Mini Project Report
On

ProCoders (Educational Webpage)

Bachelor of Technology

In

Computer Science and Engineering

By

VANSH SHARMA

2306950100173

.
.

Under the Supervision of

SIKHA RATHI

Assistant Professor

Department of Computer Science and Engineering, SRGC-IC

SHRI RAM GROUP OF COLLEGES, MUZAFFARNAGAR

AFFILIATED TO

Dr A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, UTTAR PRADESH, LUCKNOW
(2024-25)

DECLARATION

I, **VANSH SHARMA**, a student of the Department of Computer Science and Engineering, in the second year of the Bachelor of Technology (B.Tech) program, hereby declare that the mini project titled "**ProCoders (Educational Webpage)"** is the result of my own work and research. The work presented in this report has been carried out under the supervision of **Ms. Sikha Rathi**, Assistant Professor, Department of Computer Science and Engineering, SRGC-IC.

This report is an original work and has not been submitted to any other university or institution for the award of any degree or diploma.

Date: 24 - Jan - 2025
Place: Gandhi Colony Lane-17, Muzaffarnagar

Signature
VANSH SHARMA
2306950100173
B.Tech CSE, Second Year
SRGC- IC

CERTIFICATE

This is to certify that the mini project titled **"ProCoders (Educational Webpage)"** is an authentic work carried out by **VANSH SHARMA**, a student of the Department of Computer Science and Engineering, in the second year of the Bachelor of Technology (B.Tech) program at **SHRI RAM GROUP OF COLLEGES, MUZAFFARNAGAR**. The project has been completed as a part of the academic requirements for the mini project course.

The work has been conducted under the able guidance and supervision of **Ms. Sikha Rathi**, Professor, Department of Computer Science and Engineering, SRGCIC, who provided valuable insights and support throughout the project.

This project is original and has not been submitted elsewhere for the award of any other degree

Student's Signature
VANSH SHARMA
B.Tech CSE, Second Year
2306950100173
SRGC-IC

Supervisor's Signature
Ms. Sikha Rathi
Assistant Professor
CSE Department
SRGC-IC

ACKNOWLEDGEMENTS

It gives us a great sense of pleasure to present the report of the B. Tech Mini Project undertaken during B. Tech. Second Year. We owe special debt of gratitude to **Ms. Sikha Rathi** Department of Computer Science & Engineering, Shri Ram Group of Colleges, Muzaffarnagar for his constant support and guidance throughout the course of our work. His/her sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only his/her cognizant efforts that our endeavors have seen light of the day.

We also take the opportunity to acknowledge the contribution for his/her full support and assistance during the development of the project. We also would not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our mini project. Last but not the least, we acknowledge our parents and friends for their emotional and moral support during the work that helped a lot in completion off the work

ABSTRACT

ProCoder is an innovative and scalable web-based platform designed to revolutionize online learning through a combination of interactive, secure, and user-friendly features. Inspired by platforms like GeeksforGeeks and Javatpoint, ProCoder integrates real-time chat rooms, robust authentication mechanisms, dynamic quiz practice modules, and categorized text tutorials to meet the needs of modern learners. Built with Django as the backend framework and leveraging HTML, CSS, and JavaScript for the frontend, ProCoder ensures seamless performance across diverse operating systems, with a preference for Linux servers to maximize stability and security.

What is ProCoder?

ProCoders is a comprehensive online learning platform designed to bridge gaps in existing solutions. The platform is scalable and adaptable, making it suitable for a diverse range of users and educational scenarios.

Why ProCoder?

The project addresses critical challenges in the e-learning domain, such as:

1. **Lack of Real-Time Interaction:** Enabling real-time communication through WebSocket-powered chat rooms fosters collaboration and engagement.
2. **Inadequate Security:** Incorporating multi-factor authentication and robust encryption ensures data protection and user trust.
3. **Limited Engagement Features:** Adaptive quizzes and categorized tutorials make learning interactive and efficient.
4. **Scalability Issues:** ProCoders's architecture is optimized for high performance under heavy loads, ensuring reliability and usability.

Extensive testing has validated its ability to provide uninterrupted service and meet the evolving demands of online education. Future developments, such as AI-driven recommendations and encrypted chatrooms, will further enhance its utility and user experience.

LIST OF FIGURES

LIST OF TABLES

Chapter 1: INTRODUCTION

The purpose of this project, **ProCoders**, is to design and develop a scalable and user-friendly platform with essential features such as chat rooms, secure login/authentication, quiz practice modules, and text tutorials. Our inspiration stems from platforms like **GeeksforGeeks** and **Javatpoint**, which provide high-quality educational content. **ProCoder**s is designed to run seamlessly on various operating systems, with a preference for Linux servers for enhanced performance and security.

Modern educational needs require platforms that are not only informative but also interactive and secure. With the exponential growth of online learning, users demand a seamless experience that combines the best of functionality and design. **ProCoder**s bridges these gaps by focusing on real-time communication, secure user authentication, interactive quizzes, and accessible tutorials, all of which are key components of an effective learning environment.

FEATURES OVERVIEW:

- **Chat Room**: Enables real-time communication among users. This feature is particularly beneficial for collaborative learning, peer discussions, and resolving queries in a live environment.
- **Secure Login and Authentication**: Ensures the safety and privacy of user data. By leveraging Django's authentication system and additional security layers, user accounts are well-protected against unauthorized access.
- **Quiz Practice**: Provides an interactive space for users to practice quizzes and enhance their skills. Quizzes are designed to be dynamic, adaptive, and engaging to cater to diverse learning needs.
- **Text Tutorials**: Delivers structured, easy-to-read tutorials on various topics. Tutorials are organized categorically to enable efficient learning and quick access to desired topics.

