RealTime Messenger

A MINOR PROJECT-II
Submitted in Partial Fulfillment of the Requirement for the Award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING-DATA SCIENCE
SUBMITTED TO

Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)

SUBMITTED BY
Anuj Raj (0176CD211026)
Atharv Jain (0176CD211031)
Vivek Kumar (0176CD211136)

UNDER THE SUPERVISION OF
Dr Saket Jain
Department of Computer Science & Engineering-Data Science

Department of Computer Science & Engineering-Data Science

Lakshmi Narain College of Technology Excellence, Bhopal (M.P.)

June 2024

Lakshmi Narain College of Technology Excellence, Bhopal

Department of Computer Science & Engineering-Data Science

CANDIDATE'S DECLARATION

We, Student **Anuj Raj, Atharv Jain and Vivek Kumar ,**Student of **Bachelor of Technology, Computer Science & Engineering-Data Science, Lakshmi Narain College of Technology Excellence, Bhopal** session 2023-24 hereby declare that the work presented in the Minor Project-II entitled "**RealTime Messenger"** is outcome of my/our own bonafide work, which is correct to the best of our knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any previous work and has not been submitted to any University for the award of any degree.

We also declare that **"A check for plagiarism has been carried out on the Minor Project-II and is found within the acceptable limit and report of which is enclosed herewith".**

Anuj Raj (0176CD211026)
Atharv jain (0176CD211031)
Vivek Kumar (0176CD211136)

Lakshmi Narain College of Technology Excellence, Bhopal

Department of Computer Science & Engineering-Data Science

CERTIFICATE

This is to certify that the work embodies in this Minor Project-II entitled **"RealTime Messenger"** being submitted by **"Anuj Raj (0176CD211026)" , "Atharv Jain (0176CD211031)" and "Vivek Kumar (0176CD211136)"** for partial fulfillment of the requirement for the award of degree of "**Bachelor of Technology** in **Computer Science & Engineering-Data Science"** discipline to "**RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL (M.P.)**" during the academic year 2023-24 is a record of real piece of work, carried out by them under my supervision and guidance in the **"Department of Computer Science & Engineering-Data Science", Lakshmi Narain College of Technology Excellence, Bhopal (M.P.)**

SUPERVISED BY

Dr Saket Jain
Department of CSE-Data Science
LNCTE, Bhopal

FORWARDED BY

Prof. (Dr.) Neetesh Kumar Gupta
(Head, Department of CSE-Data Science)
LNCTE, Bhopal

APPROVED BY

Dr. Anil Kumar Saxena
Principal
LNCTE, Bhopal

ACKNOWLEDGEMENT

At the outset, we would like to link to thank the Almighty who made all the things possible. Writing this project report would not have been possible without the support of several people whom we need to whole heartedly thank. we express a deep sense of gratitude to our Supervisor **"Dr Saket Jain", Dept. of CSE-Data Science** for the valuable and inspirational guidance from the initial to the final level that enabled me to develop an understanding of this Project work.

we would like to give our sincere thanks to **Prof.(Dr.) Neetesh Kumar Gupta, Head, Dept. of CSE-Data Scienc**e for their kind help, encouragement and co-operation throughout my/our Project period and We owe our special thanks to Principal **Dr. Anil Kumar Saxena** for their guidance and suggestions during the Project work.

Lastly, We want to thank our parents, friends and to all those people who had contributed to my/our project directly or indirectly for their moral and psychological support.

Anuj Raj (0176CD211026)
Atharv Jain (0176CD211031)
Vivek Kumar (0176CD211136)

CONTENT

Page No.

Abstract 1

Chapter 1 Introduction..... 2-4
1.1 Motivation.....
1.2 Scope.....
1.3 Objective.....
1.4 Application.....

Chapter 2 Literature Survey..... 5-7
2.1 Literature Survey.....
2.2 Conclusion.....

Chapter 3 Problem Statement..... 8-10
3.1 Problem Statement

Chapter 4 Minimum Hardware and Software Requirements..... 11-13
4.1 Software Requirement.....
4.2 Hardware Requirement.....

Chapter 5 Methodology Used..... 14-16
5.1 Method.....

Chapter 6 Design Framework..... 17-21
6.1 ER Diagram.....
6.2 Use Case Diagram.....
6.3 Data Flow Diagram.....

Chapter 7 Implementation..... 22-36
7.1 Snapshot.....
7.2 Coding .....

Chapter 8 Testing..... 37-39
8.1 Testing .....

Chapter 9 Conclusion & Future Scope 40-42

Conclusion & Future Scope

References 43

ABSTRACT

The Realtime Messenger project aims to develop a robust messaging application using Python and various frameworks and technologies. This project leverages PyFlet for the frontend interface, SQLITE3 for efficient data storage, and web sockets for real-time communication among multiple users. The primary motivation behind this project is to create a user-friendly messaging platform that allows seamless exchange of messages in real-time. The scope of the project encompasses the development of key features such as user authentication, message encryption, and multi-user chat functionality. The objective is to implement a reliable and scalable messaging solution that can cater to the needs of both individual users and organizations. This project holds significant relevance in today's digital age where instant communication is paramount. In this report, we discuss the methodology, design framework, implementation details, testing procedures, and conclude with insights into future enhancements and applications of the Realtime Messenger.

CHAPTER 1

INTRODUCTION

The introduction serves as a gateway to understanding the Realtime Messenger project, offering an extensive overview of its objectives, scope, and significance. In today's fast-paced digital era, the demand for efficient and real-time communication solutions has surged dramatically. Messaging applications have become indispensable tools for personal and professional communication, facilitating instant exchange of messages across various platforms and devices. However, existing messaging platforms often exhibit limitations in terms of scalability, real-time communication capabilities, and data security. Recognizing these challenges, the Realtime Messenger project seeks to develop a robust and versatile messaging platform that addresses these shortcomings while offering an intuitive user experience.

1.1 Motivation

The motivation behind the Realtime Messenger project stems from the need for a comprehensive messaging solution that caters to the evolving needs of modern users. The proliferation of mobile devices, coupled with the increasing reliance on digital communication, underscores the importance of developing innovative messaging applications that deliver seamless and efficient communication experiences. By embarking on this project, we aim to leverage emerging technologies and best practices to create a messaging platform that meets the demands of individual users, businesses, and organizations alike.

