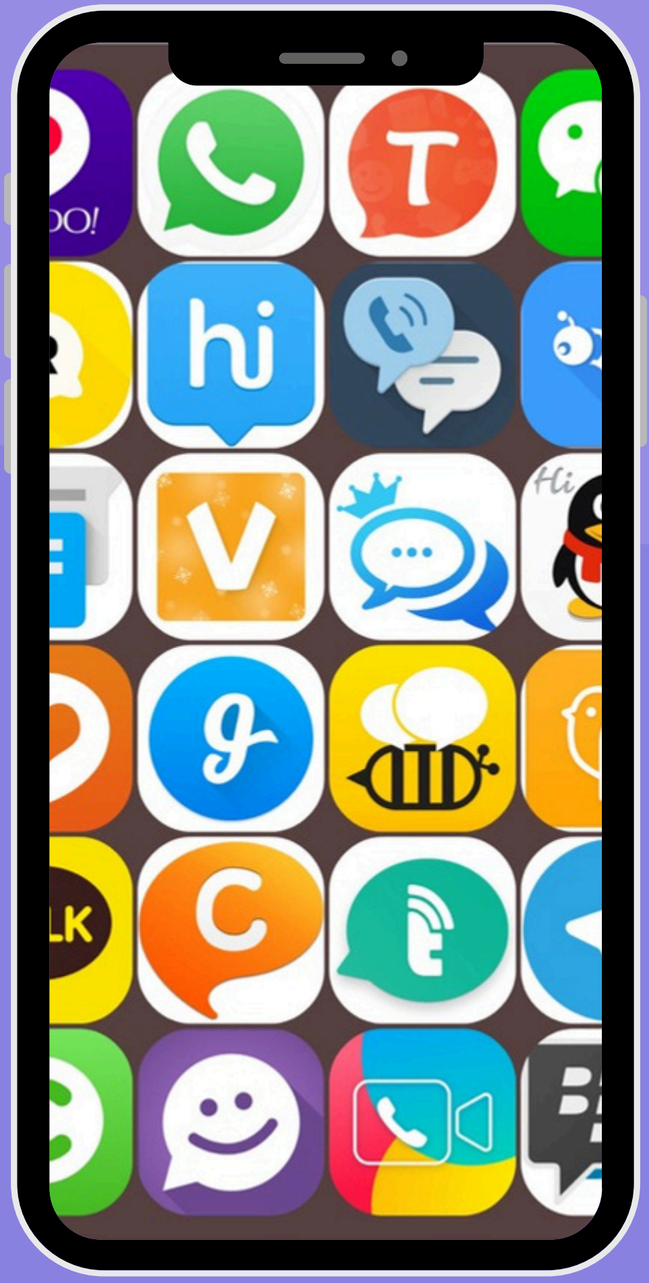
Home Contact

GROUP 1

Presentation

Chat Application



Home Contact

List of members

1. Chu Anh Duc - 20230081
2. Tran Quang Hung - 20235502
3. Phan Dinh Trung - 20230086
4. Vu Thuong Tin - 20230091
5. Nguyen The Quan - 20235548
6. Do Dang Vu - 20235578

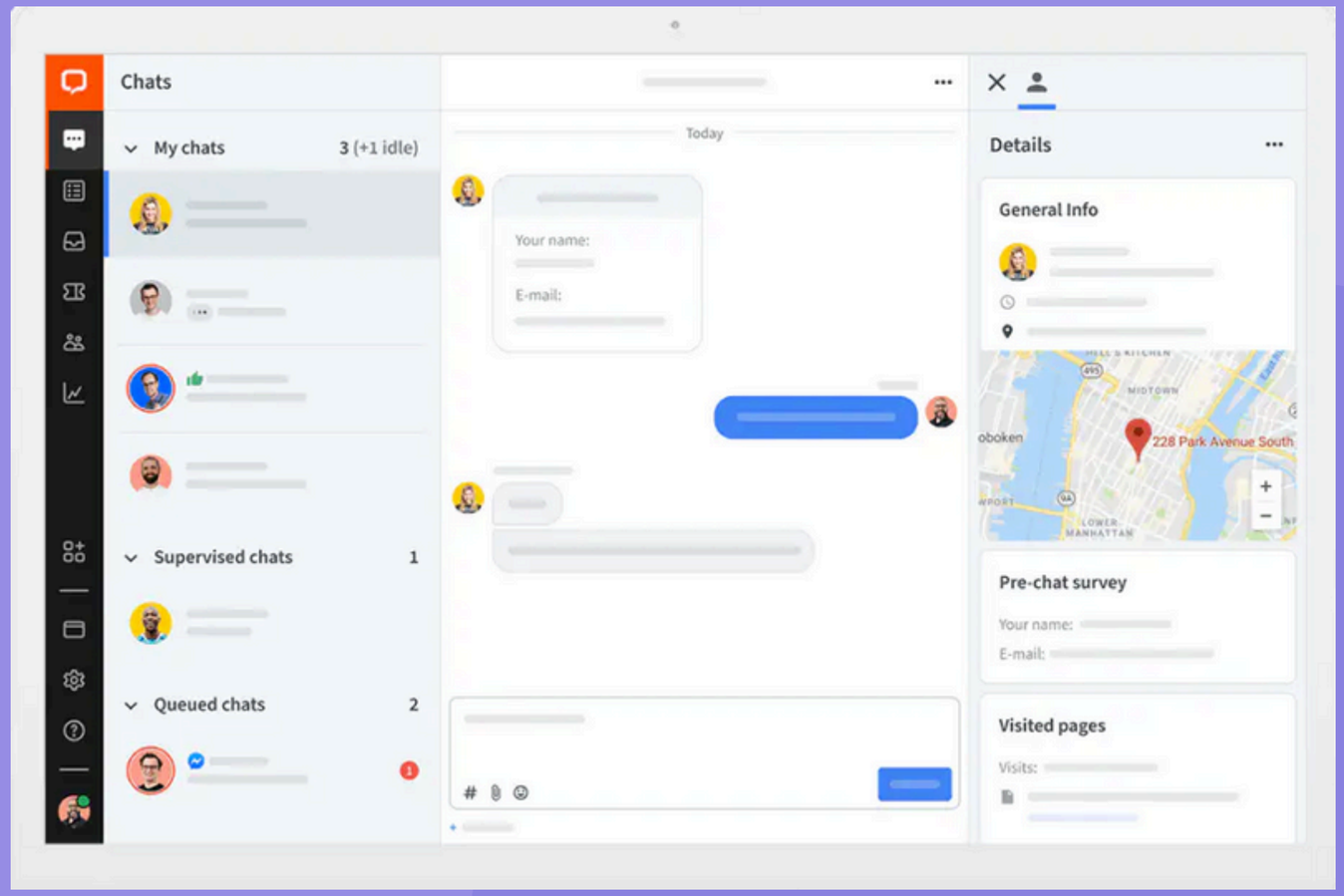
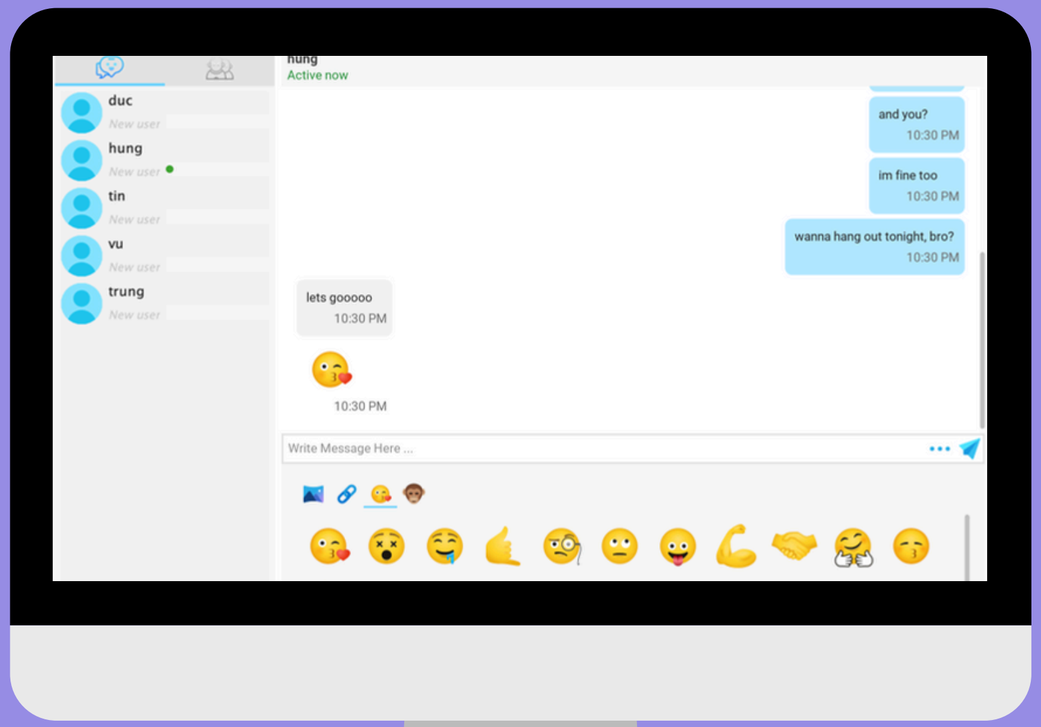