The platform's architecture emphasizes scalability, allowing it to handle increased user loads efficiently. This ensures that as **ProCoders** grows, its performance remains consistent and reliable.

Chapter 2: PROBLEM STATEMENT

Online learning platforms often lack seamless interaction capabilities, scalability, and security features. **ProCoders** addresses these challenges by providing a comprehensive solution tailored to meet the needs of modern learners and educators.

The identified challenges include:

1. **Lack of Real-Time Interaction**: Many platforms fail to provide robust communication tools, leaving users disconnected.
2. **Inadequate Security Measures**: User data and conversations are often vulnerable to breaches and unauthorized access.
3. **Limited Interactive Features**: Static content delivery leads to a lack of engagement and retention among learners.
4. **Difficulty in Scalability**: Platforms struggle to maintain performance as user demand increases.

ProCoders directly tackles these issues by integrating real-time communication, secure login mechanisms, engaging quiz modules, and scalable architecture. This ensures a seamless and secure user experience that encourages active participation and continuous learning.

TECHNOLOGIES USED AND SCREENSHOTS OF USER INTERFACE

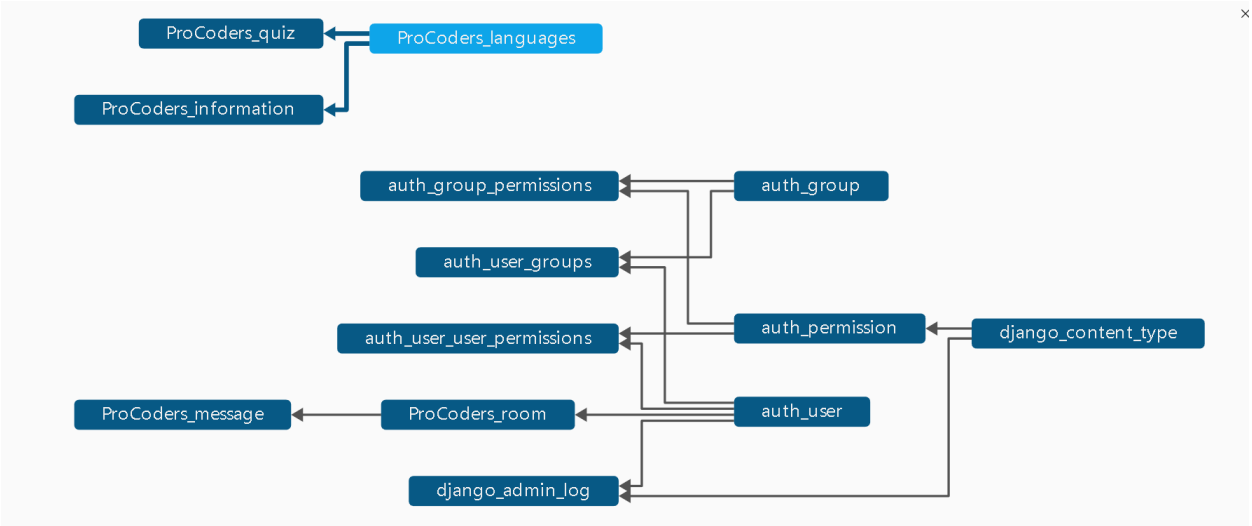
TOOLS AND TECHNOLOGIES:

- **Backend**: Django Framework for rapid development and robust backend operations.

- **Frontend**: HTML, CSS, and JavaScript to create an intuitive and responsive user interface.

- **Database**: SQLite (default) for development purposes, with an option to switch to PostgreSQL for production environments to enhance scalability and performance

  This represent the relational database tables link to each other.

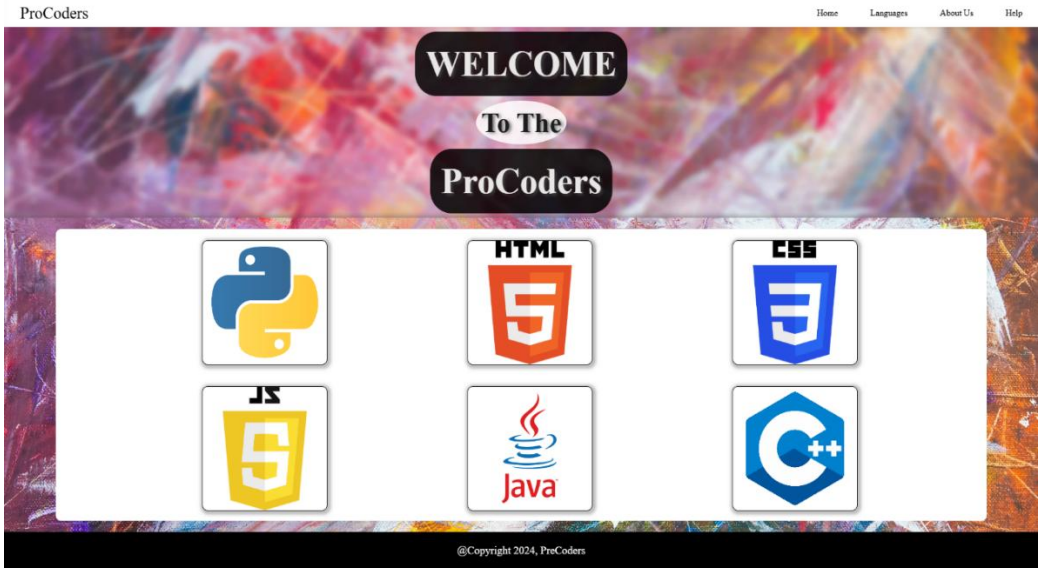


- **Server**: Linux (preferred) for its stability, security, and compatibility with Django-based applications. The platform also runs smoothly on Windows and macOS. We can use either Apache and nginx for deployment.