1.2 Scope

The scope of the Realtime Messenger project encompasses the development of key features and functionalities essential for a fully-fledged messaging application. These include user authentication mechanisms to ensure secure access to the platform, real-time messaging capabilities to enable instant communication among users, and robust data encryption techniques to safeguard sensitive information. Additionally, the project aims to explore avenues for scalability and extensibility, allowing for future enhancements and integrations with emerging technologies.

1.3 Objective

The objectives of the Realtime Messenger project are multifaceted, reflecting the diverse needs and aspirations of stakeholders involved. Firstly, the project aims to design and implement a user-friendly interface that facilitates intuitive navigation and seamless interaction with the messaging platform. Secondly, it seeks to develop robust backend infrastructure capable of handling concurrent user connections, message exchanges, and data storage efficiently. Thirdly, the project endeavors to ensure the security and privacy of user data through encryption techniques and stringent access controls. Lastly, the project aims to deliver a scalable and extensible solution that can accommodate the evolving needs of users and adapt to technological advancements in the future.

1.4 Application

In today's fast-paced digital era, the demand for efficient and real-time communication solutions has surged dramatically. Messaging applications have become indispensable tools for personal and professional communication, facilitating instant exchange of messages across various platforms and devices. However, existing messaging platforms often exhibit limitations in terms of scalability, real-time communication capabilities, and data security. Recognizing these challenges, the Realtime Messenger project seeks to develop a robust and versatile messaging platform that addresses these shortcomings while offering an intuitive user experience.

CHAPTER 2

LITERATURE SURVEY

The literature survey chapter delves into the existing research, developments, and trends in the realm of real-time messaging systems and related technologies. It serves as a comprehensive exploration of the academic and industry landscape surrounding messaging applications, providing valuable insights and perspectives that inform the design and implementation of the Realtime Messenger project.

2.1 Literature Survey

The literature survey chapter delves into the existing research, developments, and trends in the realm of real-time messaging systems and related technologies. It serves as a comprehensive exploration of the academic and industry landscape surrounding messaging applications, providing valuable insights and perspectives that inform the design and implementation of the Realtime Messenger project.

The evolution of messaging applications is a central theme in the literature survey, tracing the historical development of messaging technologies from traditional SMS to modern instant messaging platforms. The chapter examines the key milestones and innovations that have shaped the evolution of messaging applications, highlighting the transition from text-based messaging to multimedia-rich communication experiences. Additionally, it explores the emergence of real-time communication protocols and frameworks, such as web sockets, MQTT, and XMPP, which have revolutionized the way messages are exchanged and delivered in real-time.

One of the key focal points of the literature survey is the analysis of existing research and projects that have addressed similar challenges or implemented innovative solutions in the field of real-time messaging systems. By reviewing academic papers, technical articles, and case studies, the chapter synthesizes insights into the state-of-the-art approaches and best practices in designing and implementing messaging applications. Furthermore, it examines the implications of emerging technologies such as AI, machine learning, and blockchain on the

future of messaging platforms, envisioning new possibilities for enhancing user experiences and ensuring data security.

The literature survey also delves into user behavior and preferences in the context of messaging applications, exploring factors that influence user adoption, engagement, and retention. By analyzing user studies, surveys, and market research reports, the chapter identifies key trends and patterns in user interaction with messaging platforms, shedding light on user expectations, pain points, and preferences. This understanding of user behavior informs the design and development of the Realtime Messenger project, guiding decisions related to user interface design, feature prioritization, and user experience optimization.

2.2 Conclusion

In conclusion, the literature survey chapter provides a comprehensive overview of the existing body of knowledge and research in the field of real-time messaging systems. By synthesizing insights from academic literature, industry trends, and user studies, the chapter offers valuable perspectives and guidance for the design and implementation of the Realtime Messenger project. It highlights the importance of leveraging emerging technologies, understanding user behavior, and adopting best practices to create a messaging platform that meets the needs and expectations of modern users.

CHAPTER 3

PROBLEM STATEMENT

The problem statement chapter articulates the key challenges and issues that the Realtime Messenger project aims to address. It serves as a foundational component of the project report, providing a clear and concise delineation of the problems that necessitate the development of the messaging application.

In today's digital age, the proliferation of messaging applications has transformed the way individuals and organizations communicate and collaborate. However, existing messaging platforms often exhibit limitations and shortcomings that hinder their effectiveness and usability. One of the primary challenges is scalability, as many messaging applications struggle to handle large volumes of concurrent users and messages efficiently. This leads to performance issues such as latency, message delivery delays, and service outages, which can undermine the user experience and reliability of the platform.

Another key challenge is the lack of real-time communication capabilities in traditional messaging applications. While most platforms support asynchronous messaging, enabling users to send and receive messages at their convenience, few offer true real-time communication features that facilitate instant message delivery and response. This poses a barrier to effective and efficient communication, particularly in scenarios where timely responses are critical, such as team collaboration, customer support, and emergency response.

Data security and privacy are also significant concerns in the realm of messaging applications, as users increasingly share sensitive information and personal data through these platforms. The centralized nature of many messaging platforms exposes user data to risks such as unauthorized access, data breaches, and surveillance, raising concerns about confidentiality, integrity, and trust. Furthermore, the lack of end-to-end encryption and robust authentication mechanisms leaves user data vulnerable to interception and manipulation by malicious actors.

The Realtime Messenger project aims to tackle these challenges by developing a robust and versatile messaging platform that offers scalability, real-time communication capabilities, and

robust security features. By leveraging emerging technologies and best practices, the project seeks to create a messaging application that delivers a seamless and secure communication experience for users across various domains and use cases. Through rigorous analysis and evaluation, the project aims to identify innovative solutions and design principles that address the underlying problems and shortcomings of existing messaging platforms.

CHAPTER 4

MINIMUM HARDWARE AND SOFTWARE REQUIREMENTS

The minimum hardware and software requirements chapter outlines the necessary resources and configurations for deploying and running the Realtime Messenger application effectively. It serves as a practical guide for users and administrators who intend to install and operate the messaging platform in various environments.

4.1 Software Requirements

Software requirements are fundamental to ensuring the compatibility and functionality of the Realtime Messenger application across different operating systems and platforms. The primary software requirement is the Python programming language, which serves as the foundation for developing the backend logic and server-side components of the application. Additionally, the PyFlet framework is essential for frontend development, providing tools and utilities for designing and implementing the user interface of the messaging platform. The SQLITE3 database management system is utilized for efficient data storage and retrieval, enabling the management of user accounts, messages, and other application data. Furthermore, the web socket library is indispensable for establishing real-time communication channels between the messaging server and client applications, facilitating instant message delivery and updates.