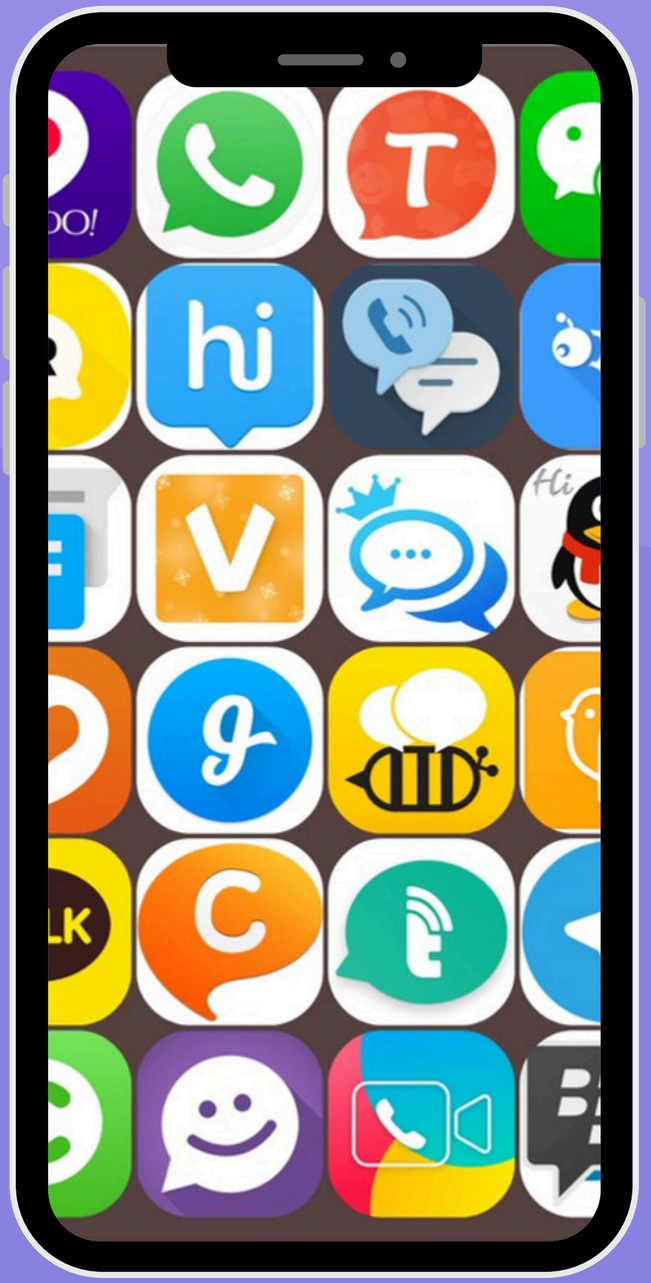
Table of Contents



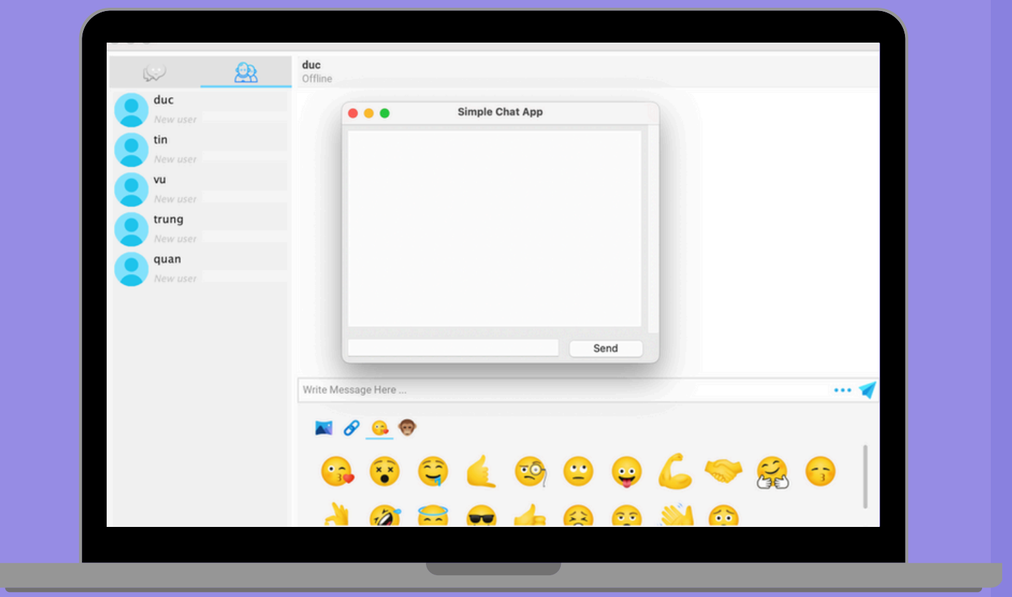
Introduction

Chat applications are communication tools that enable users to exchange messages, share media, and connect in real-time. They are often used for direct messaging, group discussions, and social interactions. With features like instant delivery and multimedia sharing, chat apps have become an essential part of modern communication. They serve a variety of purposes, making them highly versatile and impactful.

Therefore, we decided to create a chat app to gain a deeper understanding of its functionality and the underlying technology that powers it.



Apps Main Features



1

Sign Up and Log In

The app provides secure Sign Up and Log In features, allowing users to create accounts and access their profiles seamlessly.

2

Direct Messages

The app allows users to send Direct Messages for private one-on-one conversations and create Groups for collaborative and shared discussions.

3

Image and Emojis Sharing

Enabling users to express themselves visually and share engaging content in their conversations.

4

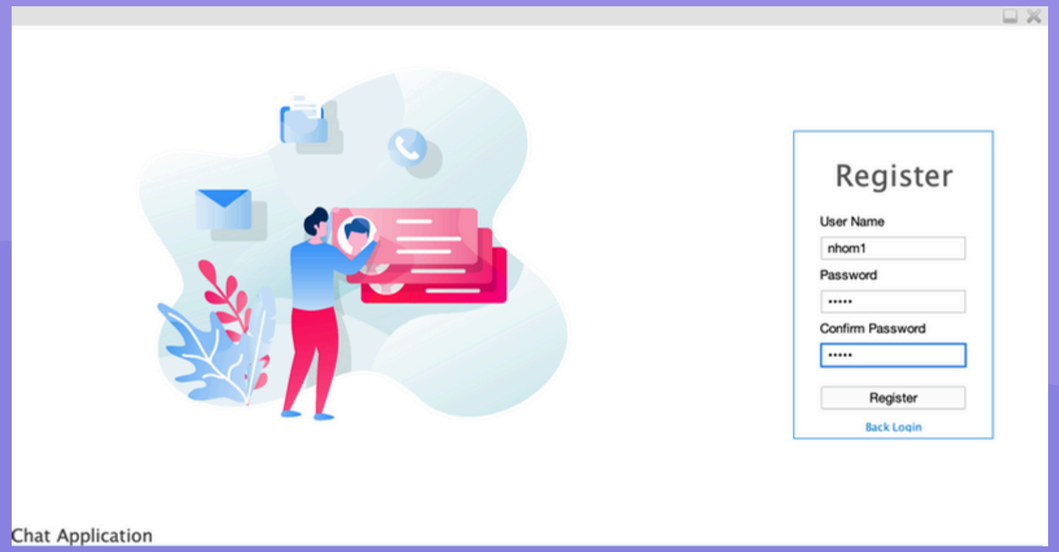
Chatbot with AI

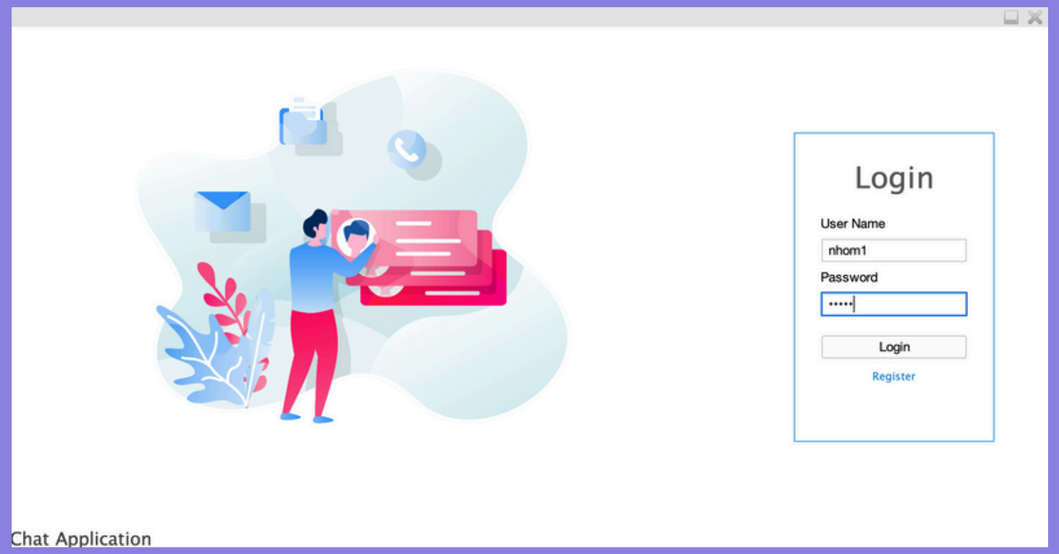
Providing users with automated, intelligent responses to enhance interactions and assist with inquiries.

