

DJANGO BASED CHAT APPLICATION

A

Project Report

Submitted for the partial fulfillment of

B.Tech degree

in

COMPUTER SCIENCE & ENGINEERING

By

Rohit Kushwaha(1805210044)

Dharmendra Kumar(1805210017)

Arbind Kumar Rao(1900520109003)

Under the supervision of

Prof. Manish Gaur

Mr. Ajeet Shukla



Department of Computer Science and Engineering

Institute of Engineering and Technology

Dr A.P.J Abdul Kalam Technical University, Lucknow, Uttar Pradesh

Contents

DECLARATION	4
CERTIFICATE	5
ACKNOWLEDGEMENT	6
ABSTRACT	7
LIST OF FIGURES.	8

Chapter 1: Introduction

1.1 Project Introduction	10
1.2 Document Structure.....	11

Chapter 2: Research

2.1 Literature Review	13
2.2 Communication Protocols.....	14

Chapter 3: Planning, Prototype and Technologies

3.1 Methodology.....	15
3.2 Use Cases and Scenarios	16
3.3 Django Architecture.....	17
3.3.1 Forms	17
3.3.2 Model.....	18
3.3.3 View.....	20
3.3.4 Templates.....	21
3.4 Technology	
3.4.1 Back End	23
3.4.2 Front End.....	25
3.4 Version Control.....	26

Chapter 4:Implementation

4.1 Database and Model.....	27
4.1.1 Users.....	28
4.1.2 Rooms.....	29
4.1.3 Chat.....	30
4.1.4 Messages.....	31
4.2 Webhooks.....	32
4.3 Setting up the development Environment.....	32
4.4 Authentication.....	33

Chapter 5: Evaluation

Experimental Results.....	34
Conclusions.....	34
Future Works.....	34

REFERENCES	35
-------------------------	-----------

Declaration

We hereby declare that this submission is our own work and that, to the best of our belief and knowledge, it contains no material previously published or written by another person or material which to a substantial error has been accepted for the award of any degree or diploma of university or another institute of higher learning, except where the acknowledgement has been made in the text. We have not submitted the project to any other institute for other degree requirements.

Submitted by:-

Date: 20 june 2022

(1) Name: Dharmendra Kumar

Roll.No.:1805210017

Branch: Computer Science & Engineering

Signature:

(2) Name: Rohit Kushwaha

Roll.No.:1805210044

Branch: Computer Science & Engineering

Signature:

(3) Name: Arbind Kumar Rao

Roll.No.:1900520109003

Branch: Computer Science & Engineering

Signature:

Certificate

This is to certify that the project report entitled "Django Based Chat Application" presented by Rohit Kushwaha, Dharmendra Kumar and Arbind Kumar Rao in the partial fulfillment for the award of Bachelor of Technology in Computer Science and Engineering, is a record of work carried out by them under my supervision and guidance at the Department of Computer Science and Engineering at Institute of Engineering and Technology, Lucknow. It is also certified that this project has not been submitted to any other institute for the award of any other degrees to the best of my knowledge.

Prof. Manish Gaur

Department of Computer Science and Engineering

Institute of Engineering & Technology, Lucknow

Mr. Ajeet Shukla

Department of Computer Science and Engineering

Institute of Engineering & Technology, Lucknow

Acknowledgement

We would like to express our gratitude to Dr. Divakar Singh Yadav, Head, CSE Department, IET Lucknow, and our humble supervisors Prof. Manish Gaur and Mr. Ajeet Shukla for their continued support and suggestions so far in this project. They have been very constructive, supportive, and kind and made it easier to write up this thesis work. Their continuous support gave us the motivation to work on this write-up and helped me to gain knowledge in various fields. I would also like to thank them for suggesting changes. They allowed me to locate improvement areas that helped me write this Interim write-up. We would like to express gratitude to all our classmates and teammates who motivated us and helped us in our time of need to complete this project. We shall be failing in duty if we do not acknowledge with thanks the authors of the references and other literature referred to in this thesis work.

Abstract

Chat refers to the process of communicating, interacting or exchanging messages over the internet . It involves two or more individuals that communicate through a chat enabled service or software. Chat may be delivered through text, audio or video communication via the internet.

A chat application has two basic components: server and client. A server is a computer program or devices. Clients who want to chat with each other connect to the server.

Our chat application is a peer to peer chat application in which a user will chat directly to the other user. It has the functionality of video chat rooms where multiple users can connect simultaneously through the video calls. This chat application is built on top of django and agora sdk.

List of Figures

Figure 1.1 Workflow Of MVT Structure.....	16
Figure 3.3.1: Class Definition of User	19
Figure 3.3.2 Class Definition of Message.....	19
Figure 3.3: View Function.....	20
Figure 3.4 Templates.....	23
Figure 4.1 Database UML Diagram.....	24
Figure 4.3: Python manage.py file.....	25
Figure 4.3 Python Settings.py file.....	26
Figure 4.4 Activity Diagram of Authentication.....	27
Figure 4.5 Homepage for Video Chat Rooms.....	30

Chapter 1

Introduction

1.1 Project Introduction

Our chat application is based on the Django Web framework and incorporates all of the functions a ordinary chat app has like Registration of consumer, login, logout, including friends and talking to a friend via the textual content or via video call.

