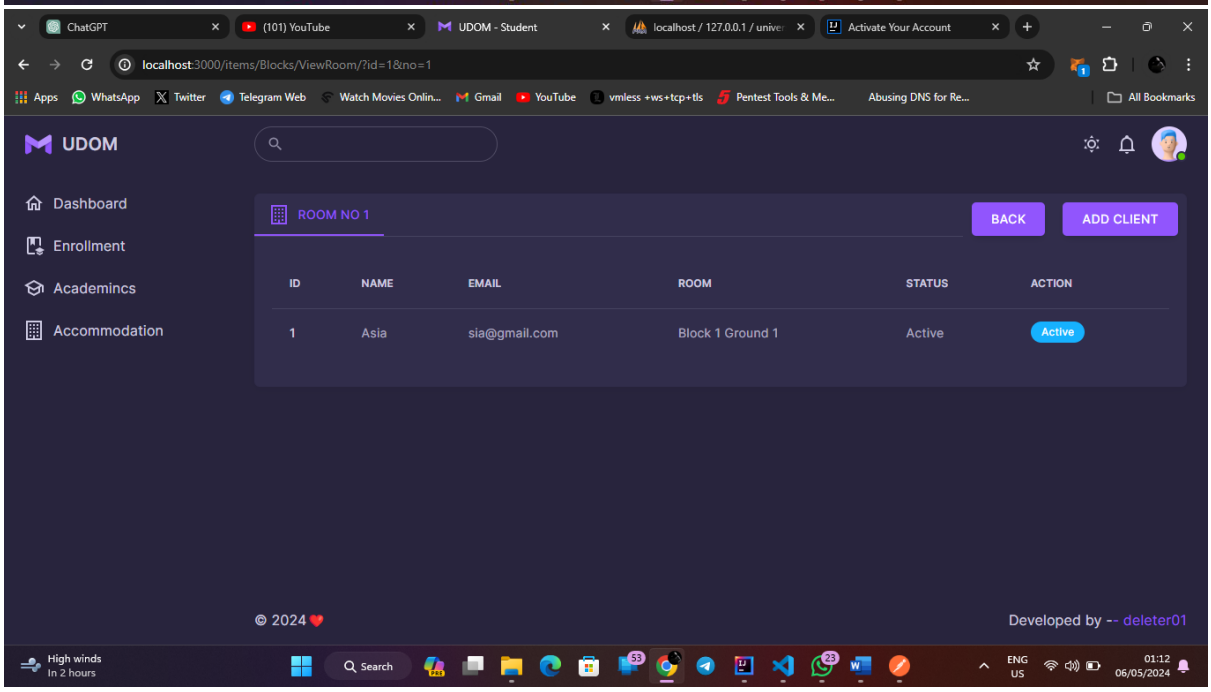


Step Four: Search student or user need to assign to the room



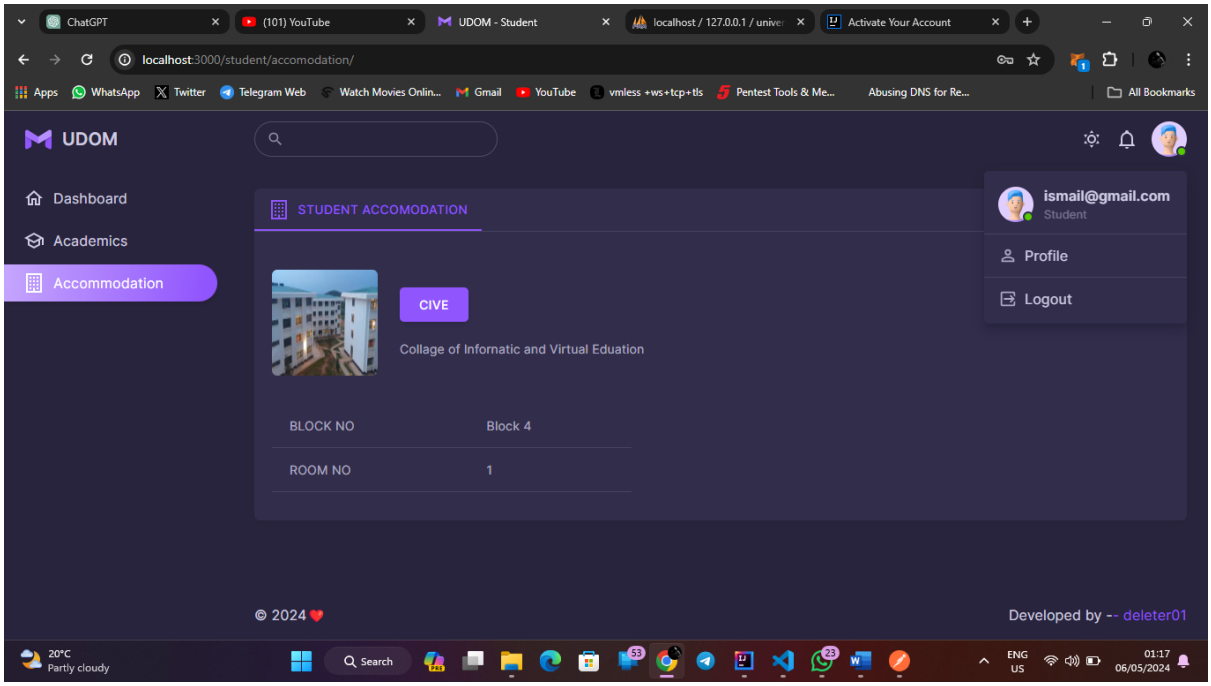
Step Five: Click button assign to assign to the room





NORMAL USER (STUDENT)

The Last user is Normal User who can the status of Room allocation



CONCLUSION

The implementation of the Login Authentication system signifies a pivotal advancement in the realm of student management systems, prioritizing security, efficiency, and user experience. Utilizing cutting-edge technologies such as React JS and Spring Boot for front end and back-end development, respectively, the system offers a seamless and intuitive interface while fortifying authentication processes with robust security mechanisms. From token-based authentication to role-based access control, every facet of the system is meticulously designed to safeguard user data and ensure smooth access to essential functionalities. Moreover, the system's functionality extends beyond authentication, empowering administrators, staff, and users with a comprehensive suite of features tailored to their roles and responsibilities, thereby streamlining administrative tasks and enhancing user engagement.

In conclusion, the Login Authentication system stands as a testament to the collaborative efforts and innovative thinking of the development team, embodying the transformative potential of modern authentication solutions in educational institutions. With its robust foundation, commitment to security, and user-centric design, the system not only addresses the immediate challenges of user authentication but also sets a precedent for future advancements in student management systems. As educational institutions navigate the complexities of digital transformation, the Login Authentication system emerges as a beacon of progress, ensuring the integrity, accessibility, and security of user data in an ever-evolving technological landscape.

REFERENCES

- React JS Documentation: link
- Spring Boot Documentation: link
- MySQL Documentation: [link](https://dev.mysql.com/doc/)
- JSON Web Tokens (JWT) Documentation: [link](https://jwt.io/introduction/)
- Spring Security Documentation: [link](https://spring.io/projects/spring-security)
- Medium: link
- Stack Overflow: link
- React JS Crash Course by Traversy Media: YouTube link
- Spring Boot Tutorial by Telusko: YouTube link
- https://github.com/maildev/maildev
- https://www.npmjs.com/package/nodemailer
- https://github.com/ritish78/Spring-Email-Verification/blob/master/README.md
- https://www.youtube.com/watch?v=4SQiTXHYrnw