Sign Up
Log in

To create an account, you have to provide a user name and a password. When registering an account, the app requires users to confirm their password to ensure it matches the initial entry, helping prevent errors and ensuring account security.

Then, you can use a registered account to log in and start chatting with other users.

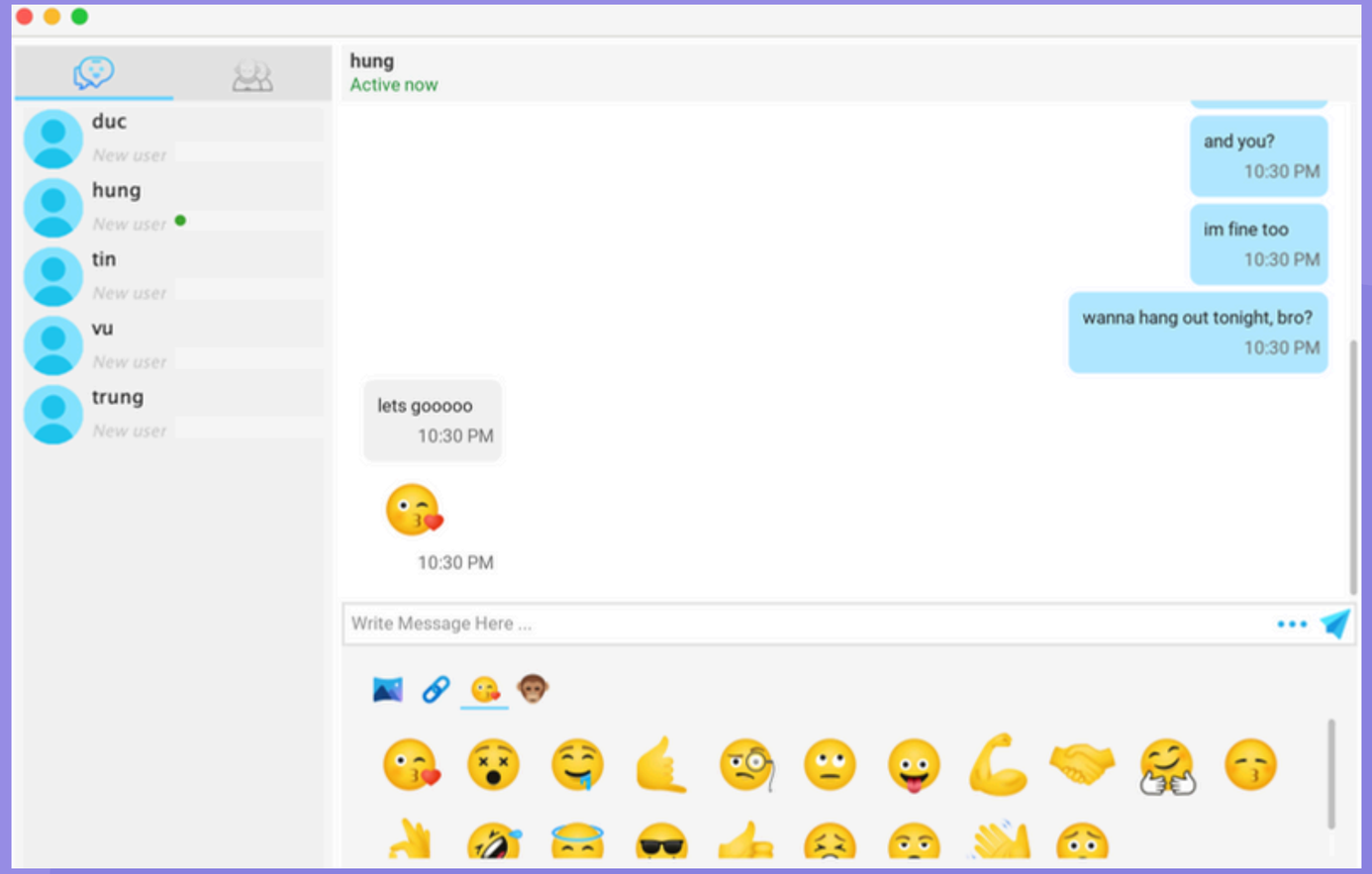




Main Interface

- **Contact list**: Displayed on the left panel, the contact list shows all available users. Each users are accompanied by an activity status to indicate if they are online or offline.

- **Chat window**: The central panel displays the conversation history with the selected contact or group. Messages are displayed in a structured format, clearly distinguishing between sent and received messages.



Main Interface

- **Message Input Box**: At the bottom of the chat window, users can type their messages
- **Emoji Panel**: Below the message input box, a wide range of emojis is available for users to enhance their communication with expressive icons.
- **Image sharing**: Users can send and receive images directly through the chat interface.

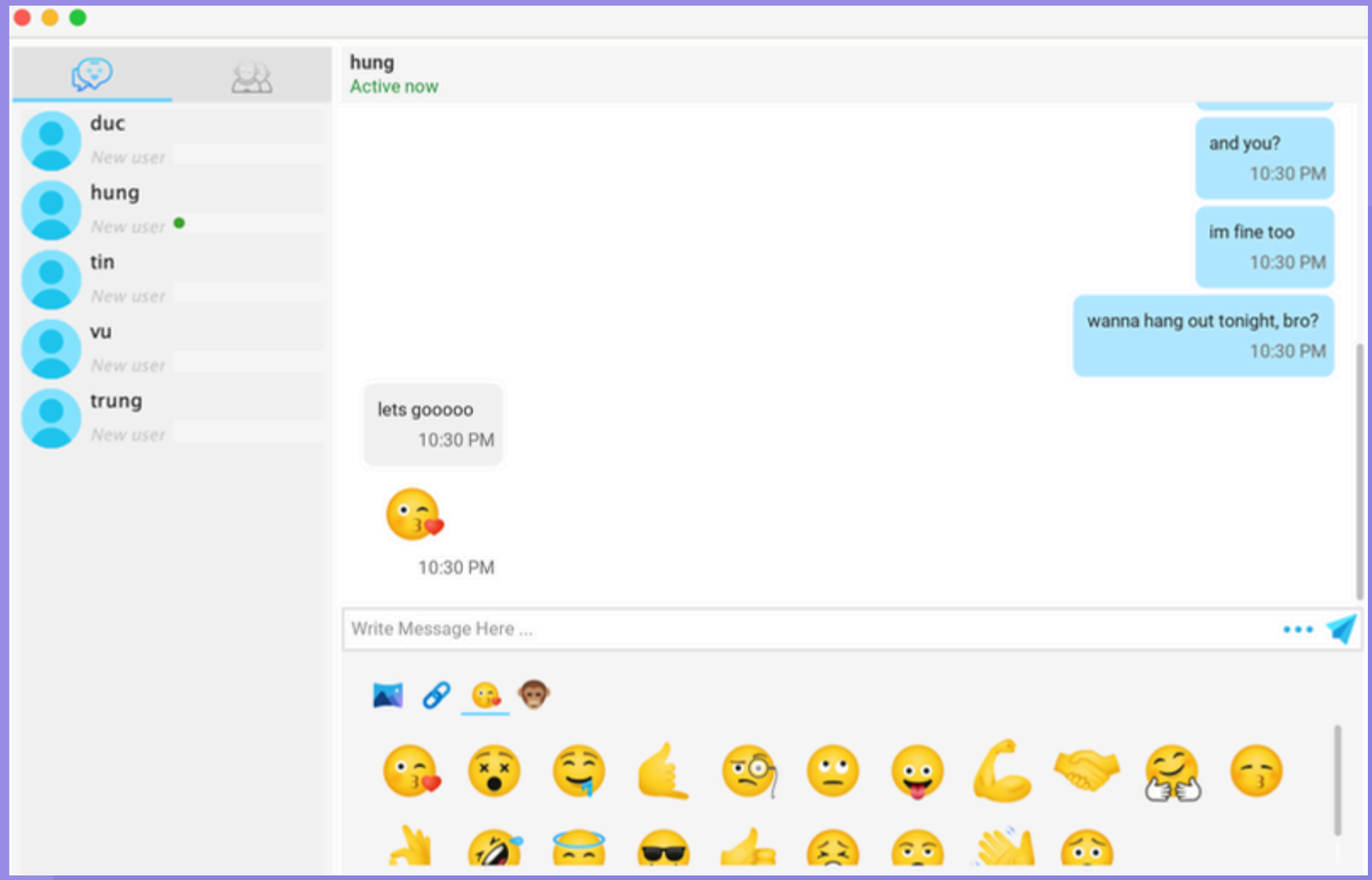
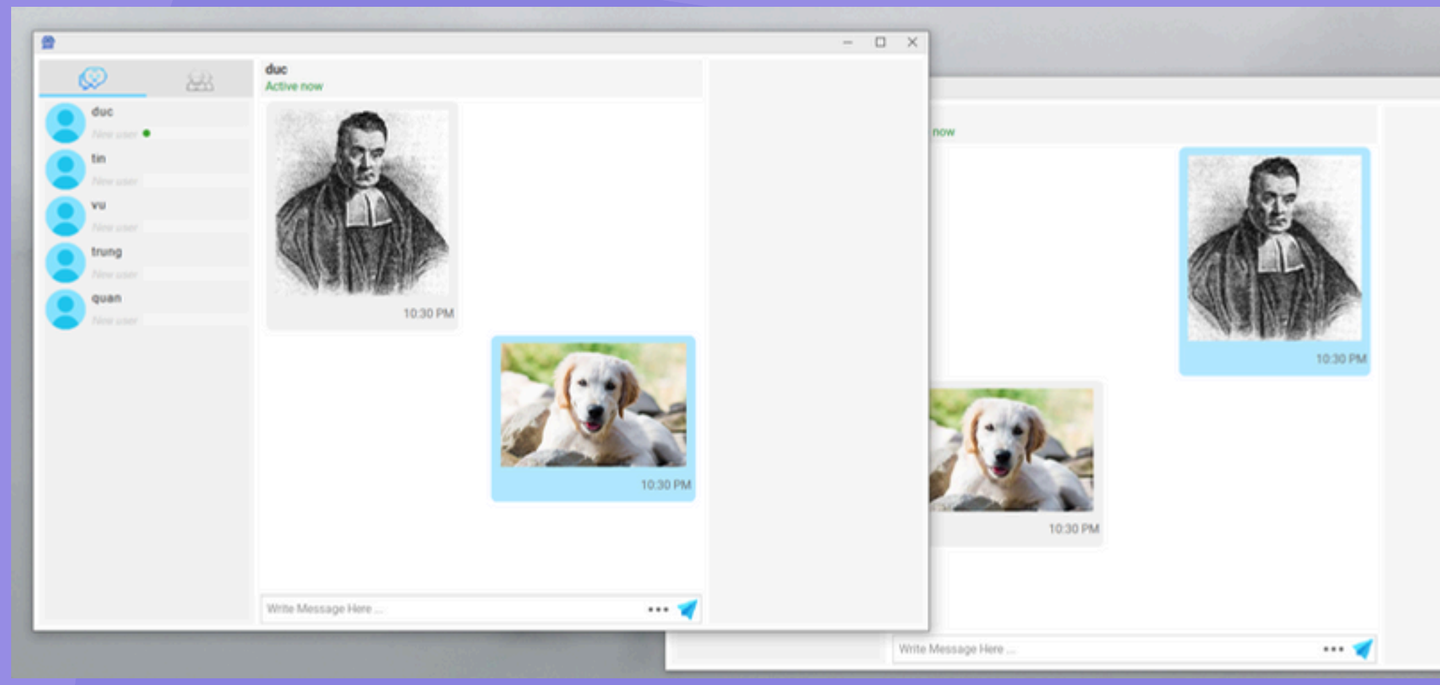