Django is a high-degree Python net framework that encourages fast improvement and clean, pragmatic design. It looks after a lot of the trouble of net improvement, so we are able to be conscious of writing our app with no need to reinvent the wheel. It's a loose and open supply.

Django allows developers to keep away from many not unusual place protection errors via way of means of offering a framework that has been engineered to "do the proper matters" to guard the internet site automatically. For example, Django offers a stable manner to manipulate consumer accounts and passwords, fending off not unusual place errors like setting session records in cookies in which it is vulnerable (rather cookies simply incorporate a key, and the real records is saved withinside the database) or without delay storing passwords in preference to a password hash

Django is written in Python, which runs on many systems. That way we aren't tied to any specific server platform, and may run packages on many flavors of Linux, Windows, and macOS. Furthermore, Django is well-supported via means of many net web website hosting providers, who regularly offer particular infrastructure and documentation for hosting Django sites.

The project targets are

- To create a talk software by the use of Django net framework
- To add the capability of video chat room to the software

1.2 Document Structure

This file is prepared via means of Chapters, albeit now no longer explicitly indicated in a number of the sections. In the subsequent segment studies, we can describe the system of searching up complementary records for the venture in diverse sources. For example, which protocols to apply whilst managing actual time records and why.

Next, we can pass onto making plans and technology, on which we do our great to have the whole thing geared up for the improvement of the venture. Afterwards, in the implementation segment, we describe the maximum applicable bits of the improvement of the net software.

The implementation segment itself we additionally type the functions via means of chronological order, in preference to describing the front give up first and lower back give up later. At the give up of the day, it's miles what makes the maximum experience due to the fact we paintings in each on the identical time: we wrote the server characteristic element first and we made positive it turned into operating as anticipated via way of means of trying out it with our customer element operating.

Chapter 2 **Research**

2.1 Literature Review

Prior to getting commenced with the software improvement, we did a few studies at the modern messaging systems out there. We had been searching ahead to construct a completely unique experience, in preference to a precise identical to an current chat platform.

We already knew of the life of numerous messaging packages, and some chat packages that are acceptable builders. However, in no way earlier than had we executed an extensive evaluation in their gear to discover whether or not they had been truly sufficient for builders. Soon, we found out that not one of the web sites had been heading in our direction. Some of them had been lacking functions which we took into consideration vital and others had possibilities for similar enhancements. Contrary to what many human beings think, having some systems round isn't always a horrific thing.

We had been capable of get thoughts of what to construct and the way and decide which technology and techniques to apply primarily based totally on their experience. Often, this turned into as easy as checking their blogs.

Companies like Slack frequently submit improvement updates (inclusive of overall performance reviews, era comparisons, and scalability posts). Other times, we needed to dig into the net to discover the special alternatives we had and choose out the only which we taken into consideration to be the maximum appropriate.

During the aggressive evaluation, we investigated the subsequent systems:

- Flowdock - <https://www.flowdock.com>

- Gitter - <https://gitter.im>
- Hangouts - <https://hangouts.google.com>
- Matrix - <http://matrix.org>
- Messenger - <https://messenger.com>
- Rocket.chat - <https://rocket.chat>
- Skype - <https://web.skype.com>
- Slack - <https://slack.com>
- Telegram - <https://web.telegram.org>
- Whatsapp - <https://web.whatsapp.com>

2.2 Communication protocols

For messaging, there are some verbal exchange protocols to be had for the net. The most famous ones are AJAX, WebSockets, and WebRTC. AJAX is a gradual approach. Not handiest due to the headers that should be dispatched in each request, however additionally, and greater importantly, due to the fact there's no manner to get notified of the latest messages in a talk room.

By the use of AJAX, we should request/pull new messages from the server each few seconds, which could bring about new messages to take in to three seconds to seem at the screen, now no longer to say the severa redundant requests that this will generate.

WebSockets are a higher approach. WebSockets connections can take in to a few seconds to establish, however on the full-duplex verbal exchange channel, messages may be exchanged swiftly (averaging some milliseconds put off consistent with message). Also, each customer and server can get notified of latest requests via the identical verbal exchange channel, because of this that is not like AJAX, the customer no longer should ship the server a petition to retrieve new messages however as a substitute anticipate the server to ship them.

WebRTC is the brand new verbal exchange protocol to be had for the maximum cutting-edge browsers (Chrome, Firefox, and Opera). It is designed for high-overall performance, extremely good verbal exchange of video, audio, and arbitrary records. WebRTC no longer requires any server as a proxy to

alternate records, apart from the signaling server this is had to proportion the community and media metadata (regularly executed via WebSockets).

The reality that move records may be exchanged among customers without delay frequently method quicker messaging and much less server-aspect workload. WebRTC can run over TCP and UDP, however it frequently runs with UDP by default. Although UDP can result in packet loss it does supply a higher overall performance that may result in an extra fluid voice or video call, and we are able to come up with the money to lose some frames while video calling. For maximum net applications, communicate protocols aren't a topic of discussion. AJAX via HTTP is the manner to head due to the fact that it's far dependable and extensively supported. However, that is

Given the blessings and downsides of the 3 technologies, we determined to apply WebSockets for real-time messaging, which ensures us packet delivery (in contrast to frames on a video call, we do now no longer need to overlook out any textual content message), in addition to having accurate compatibility and being a famous and documented choice.