In terms of operating system compatibility, the Realtime Messenger application is designed to run on various platforms, including Windows, macOS, and Linux distributions. This ensures flexibility and accessibility for users across different computing environments, allowing them to access the messaging platform seamlessly regardless of their preferred operating system.

4.2 Hardware Requirements

Hardware requirements play a crucial role in determining the performance and scalability of the Realtime Messenger application, particularly in scenarios with high user concurrency and message traffic. The CPU (Central Processing Unit) is a key hardware component responsible for executing computational tasks and processing user requests efficiently. A multicore processor with adequate clock speed and cache memory is recommended to handle concurrent user connections and message exchanges effectively. Additionally, sufficient RAM (Random Access Memory) is essential for storing temporary data and facilitating fast data access, ensuring smooth operation of the messaging platform even under heavy load conditions.

Storage capacity is another critical consideration, as the Realtime Messenger application generates and stores large volumes of message data over time. A solid-state drive (SSD) or hard disk drive (HDD) with ample storage space is recommended to accommodate the growing data storage requirements of the messaging platform.

By providing clear guidelines on the hardware and software requirements, this chapter enables users and administrators to prepare their computing environments for deploying and running the Realtime Messenger application effectively. It ensures compatibility, performance, and reliability, laying the foundation for a seamless messaging experience for users across diverse platforms and devices.

CHAPTER 5

METHODOLOGY USED

The methodology used chapter elucidates the approach and techniques employed in the development of the Realtime Messenger project, offering insights into the project lifecycle and development process. It serves as a roadmap for understanding the systematic steps and methodologies adopted in realizing the objectives of the project.

5.1 Methods

The development methodology employed for the Realtime Messenger project is agile, emphasizing iterative and incremental development cycles to deliver tangible results quickly and adapt to changing requirements and priorities. The agile approach enables flexibility, collaboration, and responsiveness to feedback, allowing for continuous improvement and refinement of the messaging platform throughout the development process. Scrum, a popular agile framework, is utilized to organize and manage development tasks, with regular sprint cycles ensuring regular progress updates and stakeholder engagement.

The project lifecycle consists of several key phases, including requirements gathering, design, implementation, testing, and deployment. Each phase is characterized by specific activities, deliverables, and milestones, contributing to the overall success of the project. Requirements gathering involves eliciting and documenting user needs, preferences, and expectations through stakeholder interviews, surveys, and feedback sessions. The design phase focuses on translating the gathered requirements into a comprehensive system architecture and design specifications, including user interface mockups, database schemas, and system workflows.

Implementation entails the actual coding and development of the Realtime Messenger application, following the design specifications and best practices. The development process is collaborative, with team members working together to implement various features and functionalities of the messaging platform. Continuous integration and version control practices are adopted to ensure code quality, consistency, and reliability throughout the development lifecycle.

Testing is an integral part of the development process, ensuring the functionality, reliability, and performance of the Realtime Messenger application. Various testing methodologies, including unit testing, integration testing, and system testing, are employed to validate different aspects of the application, such as user authentication, message delivery, and data encryption. Automated testing frameworks and tools are utilized to streamline the testing process and identify defects and issues early in the development cycle.

Deployment involves deploying the Realtime Messenger application to production environments, making it accessible to end-users and stakeholders. Continuous monitoring and maintenance activities are performed to ensure the stability, security, and scalability of the messaging platform post-deployment. Additionally, user training and support services are provided to assist users in navigating and utilizing the features of the messaging application effectively.