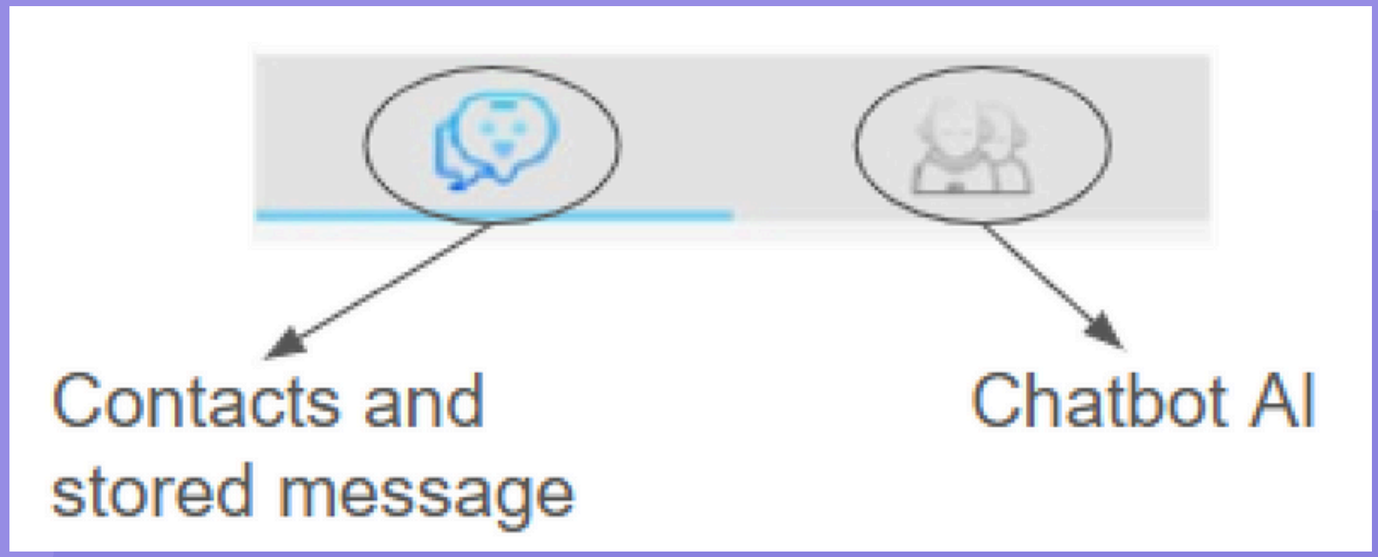
Image sharing feature

- This feature enhances communication by allowing users to share visual content directly within the chat interface. Users can easily select images from their devices and send them in real-time. The images are displayed inline with timestamps, ensuring the conversation remains clear and organized

- Particularly useful for sharing memorable moments or collaborating visually in professional settings



Two Sections



- **Contacts and stored message:** Allows users to view and manage their list of contacts, making it easy to initiate and maintain individual conversations.

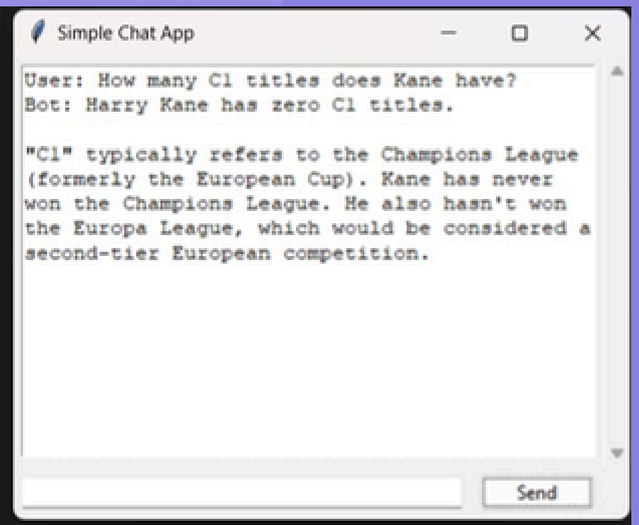
- **Chatbot AI:** This AI Chatbot can help users ask questions and get answers quickly, using the Gemini AI model

Chatbot

- The chatting application includes an AI-powered chatbot that enables users to interact with the system through natural language. This feature is accessible via the "Chatbot" button in the left menu, opening a user-friendly interface where queries can be entered and responses are displayed in real time.

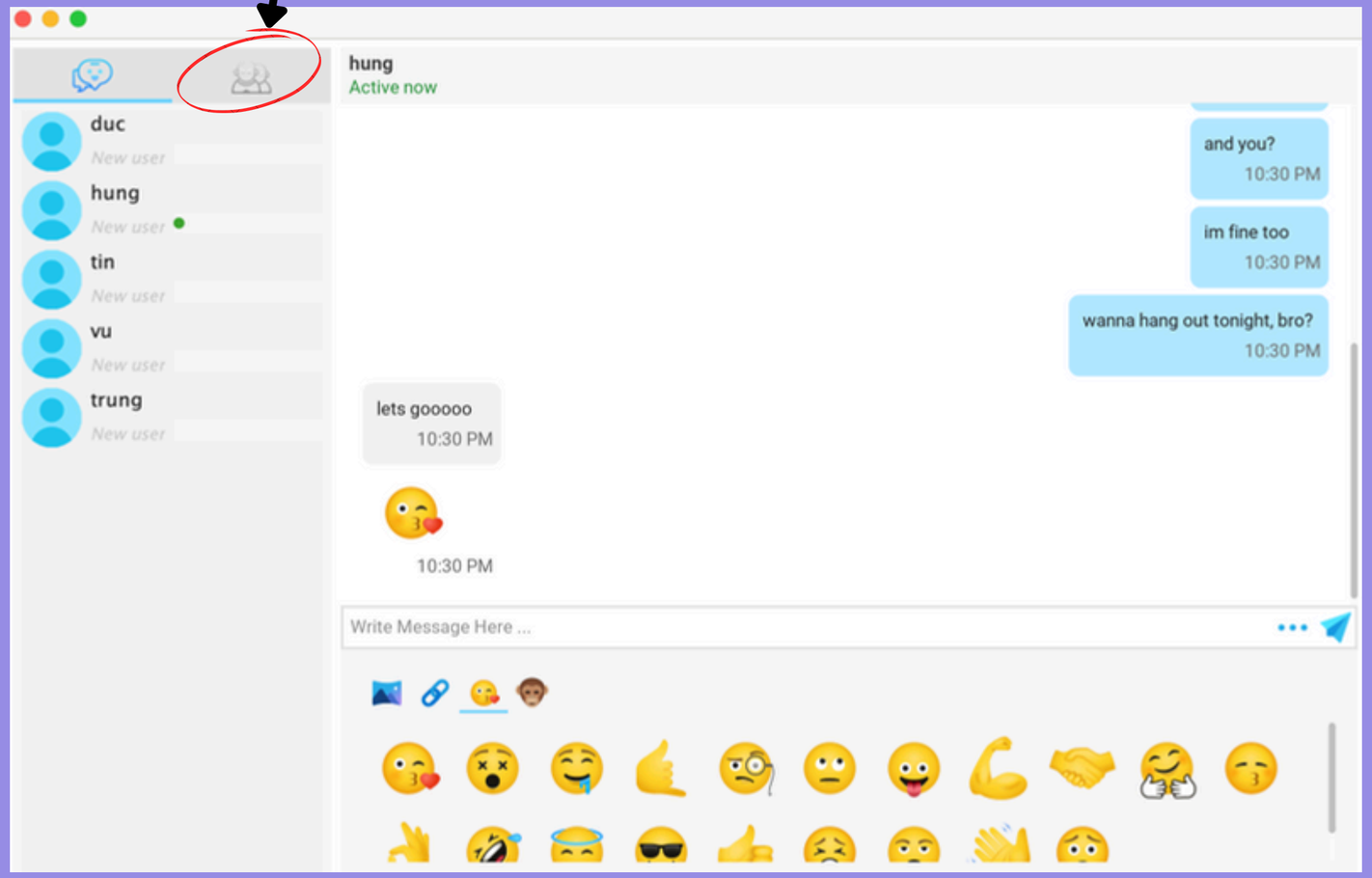
- The chatbot processes user queries using an AI model, configured with parameters such as temperature and token limits to ensure relevant and coherent responses.