USER INTERFACE PREVIEWS:

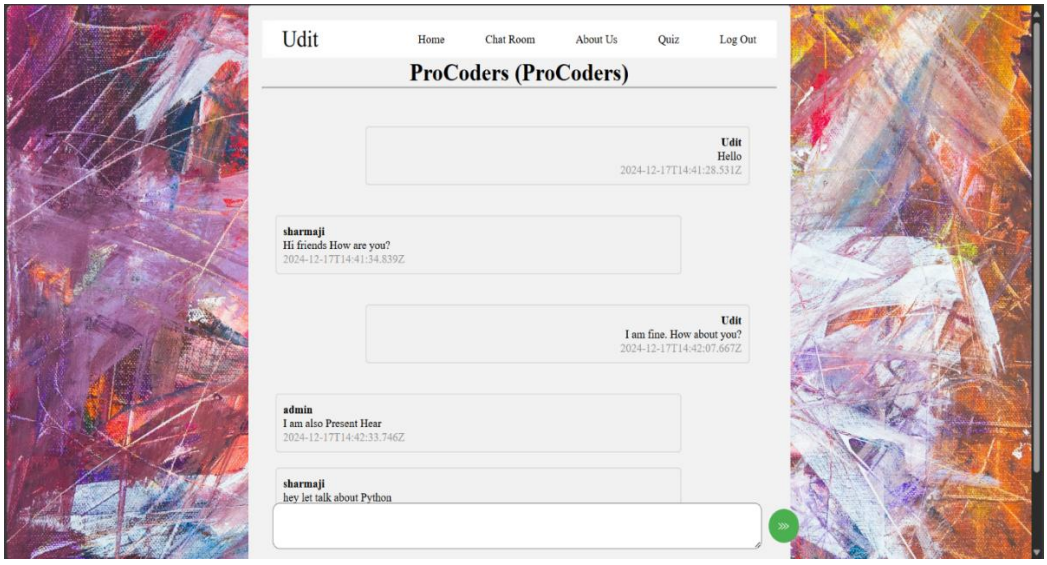
Clean HomePage

The clean homepage design ensures intuitive navigation, showcasing key features, resources, and tools. With a focus on simplicity and user experience.



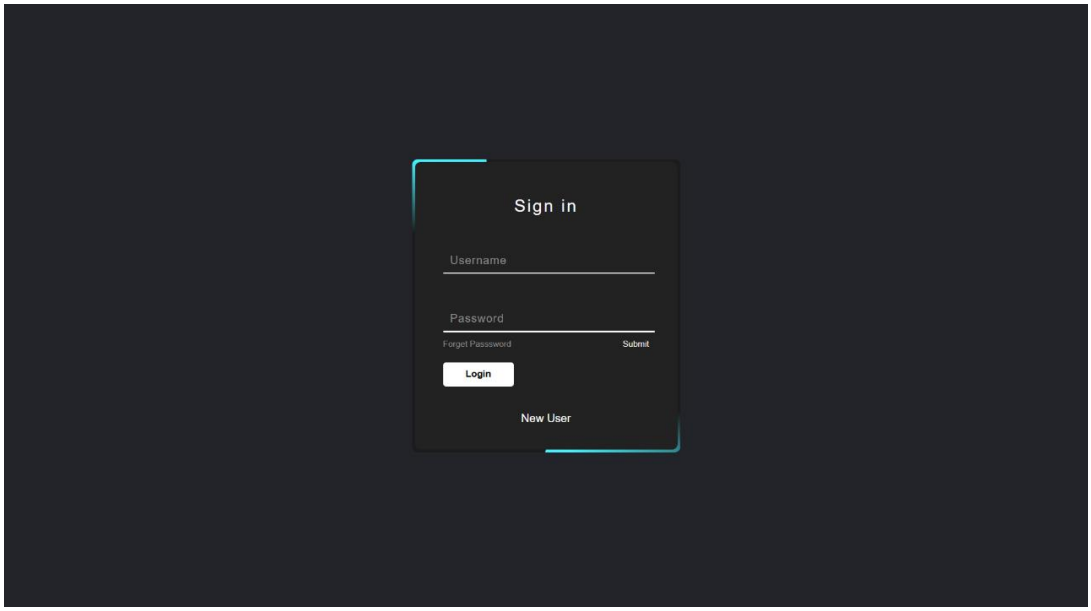
Chat Room Interface:

The chat room interface is designed for real-time interaction. It supports features like live messaging, message notifications, and user status updates.