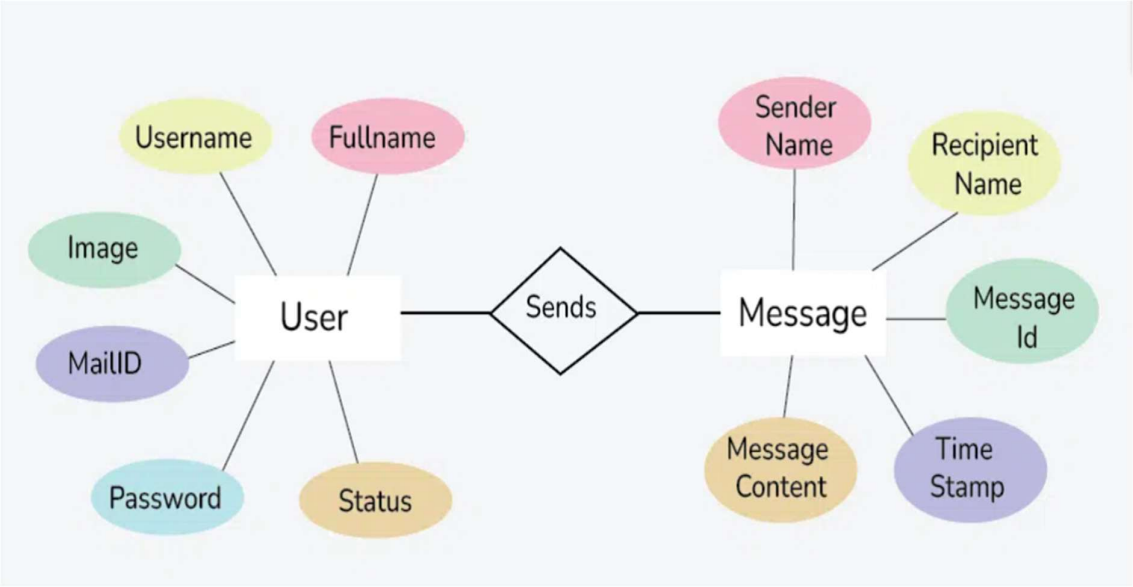
CHAPTER 6

DESIGN FRAMEWORK

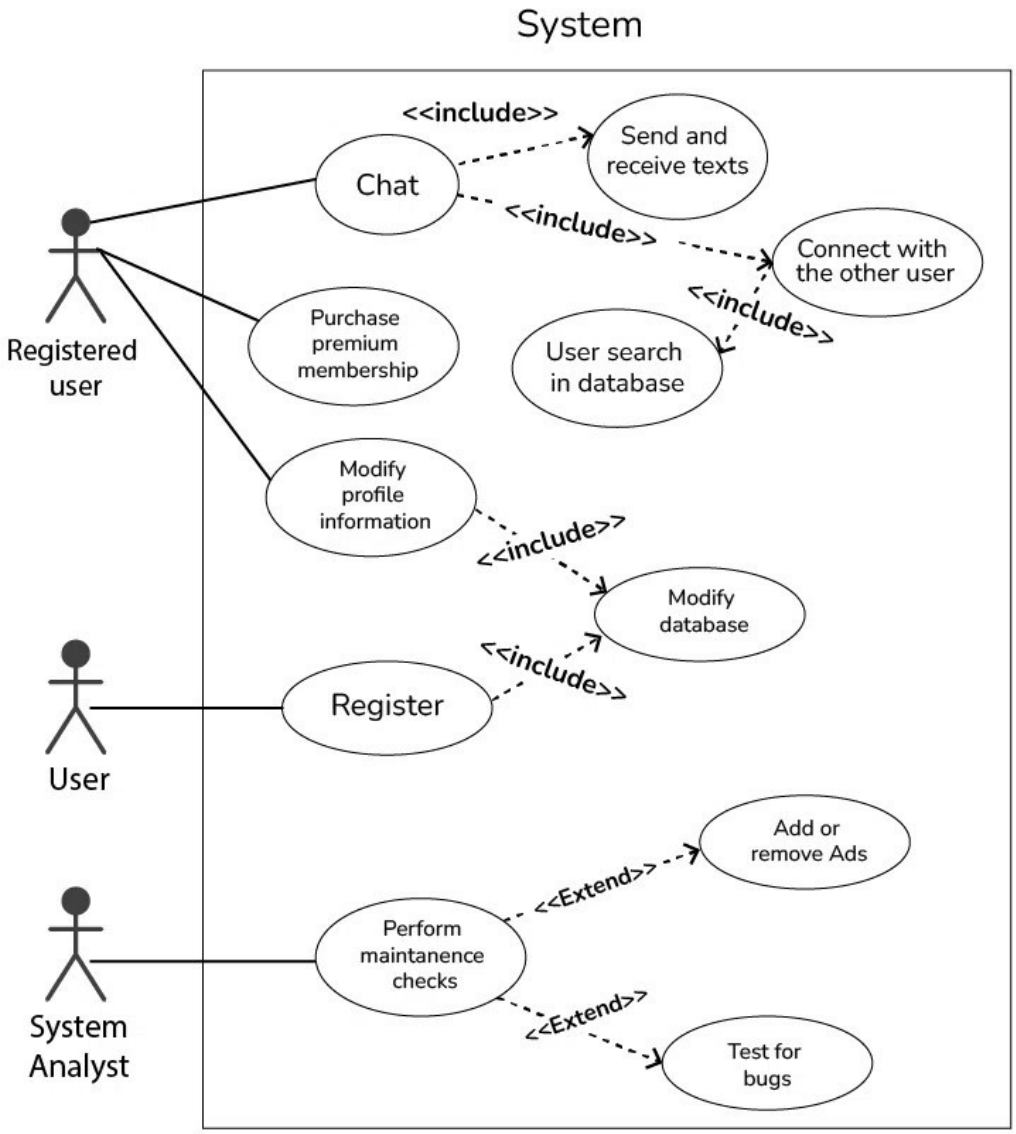
The design framework chapter presents the architectural and design considerations underlying the Realtime Messenger application, offering insights into the structural and functional aspects of the messaging platform. It serves as a blueprint for understanding the system architecture, data flow, and user interactions within the application.

6.1 ER Diagram

The Entity-Relationship (ER) diagram is a foundational component of the design framework, illustrating the database schema and relationships between various entities in the Realtime Messenger application. Entities such as users, messages, conversations, and contacts are identified and organized based on their attributes and associations. Relationships such as one-to-one, one-to-many, and many-to-many are depicted to capture the complex interactions and dependencies between different entities. The ER diagram serves as a visual representation of the data model underlying the messaging platform, guiding the implementation of database schemas and queries.



6.2 Use Case Diagram



The use case diagram is another essential artifact in the design framework, depicting the interactions between actors (users) and the system, along with the main functionalities and features of the Realtime Messenger application. Use cases such as user registration, login, message sending, message receiving, and contact management are identified and described to capture the core user interactions and system behaviors. Actors such as regular users, administrators, and system integrations are defined to represent the various roles and

permissions within the messaging platform. The use case diagram provides a holistic view of the system's functionality, guiding the development of user interfaces, backend logic, and system workflows.

6.3 DFD Diagram

The data flow diagram (DFD) complements the design framework by illustrating the flow of data and control within the Realtime Messenger application, from user input to message delivery. The DFD consists of various components such as processes, data stores, data flows, and external entities, interconnected through data flow paths and control flows. Processes

represent the computational tasks and operations performed within the system, while data stores represent the storage repositories for application data. Data flows depict the movement of data between different components, facilitating message exchange and data processing. External entities represent the external systems and interfaces that interact with the messaging platform, such as user devices, external APIs, and backend services. The data flow diagram provides a comprehensive view of the system's data flow and processing logic, enabling developers to identify bottlenecks, optimize performance, and ensure data integrity.