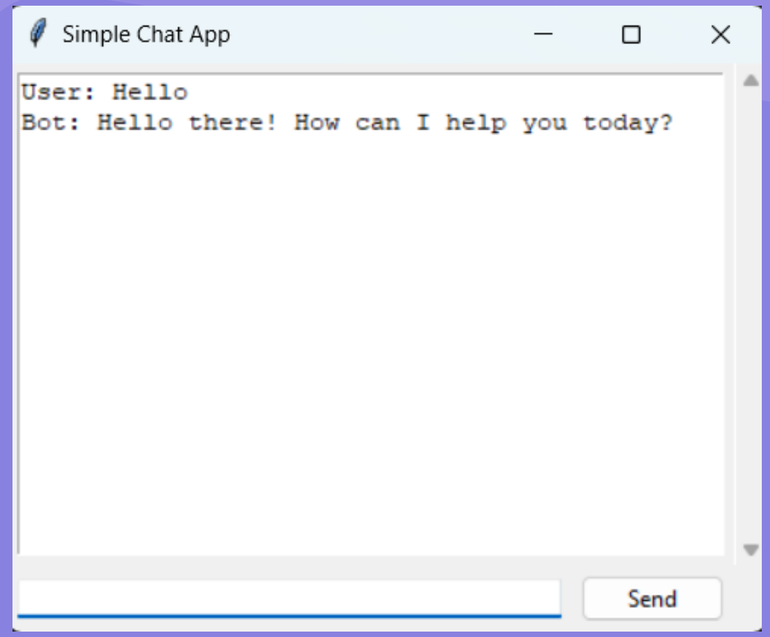
```python
import google.generativeai as genai

import tkinter as tk
from tkinter import ttk

history = []

def format_res(text: str):
    return text.replace('*', '')


def callgemini(input: str) -> str:
    global history

    gemini_api_key = "AIzaSyDQAajqUVN2fnKNLKvSY6z_rtBUoEDpxms"
    genai.configure(api_key=gemini_api_key)

    generation_config = {
        "temperature": 1,
        "top_p": 0.95,
        "top_k": 40,
        "max_output_tokens": 8192,
        "response_mime_type": "text/plain",
    }
```



