# MINI PROJECT (2020-2021)

**RT Messenger**

**MID-TERM REPORT**



**Institute of Engineering & Technology**

**Submitted by-**

**Prashant Asthana**

**(181500487)**

**Tanishq Tripathi**

**(181500751)**

**Shriyanshi Baranwal**

**(181500691)**

***Supervised By: -***

**Mr. Amir Khan**

Technical Trainer

# Department of Computer Engineering & Applications

**Contents**

|  |  |  |
| --- | --- | --- |
| **Abstract** | **3** |  |
| **1. Introduction** | **4** |  |
| 1.1 General Introduction to the topic | **4** |  |
| 1.2 Technical Details | **4** |  |
| 1.3 Hardware and Software Requirements | **5** |  |
| **2. Objectives** | **6** |  |
| **3. Implementation Details** | **7** |  |
|  |  |  |
| **4. Some Screenshots** | **8** |  |

**Abstract**

Lately we see a constant switch towards real-time applications. With the wide support for WebSockets in recent browsers, more and more frameworks are giving us the ability to use them. Django, which was primarily a request/response-based framework, is doing the switch with Django Channels. Django channels is a lot more than just support for WebSockets, it's a complete architectural change for Django and - in my honest opinion - a great move towards the new era of frameworks.

The real-time chat application **does not** save your chat messages in the database before passing it to another client.That is because it uses sockets to send and receive messages between machines. The messages are exchanged through a network, either using a Transmission Control Protocol (“TCP”) or a User Datagram Protocol (“UDP”). You could save the messages if you want, but it would be separated from the socket send-receive process.

# Introduction

* 1. **General Introduction to the topic**

Nowadays, when all sorts of chat rooms have become extremely popular, when every second large company has launched or developed its own instant messenger, when an increase of smiles and change in the text size is considered as innovation, in the era of iMessages, Slack, Hipchat, Messenger, Google Allo, Zulip, etc. We will use Django-channels.

Python is fast becoming a popular coding language in the world, and there are many popular frameworks that build on Python. One of them is Django and it has many functionalities and supporting libraries. For this article, we like to explore one interesting method that builds on Django to handle not only HTTP but also long running connections such as WebSockets, MQTT, chatbots, etc.

* 1. **Technical Details**

Basic example of a multi-room chatroom, with messages from all rooms a user is in multiplexed over a single WebSocket connection. There is no chat persistence; you only see messages sent to a room while you are in that room.

Uses the Django auth system to provide user accounts; users are only able to use the chat once logged in, and this provides their username details for the chatroom.

This package allows our application to interact with a user not only using HTTP 1.1 (request-response), but also using HTTP/2 and WebSocket.

WebSocket is designed for exchanging messages between the client and the web server in real time. You should consider it as an open channel between the client and the server, with the ability to subscribe to the events sent to it.

* 1. **Hardware and Software Requirements**

**Software Specification:**

• Technology Implemented: JavaScript, Django, python, WebSocket

• User Interface Design : Web based Application

• Web Browser: Chrome

**Hardware Requirement:**

• Processor: Intel CORE i3

• Operating System: Windows 10

• RAM:4 GB

• Hardware System: Computer System

• Hard Disk: 64 GB

**Objectives**

The main objective of creating a real-time messenger is:

Firstly, it saves storage. Real-time messaging doesn’t save your messages, as the latter are just passing through the network, making your whole chat architecture storage friendly.

Secondly, as its name shows, it is “real-time” and we can see that this type of messaging is faster than the normal one. The traditional messaging system saves your messages in the database, so every time you need your messages, you have to query to the database to get them. The real-time messaging only passes messages every time we want to send and receive them when they arrive on the socket, so it should be faster and more efficient.

Django is a web building framework based on the Python programming language. It uses a Model-View-Template (“MVT”) architecture. Django is being popular because it is fast, secure, and a scalable high-level Python programming for web building. Learning Django is not that easy, but when you get used to it, it has many great functionalities.

**Implementation Details**

1. Install and configure the environment. Create a virtual environment and Django project on Python as well as prepare them for work. It is important to add channels to the INSTALLED\_APPS setting.
2. Install Redis or another channel layer.  Configure Django-channels so that it will use the Redis as the channel storage by adding asgi\_redis.RedisChannelLayer to the CHANNEL\_LAYERS setting.
3. Add a basic message handler to multi-chat/routing.py in order to initialize the Django channels routing.
4. Check how everything works by installing any WebSocket client and running the Django web server together with the interactive session interpreter.
5. Connect to the web socket and try to send a chat message. If the message is duplicated in the server log, it means that the chat works correctly.
6. Create authorization page for the chat by creating a template and adding a stylesheet as well as redirection URLs for login and log out.
7. Create chat rooms by adding a ‘Room’ class to chat/models.py.
8. Add event handlers to manage events such as connecting to and disconnecting from the web socket, message processing, login/logout, and message sending.
9. Create a front-end part of your chat by writing Python code for connecting to the socket, connecting to and disconnecting from the rooms, and exchanging messages.

**Some Screenshots**

