CHAPTER 7

IMPLEMENTATION

The implementation chapter details the process of translating the design specifications into functional code for the Realtime Messenger application, offering insights into the development environment, coding practices, and technical challenges encountered during the implementation phase.

The development environment setup is a crucial aspect of the implementation process, ensuring that developers have access to the necessary tools, libraries, and resources for coding and testing the messaging platform. The chapter discusses the installation and configuration of development environments for frontend and backend development, including IDEs (Integrated Development Environments), text editors, version control systems, and dependency management tools. Additionally, it outlines the setup of development databases, mock servers, and testing frameworks to facilitate rapid iteration and debugging.

The implementation of key features and functionalities of the Realtime Messenger application is a collaborative effort involving frontend developers, backend developers, and UI/UX designers. The chapter delves into the coding practices and design patterns used in developing various components of the messaging platform, such as user authentication mechanisms, message encryption algorithms, and real-time communication protocols. It highlights the use of Python programming language for backend development, PyFlet framework for frontend development, and SQLITE3 for database management, emphasizing modularity, encapsulation, and code reusability.

Challenges encountered during the implementation phase are inevitable, ranging from technical complexities to resource constraints and timeline pressures. The chapter discusses the strategies employed to overcome these challenges, including problem-solving techniques, collaboration tools, and continuous communication among team members. Additionally, it addresses the importance of code review, testing, and quality assurance practices in ensuring the reliability and stability of the messaging platform.

Continuous integration and deployment (CI/CD) practices are integral to the implementation process, enabling automated build, test, and deployment pipelines for the Realtime Messenger application. The chapter discusses the setup of CI/CD pipelines, integration with version control systems, and deployment strategies for deploying new features and updates to production environments seamlessly. Furthermore, it highlights the use of monitoring and logging tools to track application performance, identify issues, and ensure system reliability post-deployment.