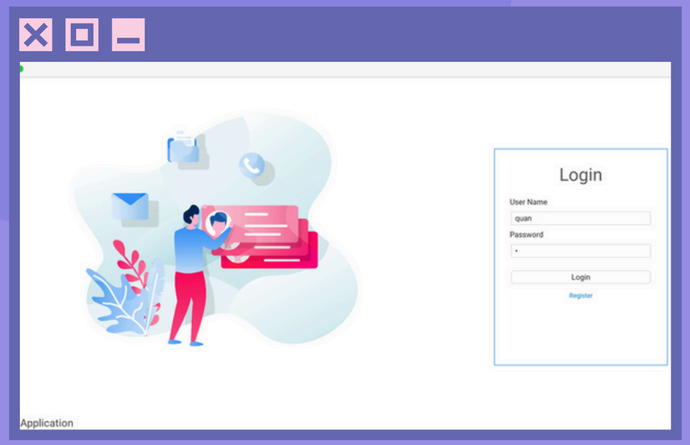
Chatbot Usage

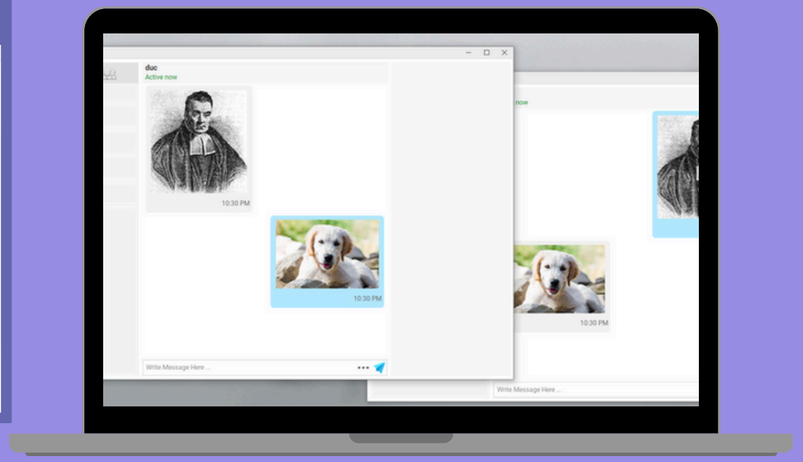
Click here



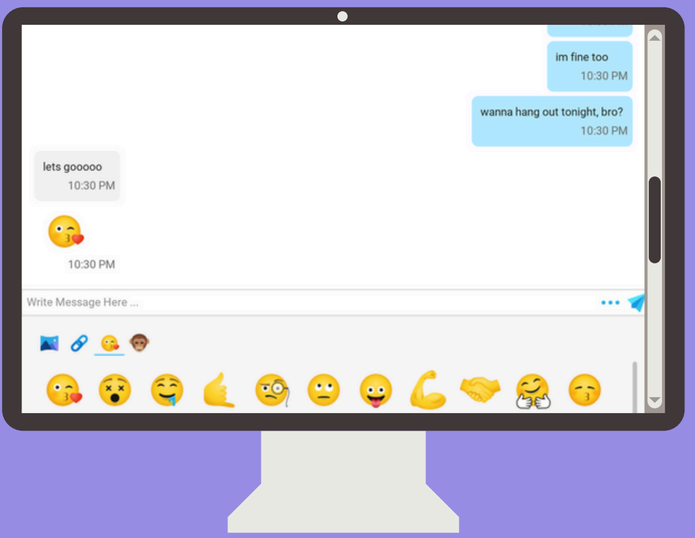


STRUCTURE



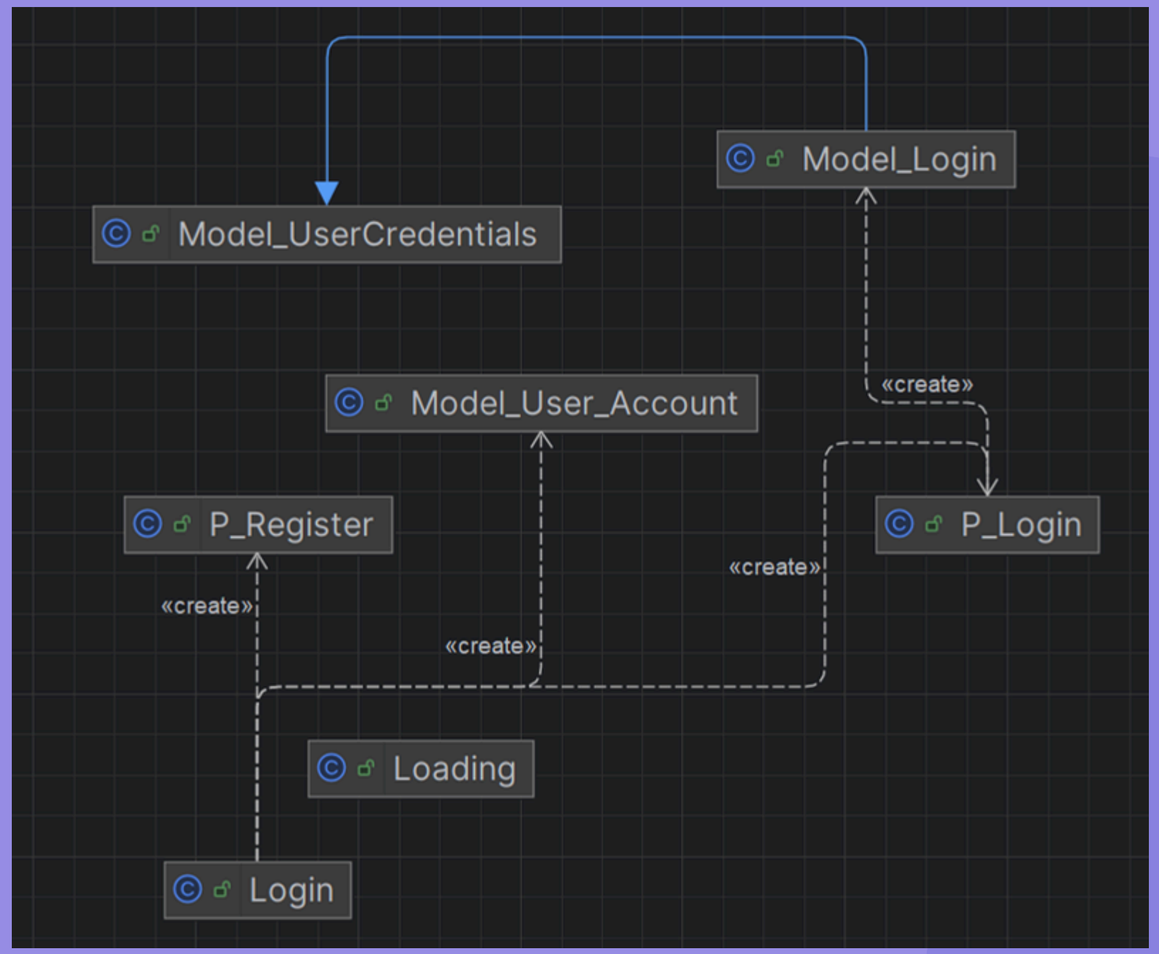


- Login & Register
- Server
- Sending direct items
  - Sending messages
  - Sending images
  - Sending icons
  - Active Status