Chapter 3

Planning, Prototype and Technology

3.1 Project Methodology

Django is a web framework that creates dynamic projects. It consists in:

- Being able to respond to changes and new requirements quickly.
- Teamwork, even with the client.
- Building operating software over extensive documentation.
- Individuals and their interaction over tools.

We believed it was a perfect fit for our project since we did not know most requirements beforehand. By using Django, we were able to focus only on the features which had the most priority at the time.

3.2 Use Cases and Scenarios

User stories are one of the primary development artifacts when working with Django methodology. A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it.

Gathered from stakeholders (people, groups or organizations who are interested in the project), they show what we have to work on. Since we were working with Django, this list did not have to be complete before we started working on the project, but it was desirable to have at least a few items to start with so that we could establish proper feature priorities.

At the commencement of every sprint, we analyzed all user stories, estimated the value they added to the project and the amount of time they would take us doing each of them, and sorted them by descending order — placing the user stories which had the most added value and the least time cost at the top. The

value was quite subjective. We gave the highest priority to features which we believed were essential to the platform (such as instant text messages) or were very related to the chat's topic.

Nonetheless, in some cases, we had to make exceptions due to user stories dependencies. For example, sign in and sign up features had to be implemented first, since we needed user information to properly identify the room owner or the message sender.

3.3 Architecture of Django

Django is primarily based totally on MVT(Model-View-Template) architecture.MVT is a software program layout sample for growing an internet utility.

MVT shape has the subsequent 3 components -

Model: The version goes to behave because the interface of the records. It is answerable for keeping records. It is the logical records shape in the back of the complete utility and is represented via way of means of a database (usually relational databases together with MySQL, Postgres).

View: The View is the person interface — what you spot on your browser whilst you render a website. It is represented via way of means of HTML/CSS/Javascript and Jinja files.

Template: A template includes static components of the preferred HTML output in addition to a few unique syntax describing how dynamic content material might be inserted. d.

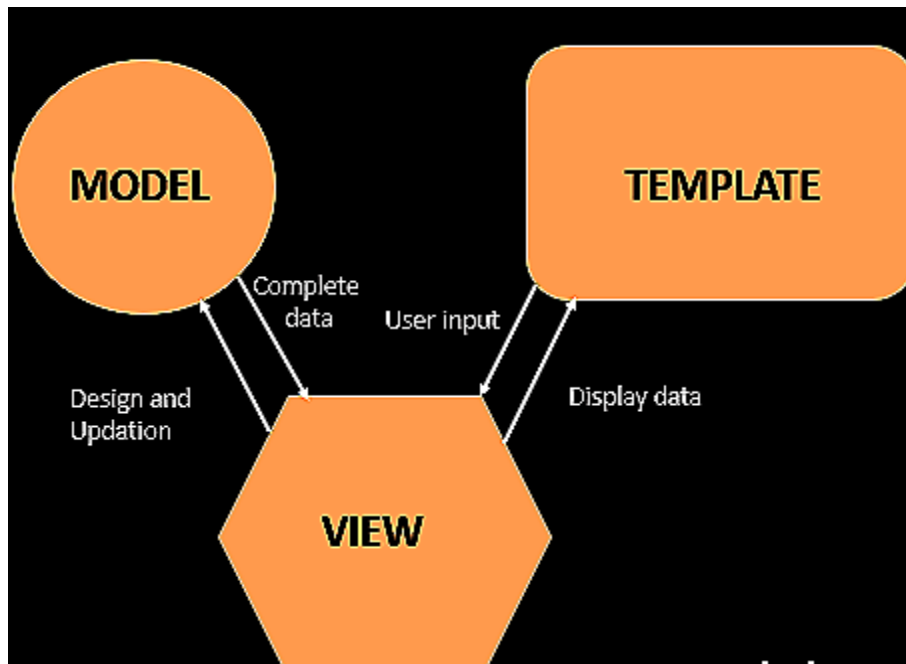


Figure 3.4.1 shows The workflow of MVT Architecture

3.3.1 Forms:

Django handles 3 wonderful components of the paintings concerned in paperwork:

making ready and restructuring records to make it prepared for rendering

growing HTML paperwork for the records

receiving and processing submitted paperwork and records from the client

In HTML, a shape is a set of factors inside ... that permit a tourist to do such things as input textual content, pick options, manage gadgets or controls, and so on, after which ship that statistics returned to the server. Some of those shape interface factors - textual content enter or checkboxes - are constructed into HTML itself. Others are a good deal extra complex; an interface that pops up a date picker or lets you to transport a slider or manage controls will commonly use JavaScript and CSS in addition to HTML shape factors to obtain those effects.

When rendering an item in Django, we usually get keep of it withinside the view (fetch it from the database, as an instance) by skip it to the template context increase it to HTML markup the use of template variables Rendering a shape in a template entails almost the identical paintings as rendering another form of item, however there are a few key differences.

In the case of a version example that contained no records, it'd hardly ever if ever be beneficial to do something with it in a template. On the other hand, it makes the best feeling to render an unpopulated shape - that's what we do while we need the person to populate it.

When we instantiate a shape, we are able to choose to depart it empty or pre-populate it, as an instance with:

records from a stored version example (as withinside the case of admin paperwork for editing)

records that we've collated from different sources

records obtained from a preceding HTML shape submission

The closing of those instances is the most interesting, due to the fact it's what makes it viable for customers now no longer simply to study a website, however to ship statistics returned to it too.