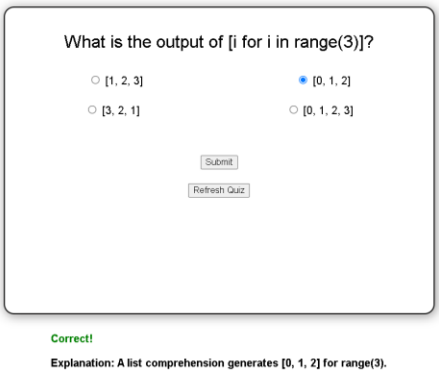
Secure Login Page:

The secure login page includes multi-factor authentication and password recovery options, ensuring a high level of user data protection



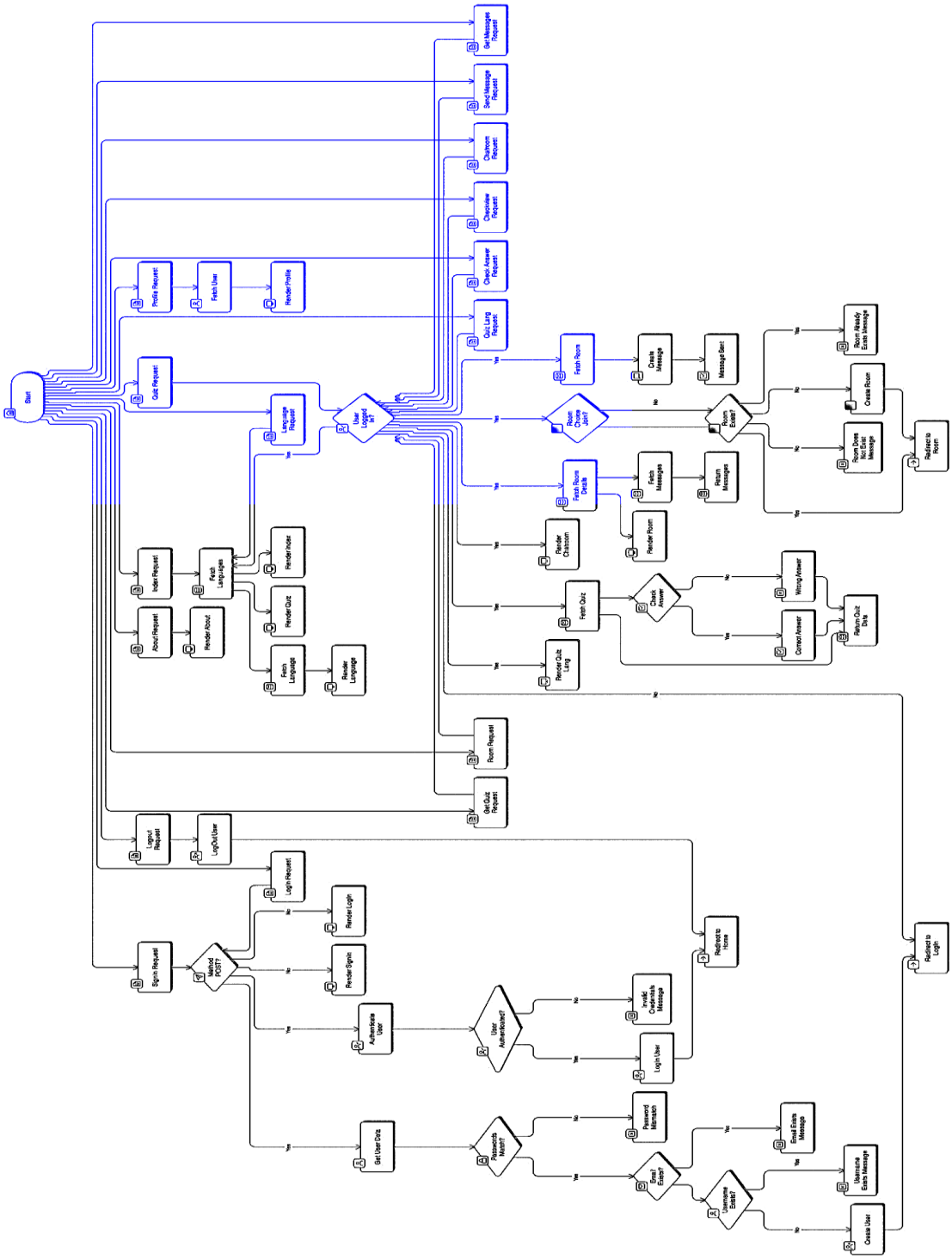
Quiz Section:

The **Quiz Section** offers interactive, engaging quizzes to test your knowledge and skills. Tailored for learners of all levels, it provides instant feedback and progress tracking.



Chapter 3: IMPLEMENTATION AND RESULTS

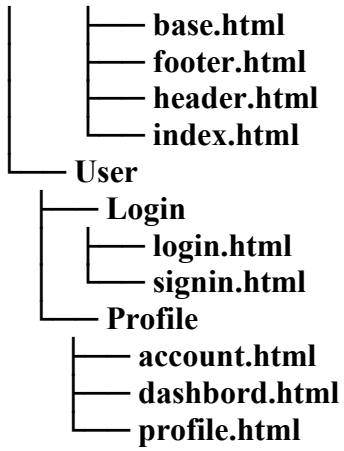
IMPLEMENTATION(FLOWCHART):