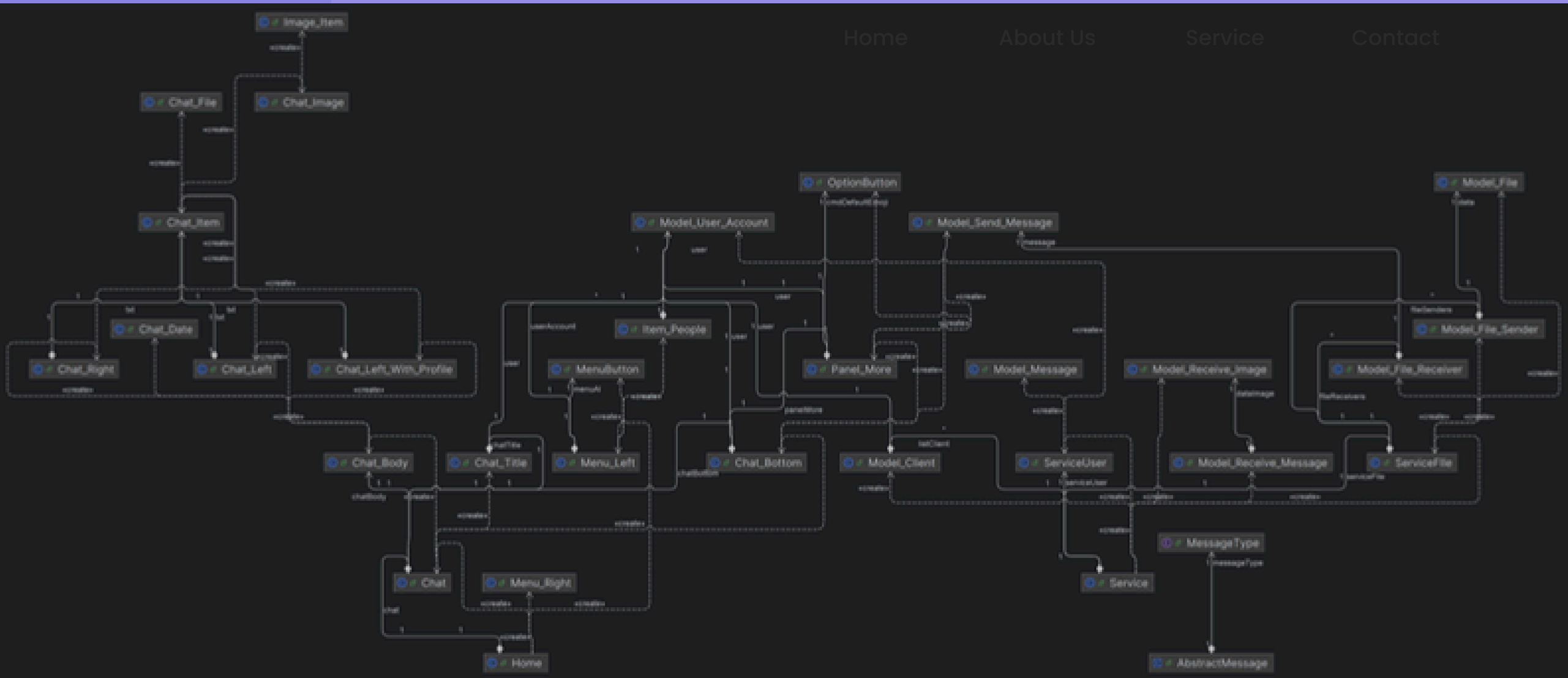
Login & Register

Log In & Register Diagram



Sending direct items

OVERALL CLASS DIAGRAM OF SENDING ITEMS



Sending messages diagram

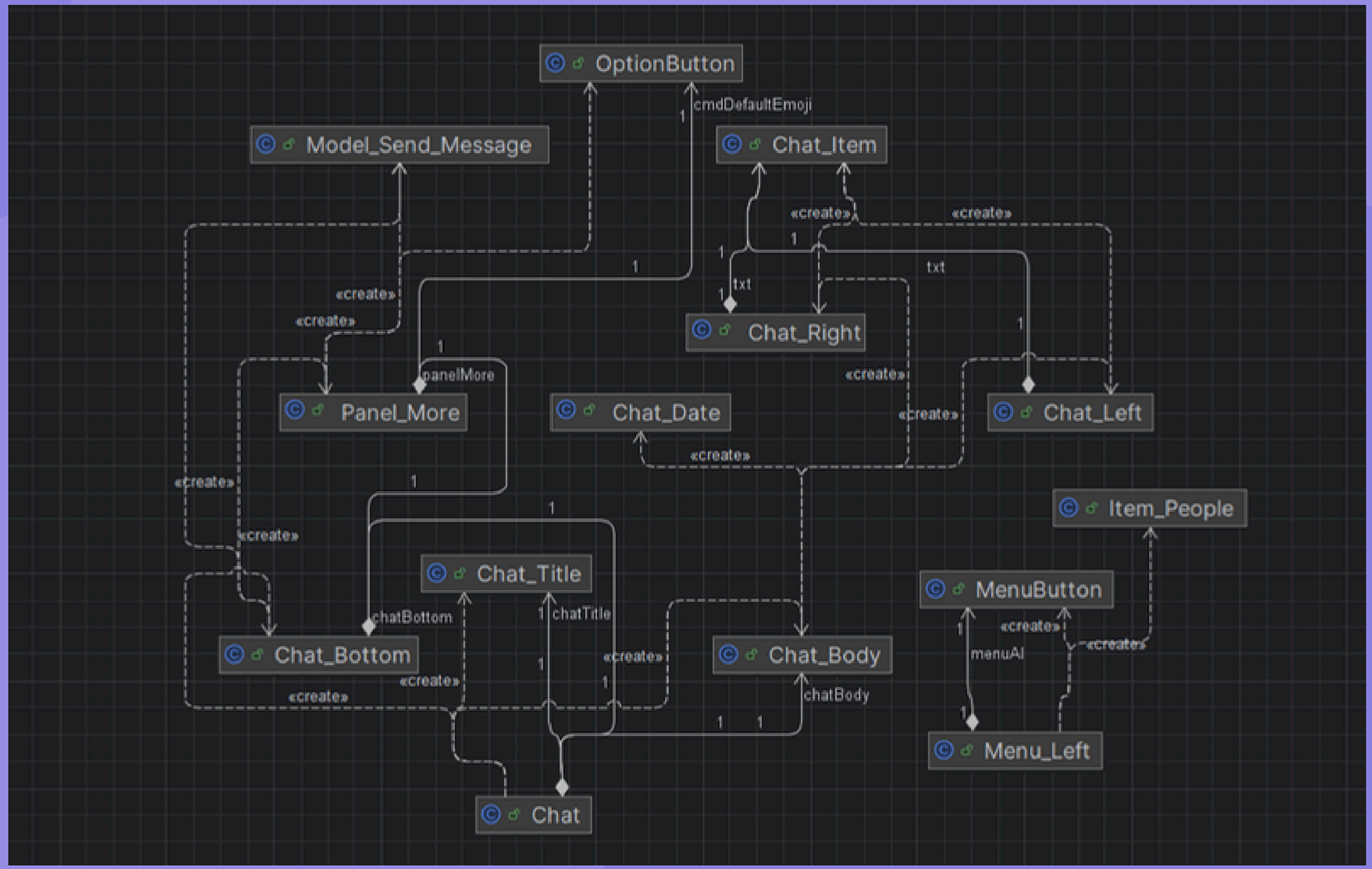


Diagram of server

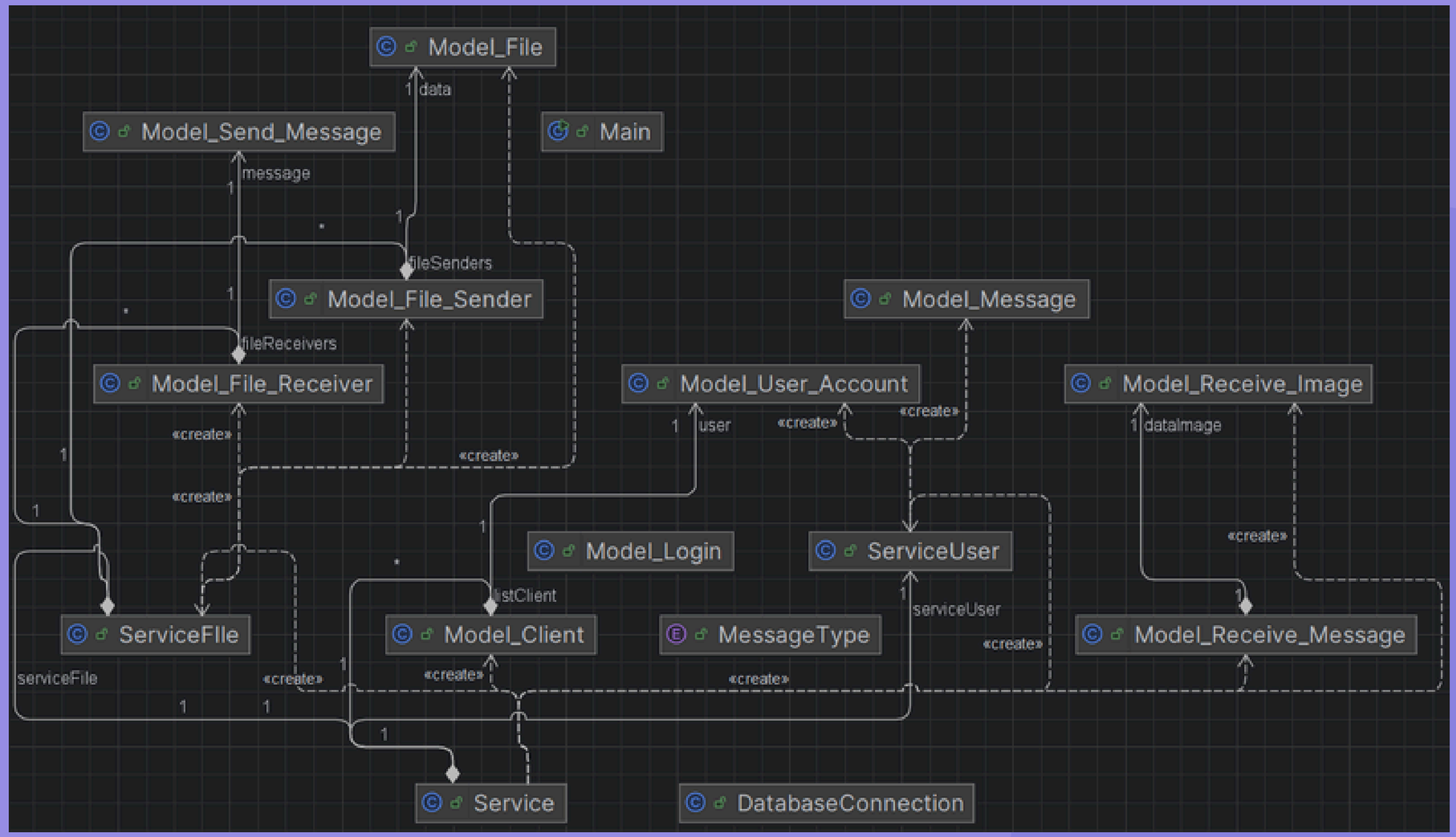


Diagram of images sharing

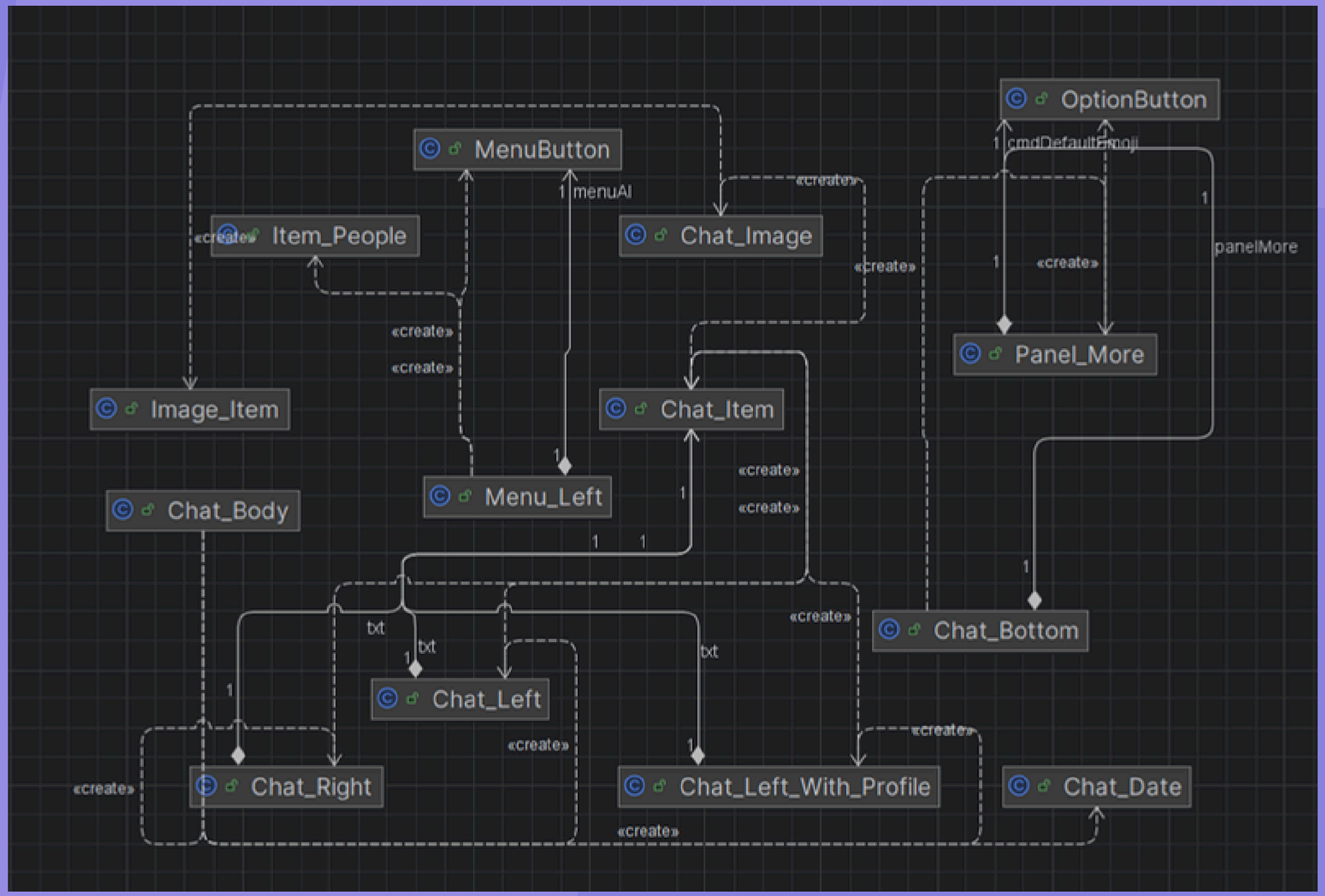
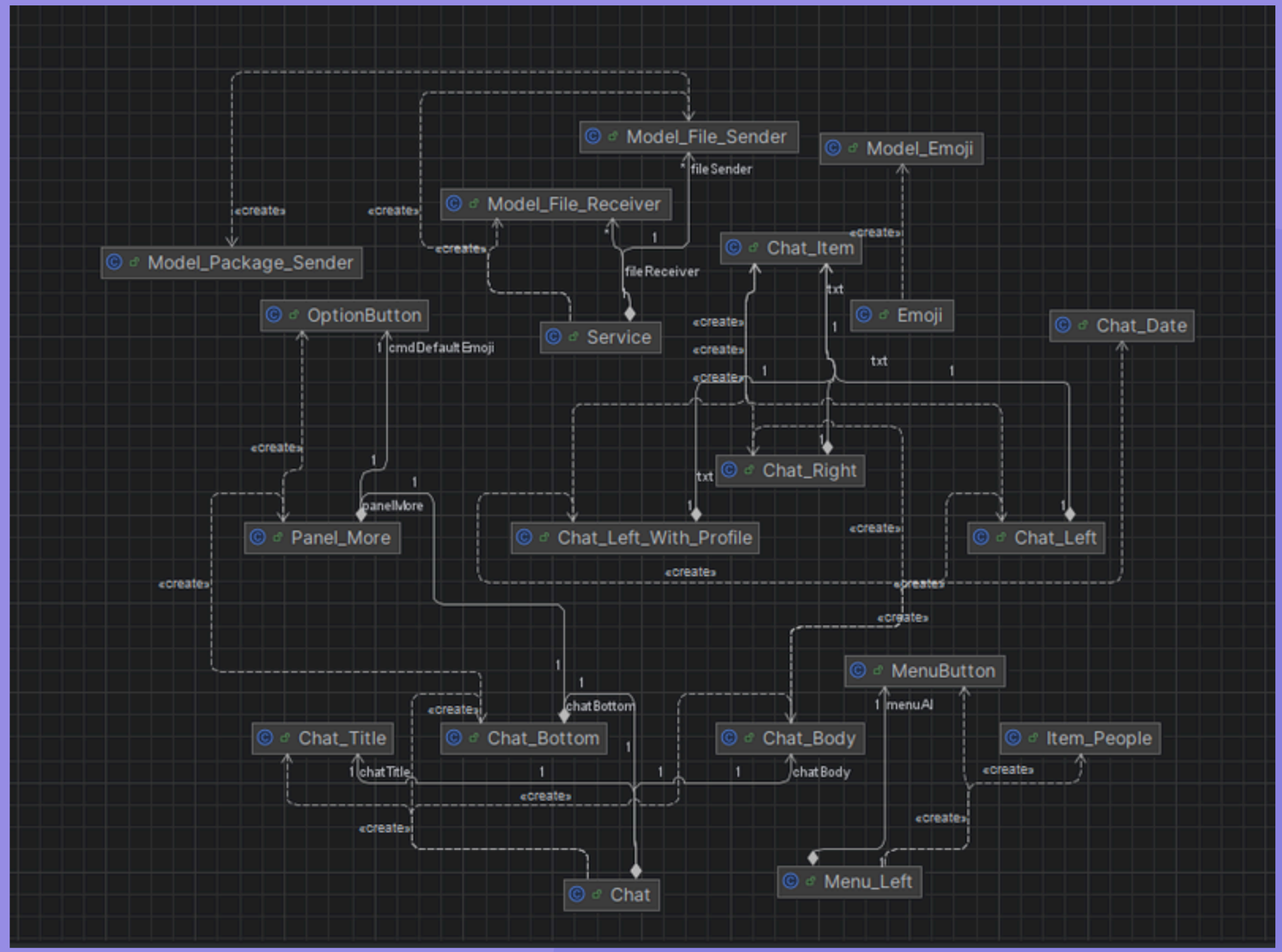