3.3.2 Models

- A model is the definitive supply of statistics. It consists of the important fields and behaviors of the records we're storing. Generally, every version maps to a single database table.
- Each version is a Python class that subclasses `django.db.models.Model`.
- Each characteristic of the model represents a database field.

```
from django.db import models
```

```
class Person(models.Model):
```

```
    first_name = models.CharField(max_length=30)
```

```
    last_name = models.CharField(max_length=30)
```

There are 3 Model classes are defined in our project-:

1. UserProfile Model
2. Message Model
3. Friends Model

```
from django.db import models

class UserProfile(models.Model):

    name = models.CharField(max_length=25)

    email = models.EmailField(unique=True)

    username = models.CharField(max_length=20, unique=True)
```

```

def __str__(self):

    return f"{self.name}"

class Messages(models.Model):

    description = models.TextField()

    sender_name = models.ForeignKey(UserProfile, on_delete=models.CASCADE,
related_name='sender')

    receiver_name = models.ForeignKey(UserProfile, on_delete=models.CASCADE,
related_name='receiver')

    time = models.TimeField(auto_now_add=True)

```

Figure 3.3.2 shows class definition of User and Message

3.3.3: Views:

A view characteristic, or view for short, is a Python characteristic that takes an internet request and returns an internet reaction. This reaction may be the HTML contents of an internet page, or a redirect, or a 404 error, or an XML document, or an image or anything, really. The view itself carries anything arbitrary common sense that is essential to go back to that reaction.

```

def chat(request, username):
    """
    Get the chat between two users.
    :param request:
    :param username:
    :return:
    """
    friend = UserProfile.objects.get(username=username)
    id = getUserId(request.user.username)
    curr_user = UserProfile.objects.get(id=id)
    messages = Messages.objects.filter(sender_name=id, receiver_name=friend.id) |
Messages.objects.filter(sender_name=friend.id, receiver_name=id)

    if request.method == "GET":
        friends = getFriendsList(id)
        return render(request, "chat/messages.html",
                      {'messages': messages,
                       'friends': friends,
                       'curr_user': curr_user, 'friend': friend})

```

Figure 3.3.3 shows view function

3.3.4: Templates:

A Django template is a textual content report or a Python string marked-up the use of the Django template language. Some constructs are diagnosed and interpreted with the aid of using the template engine. The essential ones are variables and tags.

A template is rendered with a context. Rendering replaces variables with their values, that are seemed up withinside the context, and executes tags. Everything else is output as is.

```

{% extends "chat/Base.html" %}

{% load static %}

{% block content %}

    <div style="margin-top: 30px">

        <h3>{{friend.name}}</h3>

    </div>

    <div class="messages" id="board">

        {% block message %}

        {% endblock %}

    </div>

    <div class="row">

        <form method="post" class="form-group" id="chat-box">

            {% csrf_token %}

            <div class="input-group mb-3">

                <div class="input-group-prepend">

                    <input type="text" placeholder="Send a message..."

name="message" id="msg_fiel

```

3.4 Technology

The architecture of the application consists of the back end and the front end, both of them having their own set dependencies (libraries and frameworks).

The front end is the presentation layer that the end user sees when they enter the site. The back end provides all the data and part of the logic and it is running behind the scenes.

3.4.1 Back End

The "back end" refers to the logic and data layers running on the server side. In our case, the back end makes sure that the data introduced through the client application (the front end), is valid. Since the front end can be avoided or easily manipulated (the source code is available to the end user) we have to make sure that all the requests we receive are first verified by the server: the requested URI is supported, the user has the appropriate permissions, the parameters are valid, etc.

If the request data is valid, we do often proceed to execute some logic accompanied by one or more database accesses

Socket.io: A JavaScript library which handles WebSocket connections. It abstracts most of the complexity behind WebSockets, and it also provides fallback methods which work without any special configuration.

Python; It is the main programming language that has been used however Django is built on top of python.

Data Storage: We have chosen sqlite3 database because

- They are more flexible and you can access nested data without having to perform any join.
- They are faster nested data is stored in the same place and can be consulted without any additional query.
- They scale better when distributing the data over different nodes.
- There are many types of sqlite3 databases which fit for different kinds of work, such as Key-Value for sessions or Document-based for complex data.

3.4.4 Front End

SPAs dynamically fetch data from the API as the user is browsing the site, avoiding to refresh the whole page whenever the user has filled in a form or navigated to another part of the site. The UX boost a SPA can get over a traditional website is very significant.

It is true that it often takes longer to load for the first time, due to having to download a bigger JavaScript file chunk, but once loaded the delay between operations is minimal which leads to a more fluid User eXperience, and less bandwidth use in Implementing a scalable Single-Page Application by using Vanilla JavaScript only would take an enormous amount of time, since it has none of the high-level utilities that make it simple to develop one of this kind, such as a high-level HTML renderer that allows you to build elements on the fly, storage or router. Hence, it made sense to choose an actively maintained and documented framework/library to start with.

React is a very powerful library with an enormous ecosystem (you can find many utilities that were meant to be used with React). It is featured due to its fast performance and small memory consumption, which is especially useful when targeting mobile devices. Moreover, there is a plethora of documentation on its official site and around the Internet.