7.1 Snapshots

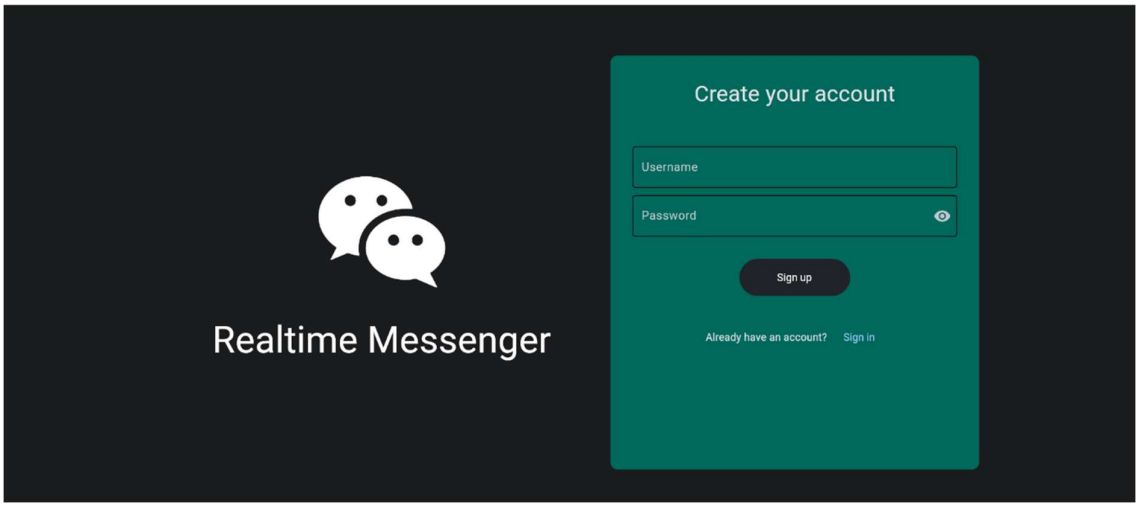


Fig : (i) Sign Up

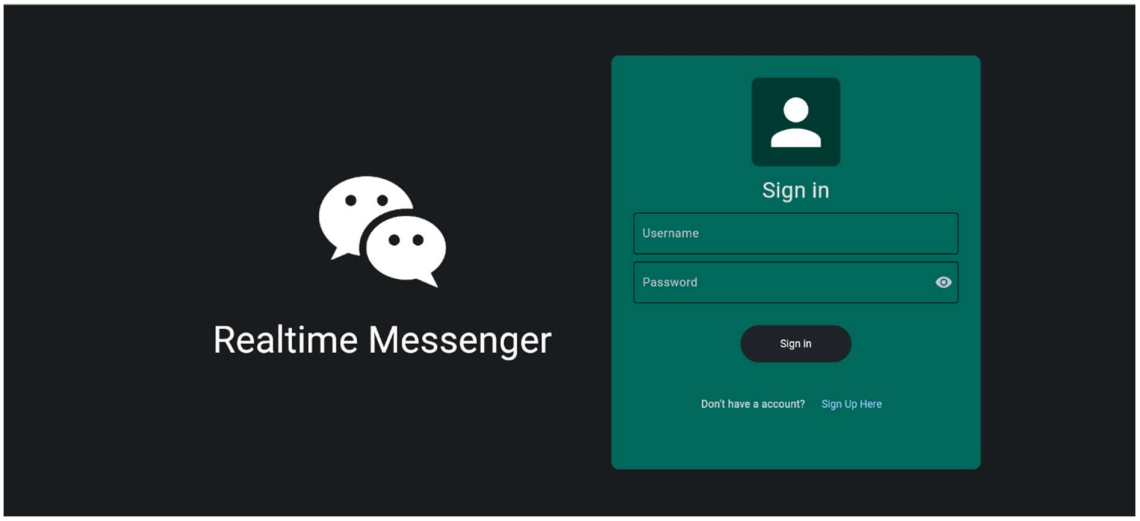


Fig:(b) Sign In

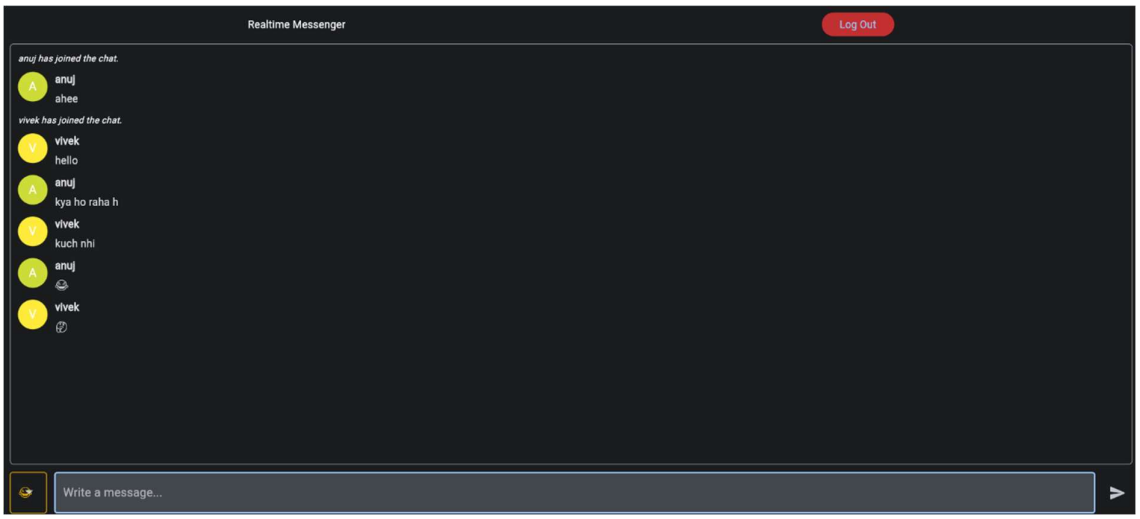


Fig:(c) User 1 view

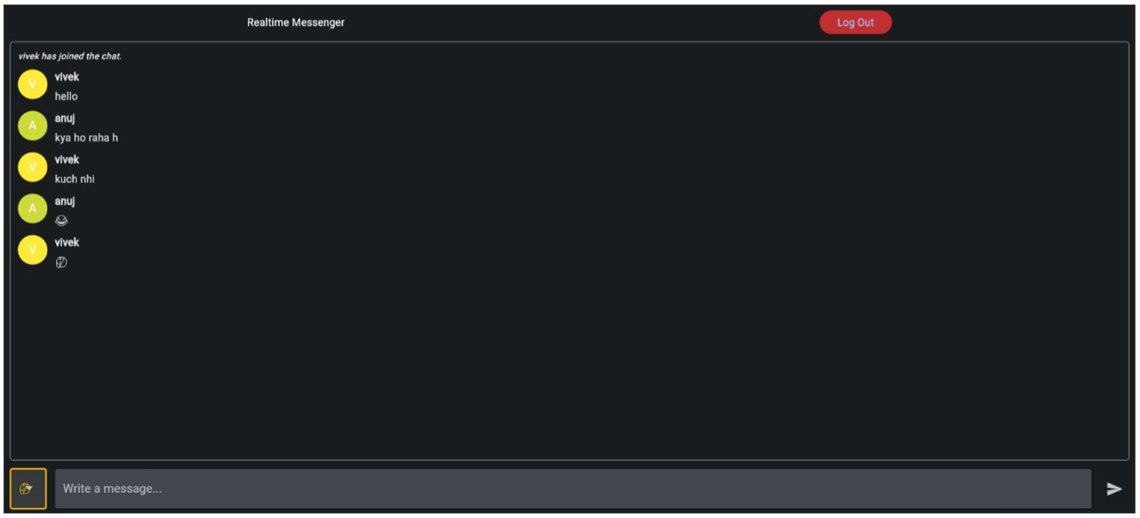


Fig:(d) User 2 view

7.2 Coding

File named "chat_message.py"

```
import flet as ft
class Message():
    def __init__(self,user:str,text:str,message_type:str):
        self.user=user
        self.text=text
        self.message_type=message_type

class ChatMessage(ft.Row):
    def __init__(self, message: Message):
        super().__init__()
        self.vertical_alignment="start"
        self.controls=[
                ft.CircleAvatar(
                    content=ft.Text(self.get_initials(message.user)),
                    color=ft.colors.WHITE,
                    bgcolor=self.get_avatar_color(message.user),
                ),
                ft.Column(
                    [
                        ft.Text(message.user, weight="bold"),
                        ft.Text(message.text, selectable=True),
                    ],
                    tight=True,
```

```
                    spacing=5,
                ),
            ]

    def get_initials(self, user: str):
        return user[:1].capitalize()

    def get_avatar_color(self, user: str):
        colors_lookup = [
            ft.colors.AMBER,
            ft.colors.BLUE,
            ft.colors.BROWN,
            ft.colors.CYAN,
            ft.colors.GREEN,
            ft.colors.INDIGO,
            ft.colors.LIME,
            ft.colors.ORANGE,
            ft.colors.PINK,
            ft.colors.PURPLE,
            ft.colors.RED,
            ft.colors.TEAL,
            ft.colors.YELLOW,
        ]
        return colors_lookup[hash(user) % len(colors_lookup)]
```