Tree :

```
ProCoders
├── autofeeddata.py
├── css.txt
├── c++.txt
├── db.sqlite3
├── html.txt
├── javamcq.txt
├── js.txt
├── manage.py
├── MiniProject
│   ├── asgi.py
│   ├── __init__.py
│   ├── __pycache__
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── ProCoders
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   │   ├── 0001_initial.py
│   │   ├── __init__.py
│   │   └── __pycache__
│   ├── models.py
│   ├── __pycache__
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── pythonmcq.txt
├── sqlmcq.txt
├── static
│   ├── Assests
│   │   ├── bdy1.jpg
│   │   ├── bdy.jpg
│   │   ├── close_24dp_000000_FILL0_wght400_GRAD0_opsz24.svg
│   │   ├── CPP.png
│   │   ├── CSS.png
│   │   ├── HTML.png
│   │   ├── Java.png
│   │   ├── JavaScript.png
│   │   ├── menu_24dp_000000_FILL0_wght400_GRAD0_opsz24.svg
│   │   └── Python.png
│   ├── styles.css
```

```
    ├── Supports
    │   ├── About
    │   │   ├── script.js
    │   │   ├── style.css
    │   │   └── Supports
    │   │       ├── About1.jpg
    │   │       ├── Content.jpg
    │   │       └── Teams.jpg
    │   ├── Information
    │   │   └── variables.png
    │   └── Language
    │       ├── C.png
    │       ├── CPP.png
    │       ├── CSS.png
    │       ├── HTML.png
    │       ├── Java.png
    │       ├── JavaScript.png
    │       ├── Python.png
    │       └── styles.css
    └── User
        ├── Login
        │   └── animlogin.css
        ├── Profile
        │   └── style.css
        └── work under progress
            └── sign_up.jsp
└── templates
    ├── Assests
    │   ├── close_24dp_000000_FILL0_wght400_GRAD0_opsz24.svg
    │   └── menu_24dp_000000_FILL0_wght400_GRAD0_opsz24.svg
    ├── ChatRoom
    │   ├── index.html
    │   └── room.html
    ├── content.html
    ├── footer.html
    ├── header.html
    ├── index.html
    ├── Quiz
    │   ├── index.html
    │   └── Language
    │       └── quiz.html
    ├── Supports
    │   ├── About
    │   │   ├── index.html
    │   │   └── Supports
    │   └── Language
```



30 directories, 83 files

Code (views.py):

```
from django.shortcuts import render, redirect
from django.contrib.auth.models import User, auth
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.http import HttpResponse,JsonResponse
from .models import *

# home page
def index(request):
    languages = Languages.objects.all()
    return render(request, 'index.html',{'languages' : languages})

# SignIn page
def signin(request):
    if request.method == 'POST':
        username = request.POST['username']
        email = request.POST['email']
        password = request.POST['password']
        password2 = request.POST['password2']

        if password == password2 :
            if User.objects.filter(email=email).exists():
                messages.info(request,'This Email is Already Registerd')
                return redirect('/User/SignIn')

            elif User.objects.filter(username=username).exists():
```

```
                messages.info(request,'This Username is Already Exist\nplease Try
anoter user name')
                return redirect('/User/SignIn')
            else:
                user = User.objects.create_user(username=username,
password=password, email=email)
                user.save()
                return redirect('/User/LogIn')
        else :
            messages.info(request, 'The Password and Confirm Password did not
match')
            return redirect('/User/SignIn')

    return render(request, 'User/Login/signin.html')

# LogIn page
def login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']

        user = auth.authenticate(username=username, password=password)
        if user is not None:
            auth.login(request, user)
            return redirect('/')
        else:
            messages.info(request,'Invalid Username or Password')
            return redirect('/User/LogIn')
    return render(request, 'User/Login/login.html')

def logout(request):
    auth.logout(request)
    return redirect('/')


# profile page
def profile(request, profile):
    user = User.objects.get(username=profile)
    context = {
        'user' : user,
    }
    return render(request, "templates/User/Profile/profile.html",
context=context)

# about page
```

```python
def about(request):
    return render(request, 'Supports/About/index.html')

# quiz page
@login_required(login_url='/User/LogIn')
def quiz(request):
    languages = Languages.objects.all()
    return render(request, 'Quiz/index.html',{'languages' : languages })

# quiz_lang page
@login_required(login_url='/User/LogIn')
def quiz_lang(request,lang):
    return render(request, 'Quiz/Language/quiz.html', { 'lang' : lang })

@login_required(login_url='/User/LogIn')
def get_quiz(request, lang):
    language = Languages.objects.get(language=lang)
    quiz = Quiz.objects.filter(language=language).order_by('?').first()

    quiz_data = {
        'id': quiz.id,
        'question': quiz.question,
        'option1': quiz.option1,
        'option2': quiz.option2,
        'option3': quiz.option3,
        'option4': quiz.option4,
    }

    return JsonResponse({
        'quiz': quiz_data
    })

@login_required(login_url='/User/LogIn')
def check_ans(request, lang):
    ans = request.GET.get('ans')
    quiz_id = request.GET.get('quiz_id')
    quiz = Quiz.objects.get(id=quiz_id)

    quiz_data = {
        'id': quiz.id,
        'question': quiz.question,
        'option1': quiz.option1,
        'option2': quiz.option2,
        'option3': quiz.option3,
        'option4': quiz.option4,
```

```
        'answer': quiz.answer,
        'explanation': quiz.explanation,
    }

    if ans == quiz.answer:
        quiz_data['status'] = 'correct'
    else:
        quiz_data['status'] = 'wrong'


    return JsonResponse({
        'quiz': quiz_data
    })

# language page
def language(request, lang):
    languages = Languages.objects.all()
    l = Languages.objects.get(language=lang)
    return render(request, 'Supports/Language/index.html', context={'lang' : l,
'languages' : languages } )

# chatroom page
@login_required(login_url='/User/LogIn')
def chatroom(request):
        return render(request, 'ChatRoom/index.html')

@login_required(login_url='/User/LogIn')
def room(request,room):
    username = request.GET.get('username')
    room_detail = Room.objects.get(name=room)
    context = {
         'username' : username,
         'room' : room,
         'room_details' : room_detail

    }
    return render(request, 'ChatRoom/room.html', context=context)

@login_required(login_url='/User/LogIn')
def checkview(request):
    room_choice = request.POST['room_choice']
    room = request.POST['room_name']
    username = request.POST['username']

    if room_choice == "join":
```

```
        if Room.objects.filter(name = room).exists():
            return redirect('/chat_room/'+room)
        else :
            return HttpResponse("<H1>Room is Does not Exist</H1>")
    else:
        if not Room.objects.filter(name = room).exists():
            user = User.objects.get(username=username)
            new_room = Room.objects.create(name=room, created_by=user)
            new_room.save()
            return redirect('/chat_room/'+room)
        else :
            return HttpResponse("<H1>Room is Already Exist</H1>")

@login_required(login_url='/User/LogIn')
def send(request):
    message = request.POST['message']
    username = request.POST['username']
    room_id = request.POST['room_id']
    room = Room.objects.get(id=room_id)
    new_message = Message.objects.create(value=message, user=username, room=room)
    new_message.save()
    return HttpResponse("Message send Successfully")

@login_required(login_url='/User/LogIn')
def get_messages(request, room):
    room_details = Room.objects.get(name=room)

    messages = Message.objects.filter(room=room_details.id)
    return JsonResponse({
         "messages" : list(messages.values())
    })
```