Redux: An in-memory storage for JavaScript. It saves application states, which in other terms are the different data that our application uses over the time. A storage like Redux avoids having to transfer data up and down the React tree, since Redux stores it all in one place which can be accessed anytime. It is also modular,

Enzyme: a library which eases React components testing. This library was made specifically for React and enables us to simulate components like if they were rendered on the DOM. It is often used to test button handlers or elements which should appear only under certain conditions.

Sinon: A JavaScript testing library. Since it is not Node.js specific, we are also using it on our front end side to make testing easier.

3.4 Version Control

A version control system can be useful to developers, even when working alone.

It enables us to go back in time to figure out what broke a certain utility, work on different features at the time and revert/merge them into the original source code with no difficulty, watch how the project evolved over the time, and so on.

We chose Git. Not only because it is the most popular and widely used version control system, but also because part of our project was the integration with GitHub, and GitHub works with Git. For the same reason as above, we chose GitHub to be our remote source code repository.

Currently, it is a public space where developers can come and have a look at the source code that is powering the chat application, report bugs they encounter or even contribute by submitting pull requests.

Chapter 4

Implementation

This chapter details the most relevant parts of the application development, decisions taken and algorithms. We have divided this chapter into three sections: databases (design), features (the most important ones) and a brief overview on how we tested our features.

4.1 Database and Model

A key defining aspect of any database-dependent application is its database structure. The database design can vary depending on many different factors, such as the number of reads over writes or the values that the user is likely to request the most. That is because as full stack developers we want the database to have the best performance, which can often be achieved by focusing the optimizations on the most common actions.

Our final database design ended up having four different collections: users, rooms, chats, and messages.

A schema constrains the contents of a collection to a known format, saving us from validating the structure of the data before or after it has been put in into the database.

4.1.1 Users

To start, we needed somewhere to store our users. Since we were expecting a significant number of entries, an individual collection for the users' themselves was the most appropriate. What we mean by that is that it was best for the users' collection to solely store the information that made reference to their authentication and personal data. Their rooms, chats, and messages should be stored somewhere else. Given that we were

expecting a lot of rooms, chats, and messages per user, we refrained from even making references to them in this collection. We are querying these other collections directly.

Schema Fields-

- `_id`: identifier.
- `username`: friendly identifier.
- `email`: email address.
- `password`: encrypted password.
- `profile`: personal details

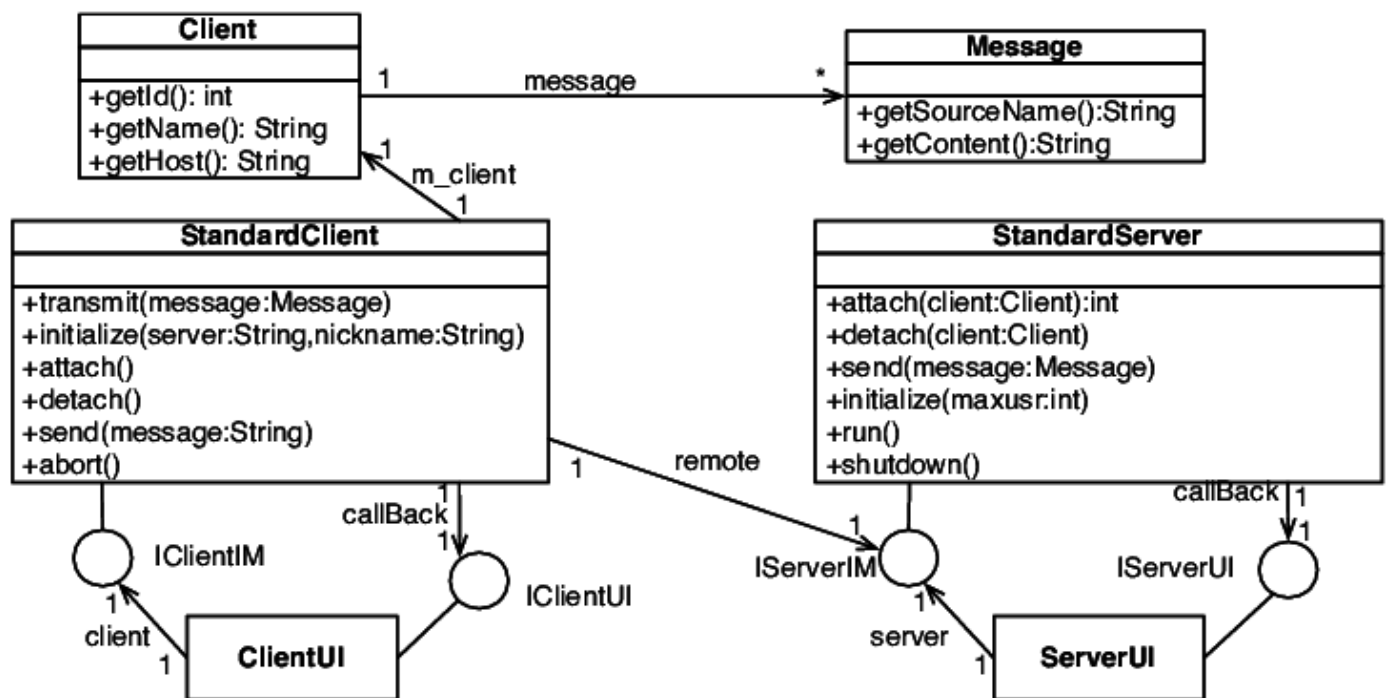


Figure 4.1 Database UML Diagram

4.1.2 Rooms

Given that we were not going to store rooms as nested data inside the users' collection, mainly because we were looking forward to referring to them directly, we created an independent collection for them. In addition, we were expecting many rooms, probably even more than users. Hence it was not a not a good idea to nest them under any other document.