File named "main.py"

```
import flet as ft
from signin_form import *
from signup_form import *
from users_db import *
from chat_message import *

def main(page: ft.Page):
    page.title = "Realtime Messenger"
    page.vertical_alignment = ft.MainAxisAlignment.CENTER
    page.horizontal_alignment = ft.CrossAxisAlignment.CENTER

    # **************  Functions             *************
    def dropdown_changed(e):
        new_message.value = new_message.value + emoji_list.value
        page.update()
```

```
    def close_banner(e):
        page.banner.open = False
        page.update()

    def open_dlg():
        page.dialog = dlg
        dlg.open = True
        page.update()

    def close_dlg(e):
        dlg.open = False
        page.route = "/"
        page.update()

    def sign_in(user: str, password: str):
        db = UsersDB()
        if not db.read_db(user, password):
            print("User no exist ...")
            page.banner.open = True
            page.update()
        else:
            print("Redirecting to chat...")
            page.session.set("user", user)
            page.route = "/chat"
            page.pubsub.send_all(
                Message(
                    user=user,
                    text=f"{user} has joined the chat.",
                    message_type="login_message",
                )
            )
            page.update()

    def sign_up(user: str, password: str):
        db = UsersDB()
        if db.write_db(user, password):
            print("Successfully Registered User...")
            open_dlg()

    def on_message(message: Message):
        if message.message_type == "chat_message":
            m = ChatMessage(message)
        elif message.message_type == "login_message":
            m = ft.Text(message.text, italic=True, color=ft.colors.WHITE,
size=12)
        chat.controls.append(m)
        page.update()
```

```
    page.pubsub.subscribe(on_message)

    def send_message_click(e):
        page.pubsub.send_all(
            Message(
                user=page.session.get("user"),
                text=new_message.value,
                message_type="chat_message",
            )
        )
        new_message.value = ""
        page.update()

    def btn_signin(e):
        page.route = "/"
        page.update()

    def btn_signup(e):
        page.route = "/signup"
        page.update()

    def btn_exit(e):
        page.session.remove("user")
        page.route = "/"
        page.update()

    # ************          Aplication
UI              **********************************
    principal_content = ft.Column(
        [
            ft.Icon(ft.icons.WECHAT, size=200, color=ft.colors.WHITE),
            ft.Text(value="Realtime Messenger", size=50,
color=ft.colors.WHITE),
        ],
        height=400,
        width=600,
        alignment=ft.MainAxisAlignment.CENTER,
        horizontal_alignment=ft.CrossAxisAlignment.CENTER,
    )
    emoji_list = ft.Dropdown(
        on_change=dropdown_changed,
        options=[
            ft.dropdown.Option("😀"),
            ft.dropdown.Option("😊"),
            ft.dropdown.Option("😂"),
            ft.dropdown.Option("😱"),
            ft.dropdown.Option("😭"),
            ft.dropdown.Option("😉"),
```

```
            ft.dropdown.Option("😍"),
            ft.dropdown.Option("🥰"),
            ft.dropdown.Option("😎"),
            ft.dropdown.Option("❤️"),
            ft.dropdown.Option("🔥"),
            ft.dropdown.Option("☑️"),
            ft.dropdown.Option("✨"),
            ft.dropdown.Option("👍"),
            ft.dropdown.Option("🎉"),
            ft.dropdown.Option("👉"),
            ft.dropdown.Option("⭐"),
            ft.dropdown.Option("🌕"),
            ft.dropdown.Option("👀"),
            ft.dropdown.Option("👇"),
            ft.dropdown.Option("🚀"),
            ft.dropdown.Option("🎂"),
            ft.dropdown.Option("💕"),
            ft.dropdown.Option("🏠"),
            ft.dropdown.Option("🏀"),
            ft.dropdown.Option("🎁"),
            ft.dropdown.Option("💯"),
            ft.dropdown.Option("💤"),
            ft.dropdown.Option("☠️"),
            ft.dropdown.Option("🍊"),
        ],
        width=50,
        value="😀",
        alignment=ft.alignment.center,
        border_color=ft.colors.AMBER,
        color=ft.colors.AMBER
    )

    signin_UI = SignInForm(sign_in, btn_signup)
    signup_UI = SignUpForm(sign_up, btn_signin)

    chat = ft.ListView(
        expand=True,
        spacing=10,
        auto_scroll=True,
    )

    new_message = ft.TextField(
        hint_text="Write a message...",
        autofocus=True,
        shift_enter=True,
        min_lines=1,
        max_lines=5,
        filled=True,
```

```
        expand=True,
        on_submit=send_message_click,
    )

    page.banner = ft.Banner(
        bgcolor=ft.colors.BLACK45,
        leading=ft.Icon(ft.icons.ERROR, color=ft.colors.RED, size=40),
        content=ft.Text("Log in failed, Incorrect User Name or Password"),
        actions=[
            ft.TextButton("Ok", on_click=close_banner),
        ],
    )

    dlg = ft.AlertDialog(
        modal=True,
        title=ft.Container(
            content=ft.Icon(
                name=ft.icons.CHECK_CIRCLE_OUTLINED, color=ft.colors.GREEN,
size=100
            ),
            width=120,
            height=120,
        ),
        content=ft.Text(
            value="Congratulations,\n your account has been successfully
created\n Please Sign In",
            text_align=ft.TextAlign.CENTER,
        ),
        actions=[
            ft.ElevatedButton(
                text="Continue", color=ft.colors.WHITE, on_click=close_dlg
            )
        ],
        actions_alignment="center",
        on_dismiss=lambda e: print("Dialog dismissed!"),
    )

    # ***************        Routes              *****************
    def route_change(route):
        if page.route == "/":
            page.clean()
            page.add(
                ft.Row(
                    [principal_content, signin_UI],
                    alignment=ft.MainAxisAlignment.CENTER,
                )
            )
```

```
        if page.route == "/signup":
            page.clean()
            page.add(
                ft.Row(
                    [principal_content, signup_UI],
                    alignment=ft.MainAxisAlignment.CENTER,
                )
            )

        if page.route == "/chat":
            if page.session.contains_key("user"):
                page.clean()
                page.add(
                    ft.Row(
                        [
                            ft.Text(value="Realtime Messenger",
color=ft.colors.WHITE),
                            ft.ElevatedButton(
                                text="Log Out",
                                bgcolor=ft.colors.RED_800,
                                on_click=btn_exit,
                            ),
                        ],
                        alignment=ft.MainAxisAlignment.SPACE_AROUND,
                    )
                )
                page.add(
                    ft.Container(
                        content=chat,
                        border=ft.border.all(1, ft.colors.OUTLINE),
                        border_radius=5,
                        padding=10,
                        expand=True,
                    )
                )
                page.add(
                    ft.Row(
                        controls=[
                            emoji_list,
                            new_message,
                            ft.IconButton(
                                icon=ft.icons.SEND_ROUNDED,
                                tooltip="Send message",
                                on_click=send_message_click,
                            ),
                        ],
                    )
                )
```

```
            else:
                page.route = "/"
                page.update()

    page.on_route_change = route_change
    page.add(
        ft.Row([principal_content, signin_UI],
alignment=ft.MainAxisAlignment.CENTER)
    )
    # Start the application
ft.app(target=main, view=ft.WEB_BROWSER)
```

File named "sign_up.py"

```
import flet as ft

# SignUp Form
class SignUpForm(ft.UserControl):
    def __init__(self, submit_values,btn_signin):
        super().__init__()
        #Return values user and password
        self.submit_values = submit_values
        #Route to signup Form
        self.btn_signin = btn_signin


    def btn_signup(self, e):
        if not self.text_user.value:
            self.text_user.error_text="Name cannot be blank!"
            self.text_user.update()
        if not self.text_password.value:
            self.text_password.error_text="Password cannot be blank!"
            self.text_password.update()
        else:
            #Return values 'user' and 'password' as arguments
            self.submit_values(self.text_user.value,self.text_password.value)
    def build(self):
        self.title_form=ft.Text(value="Create your
account",text_align=ft.TextAlign.CENTER,size=30, )
        self.text_user = ft.TextField(label="Username")
        self.text_password = ft.TextField(
            label="Password", password=True, can_reveal_password=True
        )
```

```
        self.text_signup=ft.ElevatedButton(text="Sign
up",color=ft.colors.WHITE,width=150,height=50,on_click= self.btn_signup)
        self.text_signin=ft.Row(controls=[ft.Text(value="Already have an
account?"),ft.TextButton(text="Sign
in",on_click=self.btn_signin)],alignment=ft.MainAxisAlignment.CENTER)


        return ft.Container(
            width=500,
            height=560,
            bgcolor=ft.colors.TEAL_800,
            padding=30,
            border_radius=10,
            alignment=ft.alignment.center,
            content=ft.Column(
                [
                    self.title_form,
                    ft.Container(height=30),
                    self.text_user,
                    self.text_password,
                    ft.Container(height=10),
                    self.text_signup,
                    ft.Container(height=20),
                    self.text_signin,

                ],
                horizontal_alignment=ft.CrossAxisAlignment.CENTER,
            ),
        )
```