Code(urls.py):

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='home'),
    path('Supports/About', views.about, name='about'),
    path('Quiz', views.quiz, name='quiz'),
    path('Quiz/<str:lang>', views.quiz_lang, name='quiz_lang'),
    path('get_quiz/<str:lang>/', views.get_quiz, name='get_quiz'),
    path('check_ans/<str:lang>', views.check_ans, name='check_ans'),
```

```
    path('User/LogIn', views.login, name='login'),
    path('User/SignIn', views.signin, name='signin'),
    path('User/LogOut', views.logout, name='logout'),
    path('User/<str:profile>', views.profile, name='profile'),
    path('ChatRoom', views.chatroom, name='chatroom'),
    path('chat_room/<str:room>', views.room, name='room'),
    path('checkview', views.checkview, name='checkview'),
    path('send', views.send, name='send'),
    path('chat_room/get_messages/<str:room>', views.get_messages,
name='get_messages'),
    path('Supports/Language/<str:lang>', views.language, name='language'),
]
```

Code(models.py):

```
from django.contrib.auth.models import User
from django.db import models

class Languages(models.Model):
    language = models.CharField(max_length=64, unique=True)
    image = models.ImageField(upload_to='static/Supports/Language/', null=True,
blank=True)
    iframe_url = models.TextField(null=True)

    def __str__(self):
        return self.language

class Information(models.Model):
    language = models.ForeignKey(Languages, on_delete=models.CASCADE,
related_name='information')
    heading = models.CharField(max_length=128)
    image = models.ImageField(upload_to='static/Supports/Information/',
null=True, blank=True)
    about = models.TextField()

    def __str__(self):
        return f"{self.heading} ({self.language})"

class Quiz(models.Model):
    language = models.ForeignKey(Languages, on_delete=models.CASCADE,
related_name='quizzes')
    question = models.CharField(max_length=512)
    option1 = models.CharField(max_length=256)
    option2 = models.CharField(max_length=256)
    option3 = models.CharField(max_length=256)
```

```
    option4 = models.CharField(max_length=256)
    answer = models.CharField(max_length=1)
    explanation = models.CharField(max_length=512)

    def __str__(self):
        return f"Quiz in {self.language}: {self.question}"

class Room(models.Model):
    created_by = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="created_rooms")
    name = models.CharField(max_length=256)

    def __str__(self):
        return self.name

class Message(models.Model):
    room = models.ForeignKey(Room, on_delete=models.CASCADE,
related_name="messages")
    value = models.CharField(max_length=2048)
    date = models.DateTimeField(auto_now=True)
    user = models.CharField(max_length=128)


'''class QuizRecord:
    pass
'''
```

Chat Room:

The chat room functionality in **ProCoders** is built using Django channels to support asynchronous communication. WebSocket protocol is employed to ensure efficient and real-time message delivery. Key features include:

- User authentication before accessing chat rooms.
- Dynamic room creation for group or private chats.
- Automatic updates to reflect the active status of participants.

Script(room.html):