Schema fields:

- `_id`: identifier.
- `title`.
- `slug`: room URL identifier.
- `description`.
- `owner`: `_id` of the owner user.
- `isPrivate`: whether the room is private or public.
- `members`: array of user `_id` who are members of that room.
- `updatedAt`: modification date.
- `createdAt`: creation date.

4.1.3 Chats

As we stated earlier, our chats were going to be in individual collections. There might be rooms in which their members have few chats, but others might have hundreds (even if that leads to having a few inactive ones). Once again, we had to think whether it was worth embedding or referring messages inside the Chats collection or keeping them isolated in another one. In this case, it was evident. We were expecting thousands of messages in any Chat, which would rapidly go over the 16MB that any sqlite document can hold

even if only storing references. Thus, messages had to be saved in a different collection.

Schema fields: • `_id`: identifier.

- `room`: identifier of the room it belongs to.

- `title`. • `description`.

- `firstMessageAt`: date of the first message sent. It is used to determine whether the user has already retrieved all messages of a chat.

- `lastMessageAt`: date of the last message sent.

- `updatedAt`: modification date.

- `createdAt`: creation date.

4.1.4 Messages

We can summarize our Messages needs as follow:

1. Ability to store hundreds of messages per hour.
2. Ability to retrieve thousands of messages per hour.
3. Ability to retrieve messages in chronological order (most recent first).

We have indexed messages by `_id` and `chat + createdAt`. The first one helps when looking for a specific message, whereas the second composed index works out well when looking for past messages. We have composed the date with the chat to filter only the messages which belong to a particular chat since we will never be interested in mixed chat messages.

4.2 Webhooks

During the implementation of the application, we got to the point when we needed to store data from GitHub WebHooks, which we will go through into why we needed them. GitHub WebHooks only sends the information once per repository, so we had to make sure that the data was stored in a way that any chat linked to that GitHub repository could access GitHub updates.

The information had to be able to be fetched and stored quickly, since GitHub updates tend to be numerous and we wanted to notify the Chat users in real time. This case resembles the Messages one, though this time we would be dealing with messages sent by bots.

Schema fields:

- type: webhook type, in case there was more than one provider.

- uid: unique identifier of the webhook message (sent by the saving/updating).
- github: GitHub specific fields, such as repository name or action.
- updatedAt: modification date.
- createdAt: creation date.

We have indexed webhooks by `_id` and `_id + github.repository`, which is similar to what we did with messages. Nonetheless, we would probably have to create more indexes if we had more providers, since `_id + github.repository` only suits GitHub ones.

4.3 Setting Up development Environment

First of all whilst we create any project in django a few not unusual place document get created automatically they're as follows —

manage.py- This document is used to engage with the project through the command line(begin the server, sync the database... etc). Here is the screenshot connected to the manage.py document in our project.