Diagram of sending icons



Four properties of OOP:

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

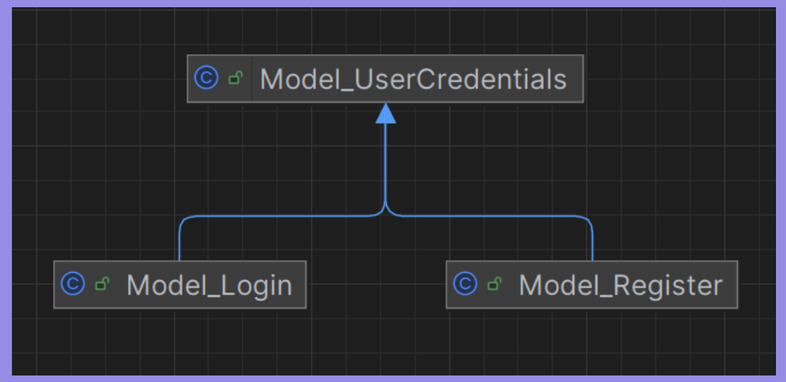
Encapsulation

- **Model_send_message is an example of OOP Encapsulation**
- **The Model_Send_Message class obtains both private and public classes, and provides getters and setters for other classes to access to its value.**
- **However, not all attributes are accessible by other classes**

```java
package com.hi.model;

import com.hi.app.MessageType;
import org.json.JSONException;
import org.json.JSONObject;

public class Model_Send_Message extends AbstractMessage {
    private int toUserID;
    private Model_File_Sender file;
    public Model_Send_Message(MessageType messageType, int fromUserID, int toUserID, String text) {
        super(messageType, fromUserID, text);
        this.toUserID = toUserID;
    }
    public Model_Send_Message() {
    }
    // Getters and setters
    public int getToUserID() {
        return toUserID;
    }
    public void setToUserID(int toUserID) {
        this.toUserID = toUserID;
    }
    public Model_File_Sender getFile() {
        return file;
    }
    public void setFile(Model_File_Sender file) {
        this.file = file;
    }
```

Inheritance

The package shows strong inheritance among its classes. In the package, classes Model_Login, Model_Register all extends the parent class Model_UserCredentials, therefore, demonstrating its inheritance




```java
package com.hi.model;
import org.json.JSONException;
import org.json.JSONObject;
// Represents user credentials (used in Login and Registration)
public class Model_UserCredentials {
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    private String userName;
    private String password;
    public Model_UserCredentials(String userName, String password) {
        this.userName = userName;
        this.password = password;
    }
```

```java
package com.hi.model;

public class Model_Register extends Model_UserCredentials {
    public Model_Register(String userName, String password) {
        super(userName, password);
    }

    public Model_Register() {
    }
}
```

```java
package com.hi.model;

public class Model_Login extends Model_UserCredentials {
    public Model_Login(String userName, String password) {
        super(userName, password);
    }

    public Model_Login() {
    }
}
```

Polymorphism

- Model_Receive_Message and AbstractMessage together exhibit the OOP feature of Polymorphism through method overriding and the abstract method.

- The AbstractMessage class defines an abstract method to JsonObject(). This abstract method serves as a "contract," requiring all concrete subclasses (like Model_Receive_Message) to provide their specific implementation of to JsonObject().

- The Model_Receive_Message class provides its implementation of to JsonObject(), which includes the data of received message including the dataImage (if available). This overriding of the abstract method demonstrates polymorphism, allowing different message types to handle JSON conversion in their own unique ways.

```java
package com.hi.model;

import com.hi.app.MessageType;
import org.json.JSONException;
import org.json.JSONObject;

public class Model_Receive_Message extends AbstractMessage {

    @Override
    public JSONObject toJsonObject() throws JSONException {
        JSONObject json = new JSONObject();
        json.put("messageType", messageType.getValue());
        json.put("fromUserID", fromUserID);
        json.put("text", text);
        if (dataImage != null) {
            json.put("dataImage", dataImage.toJsonObject());
        }
        return json;
    }
```

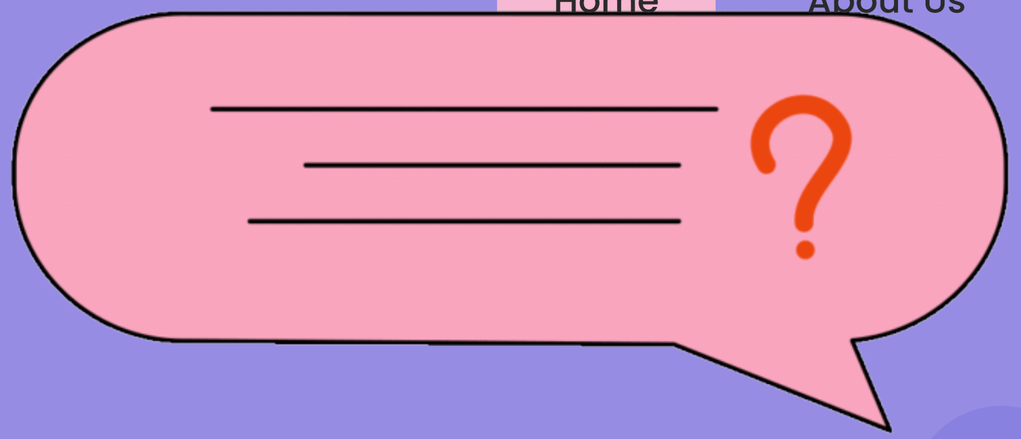
```java
package com.hi.model;
import com.hi.app.MessageType;
import org.json.JSONException;
import org.json.JSONObject;
public abstract class AbstractMessage {

    public abstract JSONObject toJsonObject() throws JSONException;
}
```

Abstraction

- The package effectively utilizes Abstraction through its AbstractMessage class.

- AbstractMessage serves as an abstract base class that defines common properties and behaviors for different types of messages.

```java
package com.hi.model;
import com.hi.app.MessageType;
import org.json.JSONException;
import org.json.JSONObject;
public abstract class AbstractMessage {
    protected MessageType messageType;
    protected int fromUserID;
    protected String text;
    // Constructor
    public AbstractMessage(MessageType messageType, int fromUserID, String text) {
        this.messageType = messageType;
        this.fromUserID = fromUserID;
        this.text = text;
    }
    public AbstractMessage() {
    }
    // Getters and setters
    public MessageType getMessageType() {
        return messageType;
    }
    public void setMessageType(MessageType messageType) {
        this.messageType = messageType;
    }
    public int getFromUserID() {
        return fromUserID;
    }
    public void setFromUserID(int fromUserID) {
        this.fromUserID = fromUserID;
    }
    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }
}
```



Thank you for listening!