```
$(document).ready(function(){

            setInterval(function(){
```

```
                $.ajax({
                    type: 'GET',
                    url : "/chat_room/get_messages/{{room}}",
                    success: function(response){
                        //console.log(response);
                        $("#display").empty();
                        for (var key in response.messages)
                        {
                            if(response.messages[key].user ==
"{{user.username}}") {
                                var temp="<div class='container-darker-
right1'><div class='container-darker-
right'><b>"+response.messages[key].user+"</b><p>"+response.messages[key].value.re
place(/(?:\r\n|\r|\n)/g, '<br>')+"</p><span class='time-
left'>"+response.messages[key].date+"</span></div></div>";
                            }
                            else {
                                var temp="<div class='container-darker-
left'><b>"+response.messages[key].user+"</b><p>"+response.messages[key].value.rep
lace(/(?:\r\n|\r|\n)/g, '<br>')+"</p><span class='time-
left'>"+response.messages[key].date+"</span></div>";
                            }
                            $("#display").append(temp);
                        }
                    },
                    error: function(response){
                        alert('An error occured')
                    }
                });
            },1000);
            })
```

Secure Login and Authentication:

The secure login mechanism leverages Django's built-in authentication framework. Additional features include:

- Integration of multi-factor authentication.
- Implementation of password encryption using PBKDF2 hashing.
- Middleware to log and monitor suspicious login attempts.

Packages :

This setup includes essential packages for a Django project.

- **asgiref==3.8.1**: Provides ASGI support for Django.
- **Django==5.1.3**: The core framework for building web applications.
- **gunicorn==23.0.0**: A WSGI HTTP server for serving the Django app in production.
- **packaging==24.2**: A library for working with Python package versioning.
- **pillow==11.0.0**: Image processing library for Django applications.
- **sqlparse==0.5.2**: A library for parsing SQL queries.
- **tzdata==2024.2**: Time zone data for Python's timezone library.

These libraries help ensure smooth development and deployment of a Django-based web application.

Scalability:

Scalability was a critical aspect of **ProCoders**. Testing was conducted on various devices and platforms to ensure consistent performance. Key outcomes include:

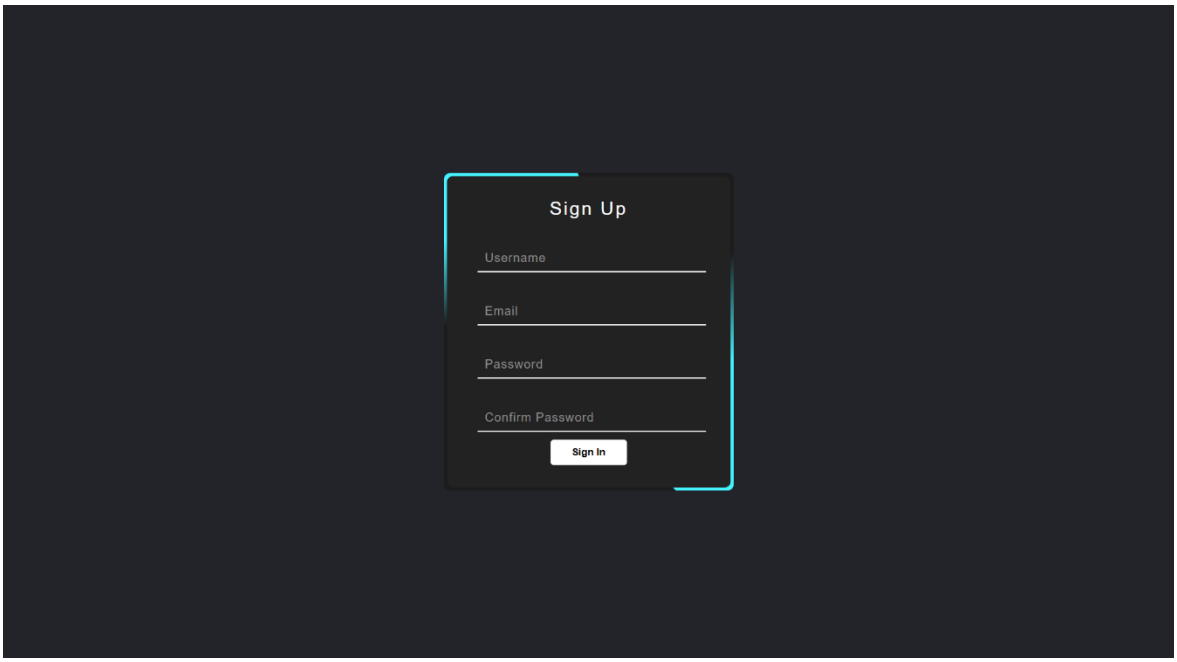
- Seamless handling of concurrent users in the chat room.
- Stable database performance under high query loads.
- Compatibility with multiple operating systems, ensuring broader accessibility.