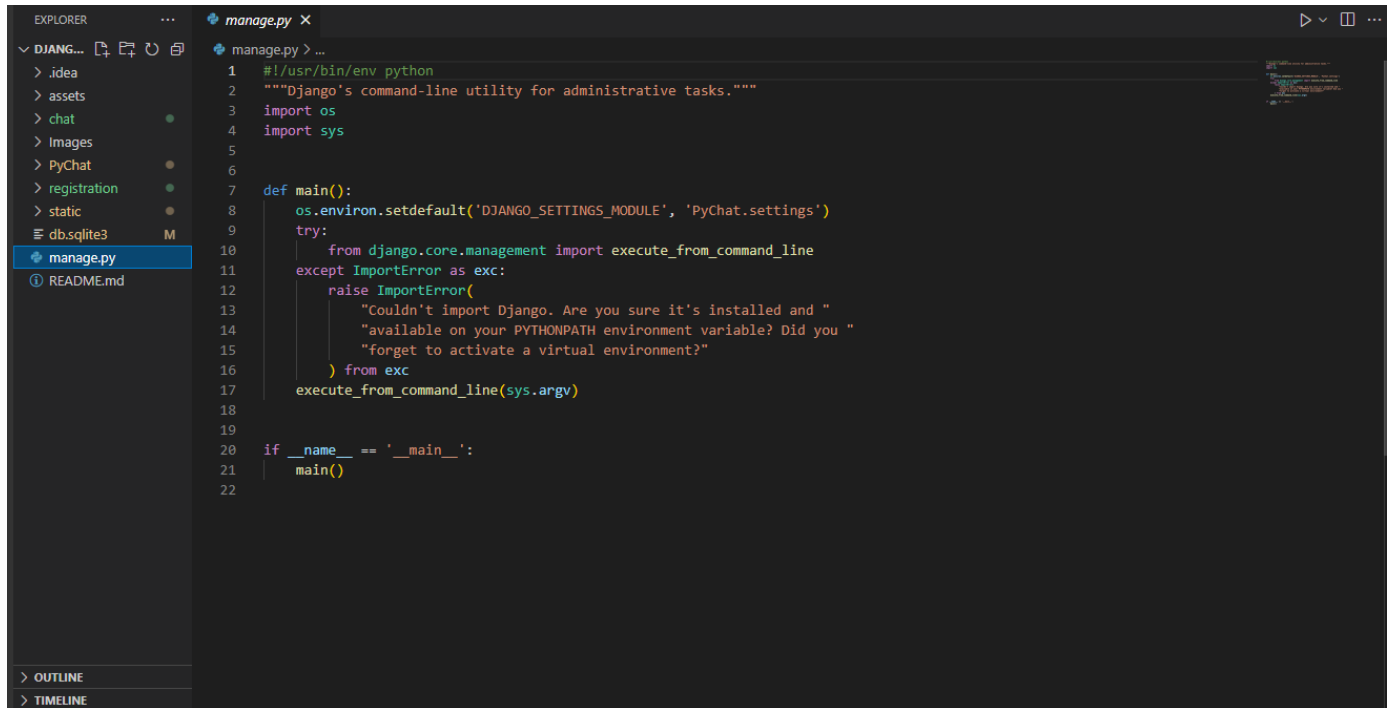


Figure 4.3 shows Manage.py

`__init__.py` –It is a python bundle. It is invoked whilst the bundle or a module withinside the bundle is imported. We commonly use this to execute bundle initialization code, for instance for the initialization of package-level data.

`settings.py` – As the call suggests it consists of all of the internet site settings. In this document, we register any packages we create, the region of our static files, database configuration details, etc.

```
1 """
2 Django settings for PyChat project.
3
4 Generated by 'django-admin startproject' using Django 3.0.8.
5
6 For more information on this file, see
7 https://docs.djangoproject.com/en/3.0/topics/settings/
8
9 For the full list of settings and their values, see
10 https://docs.djangoproject.com/en/3.0/ref/settings/
11 """
12
13 import os
14
15 # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
16 BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
17
18
19 # Quick-start development settings - unsuitable for production
20 # See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/
21
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = 'usnm_pu9ng2xf4@q#aens^n4t_1)b6icnc^1u-f_jko++*=(#'
24
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = ['127.0.0.1', '192.168.43.177']
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
```

Figure 4.3.2 shows Setting.py

urls.py – In this document, we store all hyperlinks of the project and capabilities to call.

wsgi.py – This document is utilized in deploying the project in WSGI. It is used to assist Django packages communicate with the webserver.

In Django, the version does the linking to the database and every version receives mapped to a single desk withinside the database. These fields and strategies are declared under the document models.py

With this linking to the database, we will truly file each and every file or row from that unique desk and might carry out the DML operations at the desk.

Django.db.models.The version is the subclass that is used here. We can use the import declaration through defining as from django.db import models.

So after defining our database tables, columns and records; we're going to get the statistics related to our utility through defining the mapping in settings.py document beneath the INSTALLED_APP.

Django receives person requests through the URL locator and responds to it. django.urls module is utilized by it to manipulate the URL's requests.

4.4: Authentication:

The Django authentication system handles each authentication and authorization. Briefly, authentication verifies a person is who they declare to be, and authorization determines what an authenticated person is permitted to do. Here the time period authentication is used to consult each task.

The auth system is composed of:

Users Permissions: Binary (yes/no) flags designating whether or not a person may also carry out a positive task.

Groups: A common manner of making use of labels and permissions to multiple people.

A configurable password hashing system Forms and thinks about equipment for logging in users, or restricting content.

The authentication system in Django aims to be very common and doesn't offer a few functions generally determined in internet authentication systems. Solutions for a number of those not unusualplace issues were applied in third-celebration packages:

- Password strength checking
- Throttling of login attempts

Authentication against third-parties (OAuth, for example)