File named "sign_in.py"

```
import flet as ft

# SignIn Form
class SignInForm(ft.UserControl):
    def __init__(self, submit_values,btn_signup):
        super().__init__()
        #Return values user and password
        self.submit_values = submit_values
        #Route to Sign Up Form
        self.btn_signup = btn_signup

    def btn_signin(self, e):
        if not self.text_user.value:
```

```
            self.text_user.error_text="Name cannot be blank!"
            self.text_user.update()
        if not self.text_password.value:
            self.text_password.error_text="Password cannot be blank!"
            self.text_password.update()
        else:
            #Return values 'user' and 'password' as arguments
            self.submit_values(self.text_user.value,self.text_password.value)

    def build(self):
        self.signin_image = ft.Container(
            content=ft.Icon(name=ft.icons.PERSON, color=ft.colors.WHITE,
size=100),
            width=120,
            height=120,
            bgcolor=ft.colors.BLACK45,
            border_radius=10,
        )
        self.title_form=ft.Text(value="Sign
in",text_align=ft.TextAlign.CENTER,size=30, )

        self.text_user = ft.TextField(label="Username")
        self.text_password = ft.TextField(
            label="Password", password=True, can_reveal_password=True
        )
        self.text_signin=ft.ElevatedButton(text="Sign
in",color=ft.colors.WHITE,width=150,height=50,on_click= self.btn_signin)
        self.text_signup=ft.Row(controls=[ft.Text(value="Don't have a
account?"),ft.TextButton(text="Sign Up
Here",on_click=self.btn_signup)],alignment=ft.MainAxisAlignment.CENTER)


        return ft.Container(
            width=500,
            height=560,
            bgcolor=ft.colors.TEAL_800,
            padding=30,
            border_radius=10,
            alignment=ft.alignment.center,
            content=ft.Column(
                [
                    self.signin_image,
                    self.title_form,
                    self.text_user,
                    self.text_password,
                    ft.Container(height=10),
                    self.text_signin,
                    ft.Container(height=20),
```

```
                self.text_signup,
            ],
            horizontal_alignment=ft.CrossAxisAlignment.CENTER,
        ),
    )
```

CHAPTER 8

TESTING

The testing chapter focuses on validating and verifying the functionality, reliability, and performance of the Realtime Messenger application, offering insights into the testing methodologies, strategies, and best practices employed to ensure the quality of the messaging platform.

8.1 Testing

Testing is an integral part of the development process, serving as a critical quality assurance mechanism to identify defects, errors, and vulnerabilities in the Realtime Messenger application. The chapter discusses various testing methodologies employed, including unit testing, integration testing, system testing, and user acceptance testing (UAT). Each testing methodology targets different aspects of the application, from individual components to end-to-end user workflows, ensuring comprehensive test coverage and thorough validation of the messaging platform.

Unit testing focuses on testing individual components or modules of the Realtime Messenger application in isolation, verifying their behavior and functionality against predefined test cases. The chapter discusses the use of testing frameworks and libraries such as pytest and unittest for writing and executing unit tests, ensuring code quality, consistency, and reliability. Mocking and stubbing techniques are employed to simulate external dependencies and isolate components for testing, facilitating rapid iteration and debugging.

Integration testing validates the interactions and interfaces between different components and modules of the Realtime Messenger application, ensuring seamless integration and interoperability. The chapter outlines the use of integration testing frameworks and tools to orchestrate and automate integration tests, verifying communication protocols, data exchange formats, and system interfaces. End-to-end testing scenarios are simulated to emulate real-world user interactions and workflows, validating the system's functionality and behavior under various usage scenarios.

System testing evaluates the overall functionality, reliability, and performance of the Realtime Messenger application as a whole, focusing on system-wide behaviors and interactions. The chapter discusses the use of system testing techniques such as regression testing, load testing, and stress testing to identify and mitigate performance bottlenecks, scalability issues, and reliability concerns. Test environments are configured to replicate production conditions, enabling realistic testing of the messaging platform's performance and behavior in live scenarios.

User acceptance testing (UAT) involves validating the Realtime Messenger application against user expectations, preferences, and requirements, ensuring that it meets the needs and desires of end-users effectively. The chapter discusses the involvement of stakeholders, including end-users, administrators, and domain experts, in defining test scenarios, executing test cases, and providing feedback on the usability, functionality, and overall user experience of the messaging platform. UAT serves as the final validation step before the deployment of the messaging platform to production environments, ensuring that it meets the quality standards and expectations of its intended audience.

CHAPTER 9

CONCLUSION & FUTURE SCOPE

The conclusion and future scope chapter offers a summary of the key findings, insights, and accomplishments of the Realtime Messenger project, along with recommendations for future enhancements and extensions to the messaging platform.

In conclusion, the Realtime Messenger project has successfully achieved its objectives of designing, developing, and deploying a robust and versatile messaging platform that addresses the evolving needs of modern users and organizations. The project has demonstrated the feasibility and effectiveness of leveraging emerging technologies and best practices to create a user-friendly, scalable, and secure messaging solution that facilitates real-time communication and collaboration.

Throughout the project lifecycle, several key accomplishments and milestones have been achieved, including the implementation of user authentication mechanisms, real-time messaging capabilities, and robust data encryption techniques. The project has also contributed to advancing knowledge and understanding in the field of real-time communication systems, through comprehensive research, analysis, and experimentation.

Looking ahead, there are several avenues for future enhancements and extensions to the Realtime Messenger application, aimed at further improving its functionality, usability, and scalability. One potential area for future development is the integration of multimedia messaging capabilities, allowing users to exchange not only text messages but also images, videos, and other multimedia content seamlessly. Additionally, enhancing security features such as end-to-end encryption and multi-factor authentication can further bolster the privacy and confidentiality of user communications.

Another promising direction for future development is the expansion of platform support to include mobile devices, enabling users to access the messaging platform on smartphones and tablets through dedicated mobile applications. This would enhance accessibility and

convenience for users, particularly those who prefer mobile-centric communication experiences.

Furthermore, exploring opportunities for integration with emerging technologies such as artificial intelligence (AI) and natural language processing (NLP) can unlock new possibilities for enhancing user experiences and automating repetitive tasks within the messaging platform. AI-powered chatbots, for example, can assist users in finding information, scheduling tasks, and performing other routine activities, thereby enhancing productivity and efficiency.

REFERENCES

1. Grinberg, Miguel. "Flask Web Development: Developing Web Applications with Python." O'Reilly Media, 2018.
2. Lutz, Mark. "Learning Python." O'Reilly Media, 2013.
3. Chopra, Varun. "WebSocket Essentials – Building Apps with HTML5 WebSockets." Packt Publishing, 2015.
4. Python Software Foundation. "Python Documentation." Available at: https://docs.python.org/3/.
5. Flask Documentation. "Flask Documentation." Available at: https://flask.palletsprojects.com/en/2.0.x/.
6. SQLite. "SQLite Documentation." Available at: https://www.sqlite.org/docs.html.
7. PyFlet Framework. "PyFlet Documentation." Available at: [Insert URL].
8. GitHub Repository for PyFlet. "PyFlet GitHub Repository." Available at: [Insert URL].
9. Academic Paper on Real-Time Messaging Systems. [Insert citation details].
10. Online Tutorial on PyFlet Development. [Insert citation details].