Results:

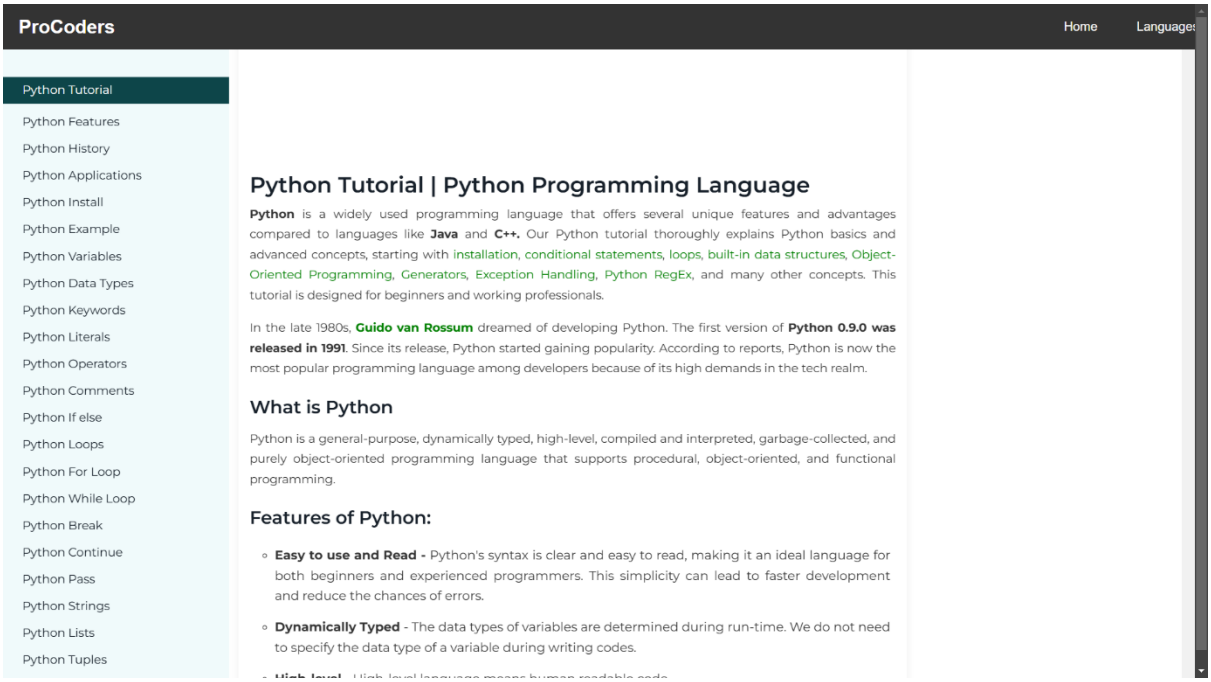
1. Real-time communication among multiple users was achieved without delays or interruptions.
2. Secure login functionality safeguarded user accounts against unauthorized access.
3. Interactive quizzes and tutorial modules successfully enhanced user engagement and learning outcomes.
4. The platform demonstrated excellent scalability, maintaining performance during stress testing

Some other Previews:

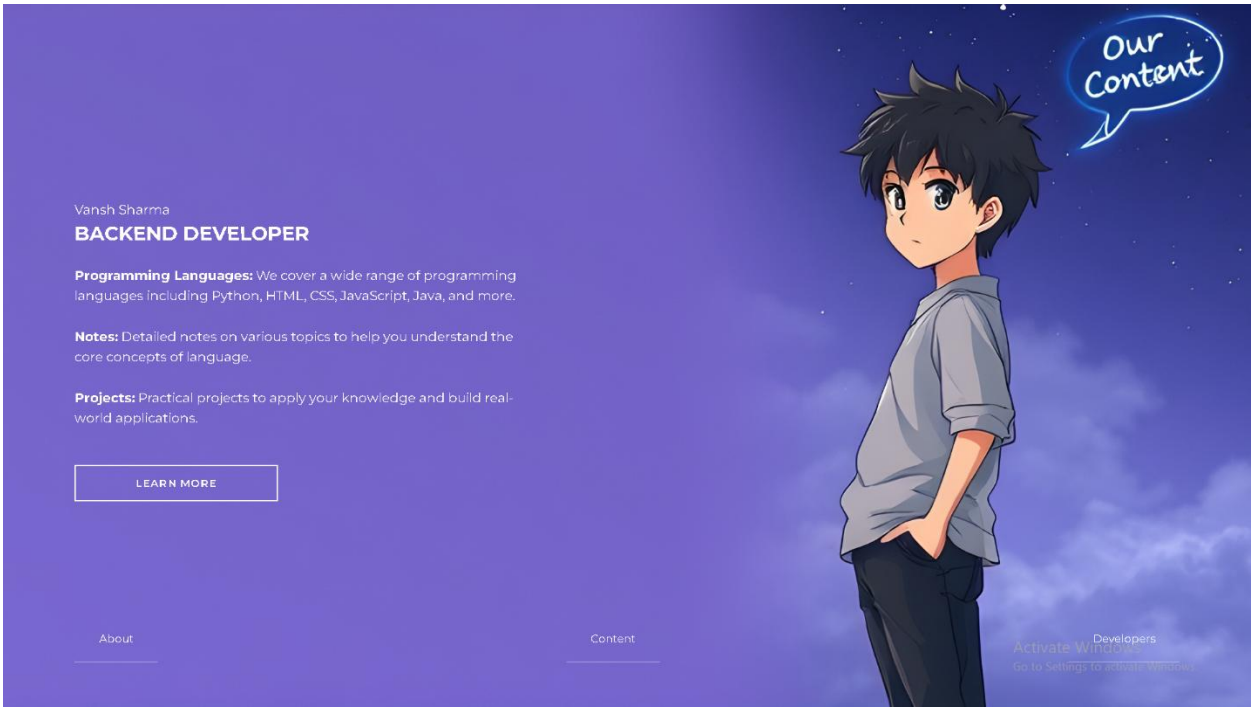
Sign Up:



Tutorials:



About Us:


Vansh Sharma
BACKEND DEVELOPER
Programming Languages: We cover a wide range of programming languages including Python, HTML, CSS, JavaScript, Java, and more.
Notes: Detailed notes on various topics to help you understand the core concepts of language.
Projects: Practical projects to apply your knowledge and build real-world applications.
LEARN MORE
Our Content
About
Content
Developers
Activate Windows
Go to Settings to activate Windows.

Chapter 4: CONCLUSION AND FUTURE WORK

CONCLUSION:

ProCoders successfully addresses the challenges of existing online learning platforms by integrating secure, scalable, and interactive features. By focusing on user needs, **ProCoders** enhances both the learning experience and educational outcomes. The project has laid a strong foundation for future enhancements and expansions.

FUTURE WORK:

1. **Profile Page**:
    - Personalize the user experience with profile management.
    - Allow users to upload avatars and manage personal information securely.
2. **Account Section**:
    - Provide options to adjust settings and preferences.
    - Enable users to view activity logs and manage subscription plans.
3. **Encrypted Chatrooms**:
    - Enhance the privacy of user conversations using end-to-end encryption.
    - Develop tools for moderating and managing secure group discussions.
4. **Advanced Quiz Features**:
    - Include timed quizzes and progress tracking.
    - Provide detailed analytics and recommendations based on quiz performance.
5. **AI-Powered Recommendations**:
    - Use machine learning algorithms to suggest relevant tutorials and quizzes based on user activity.

References

1. Django Official Documentation - Comprehensive guide and references for Django framework.
2. SQLite Documentation - Official documentation for SQLite, a lightweight and powerful database engine used for development.
3. WebSocket Protocol Specification - Detailed specification of WebSocket protocol.
4. GeeksforGeeks Learning Resources - Tutorials and practice questions for coding and computer science topics.
5. PostgreSQL Performance Tuning Guide - Official tips for optimizing PostgreSQL performance.
6. Linux Server Administration Manuals - Essential guides for Linux server administration and operations.