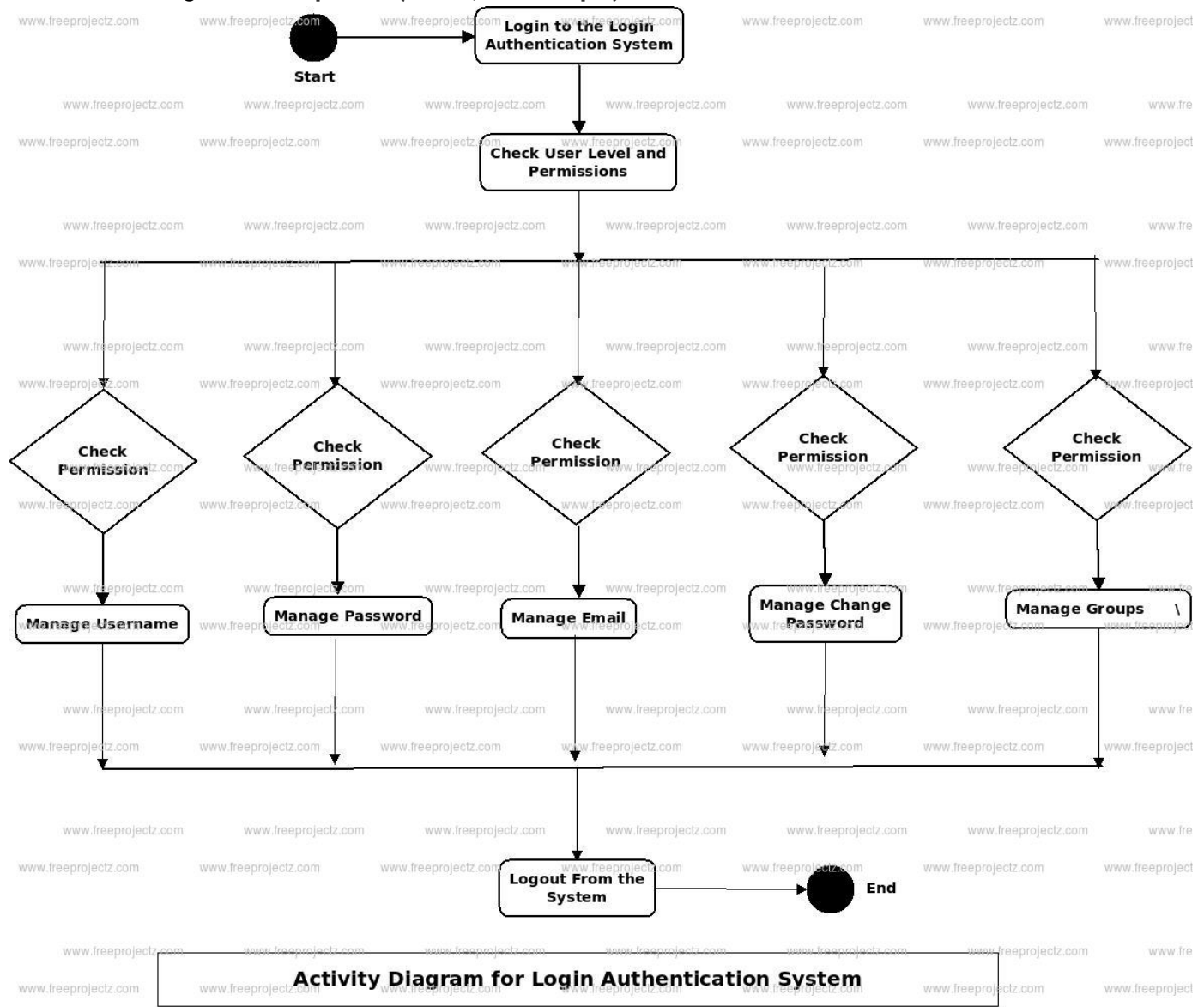



Figure 4.4 Shows Activity diagram for authentication

Chapter 5

Evaluation

The application is built on the Django framework running successfully. All the features of the website are working properly.



Django Based Chat Application

Created As Final Year Project 2022

Created By:

Dharmendra Kumar
Arvind Kumar Rao
Rohit Kushwaha

Room:

Name:

Join Stream

Figure 4.1 show homepage for video chat room

5. 1 Conclusions

In this project we have learned the development process of an application specially built on the django web framework.

5. 2 Future Works

Although the application itself works well, much was learned during its development. For this reason, we wrote a list of possible improvements/changes, some of which are easy to execute, others might require rewriting a significant amount of the current source code. Apart from that, the Ideal application (which we described in section 2.2) was too ambitious, which resulted in many features not being able to be implemented during the course of the project.

While our application already provides the basics to programmers who want to talk about themselves, having more of these model features done would probably attract the attention of more of them.

References

- [1] GreenField, Daniel & Roy Audrey. (2013, September). Two scoops of Django, 2013. ICIP 2005. IEEE International Conference on (Vol. 2, pp. II-370). IEEE.
- [2] Jeff Forcier, Paul Bisex, & Wesley Chun. (2008). Python Web development with Django. COnline: On-Line Compendium of Computer Vision, 9.
- [3] Django Official Documentation (2022)
- [4] Arun Ravindran(2013). "Django Design Pattern And Best Practices. 16 (5): 555–559. doi:10.1016/S0893-6080(03)00115-1. Retrieved 17 November 2013.
- [5] Beun Curtin. "Django Cookbook Web Development with Django". Retrieved 16 November 2013
- [6] Dauzen S, “Getting Started with Django, 2012
- [12] “Scaling secret: Real-time chat.” <https://medium.com/always-be-coding/scaling-secret-real-time-chat-d8589f8f0c9b#.m5jjgxq6x>, may 2015. Accessed: 2016-10-18.
- [13] “Analysis of json use cases.” https://blogs.oracle.com/xmlorb/entry/analysis_of_json_use_cases, apr 2013. Accessed: 2016-11-08.
- [14] “Crud cycle (create, read, update and delete cycle).” <http://searchdatamanagement.techtarget.com/definition/CRUD-cycle>. Accessed: 2016-11-08.
- [15] “About native xmlhttp.” [https://msdn.microsoft.com/en-us/library/ms537505\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms537505(v=vs.85).aspx). Accessed: 2016-12-18.
- [16] “Please. don’t patch like an idiot..” <http://williamdurand.fr/2014/02/14/please-do-not-patch-like-an-idiot/>, aug 2016. Accessed: 2016-12-18.
- [17] “Rfc 5789 - patch method for http.” <https://tools.ietf.org/html/rfc5789>, mar 2010. Accessed: 2016-12-18.
- [18] “Performance tips | google cloud platform.” https://cloud.google.com/storage/docs/json_api/v1/how-tos/performance#patch. Accessed: 2016- 12-18.
- [19] “container vs component?.” <https://github.com/reactjs/redux/issues/756#issuecomment-141683834>, sep 2015. Accessed: 2016-11-13.