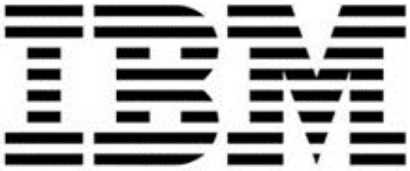




**COLLEGE CODE : 9111**

**COLLEGE NAME: SRM MADURAI COLLEGE FOR ENGINEERING AND TECHNOLOGY**

**DEPARTMENT: B.E COMPUTER SCIENCE AND ENGINEERING**

**STUDENT NM-ID:** 29693809323E58D85804E77C620A3B33
E84A706084E253582B34A98D05CA3B22
5783594E2D19B3FA18F47F4C526F12C7
1069BEA60F6B24E85854A583833EB1C6
5EAED41CE1AB1D89AC442AA0A8A642F7

**ROLL NO:** 911123104021
911123104003
911123104049
911123104035
911123104042

**DATE:**

**Completed the project named as**

**Phase_III_**

**TECHNOLOGY PROJECT NAME : CHAT APPLICATION UI**

**SUBMITTED BY,**

**NAME : C.AKSHAYA**
**H.JESIMA YOHAANA**
**S.SHREYA MERCY**
**D.PRIYA DHARSHINI**
**M.K.ROSHINI**

# CHAT APPLICATION UI- PHASE III

## 1. Project Setup

- **Initialize the project:**
  - Use a frontend framework like **React** (or another you prefer).
  - Set up project structure with folders for components, styles, and utilities.
- **Install dependencies:**
  - React, ReactDOM, any UI libraries (optional, e.g., Material-UI, Tailwind).
- **Set up version control:**
  - Initialize a **Git** repository.
  - Create `.gitignore` file to exclude `node_modules`, build files, etc.
- **Basic file scaffolding:**
  - Create main files: `App.js`, `index.js`.
  - Create UI components folder:
    e.g., `ChatWindow.js`, `MessageInput.js`.

## 2. Core Features Implementation

- **Chat window:**
  - Display messages with sender labels.
  - Scrollable message area.
- **Message input:**
  - Text input box.
  - Send button.
  - Support pressing Enter to send.
- **Message sending:**
  - User can type and send messages.
  - Display sent messages immediately in UI.
- **Simulated reply (optional for MVP):**
  - Generate simple automated replies after delay to mimic chat partner.

## 3. Data Storage (Local State / Database)

- **Local State Management:**
  - Store chat messages in React component state (`useState`).
  - Messages stored as an array of objects `{ sender: 'me' | 'friend', text: string }`.
- **Persistence (optional for MVP):**
  - Use `localStorage` to persist messages across reloads.
- **Database Integration (for later versions):**
  - Backend API with database (Firebase, REST API, etc.) to store messages.
  - Real-time messaging with WebSocket or Firebase Realtime DB.

**4. Testing Core Features**

- **Unit Tests:**
  - Test components render properly (e.g., using Jest + React Testing Library).
  - Test message sending logic updates state correctly.
- **Integration Tests:**
  - Verify that typing a message and pressing send updates the chat window.
- **Manual Testing:**
  - Check UI responsiveness on different screen sizes.
  - Verify input edge cases: empty messages, very long messages.
- **Automated UI tests (optional):**
  - Use Cypress or Selenium for end-to-end testing.

**5. Version Control (GitHub)**

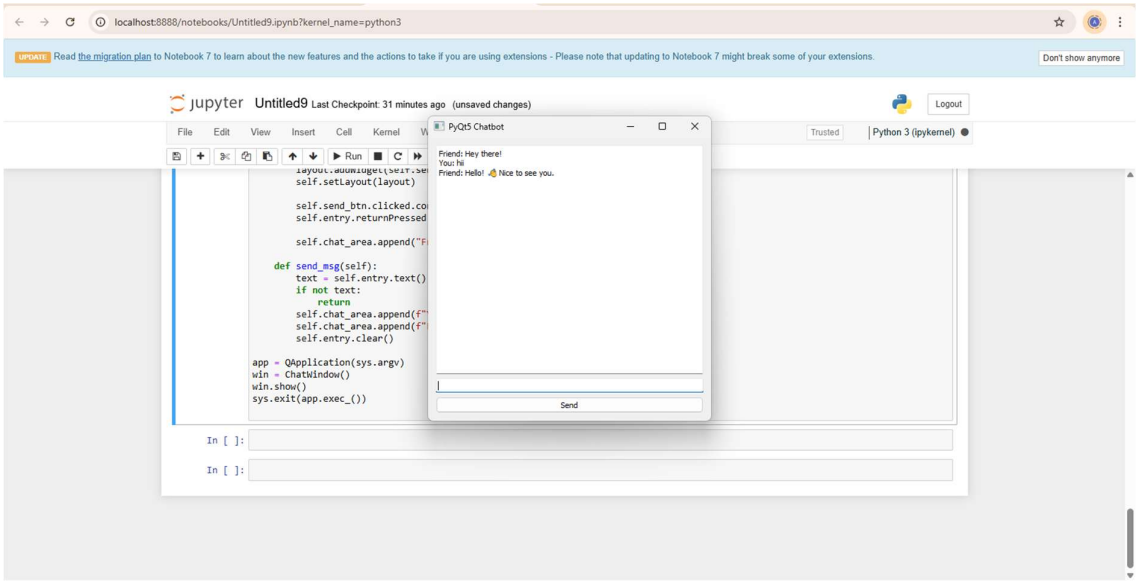
- **Initialize Git Repository:**
  - `git init` at project root.
- **Commit frequently:**
  - After project setup.
  - After core features completion.
  - After adding tests.
- **Branching Strategy:**
  - Use branches like `feature/chat-ui`, `feature/tests`.
- **Push to remote:**
  - Create repository on GitHub.
  - `git remote add origin <repo-url>`
  - `git push -u origin main`
- **Pull Requests:**
  - Use PRs to review and merge features.
- **Documentation:**
  - Maintain `README.md` with setup and usage instructions.
  - Optionally include feature roadmap.

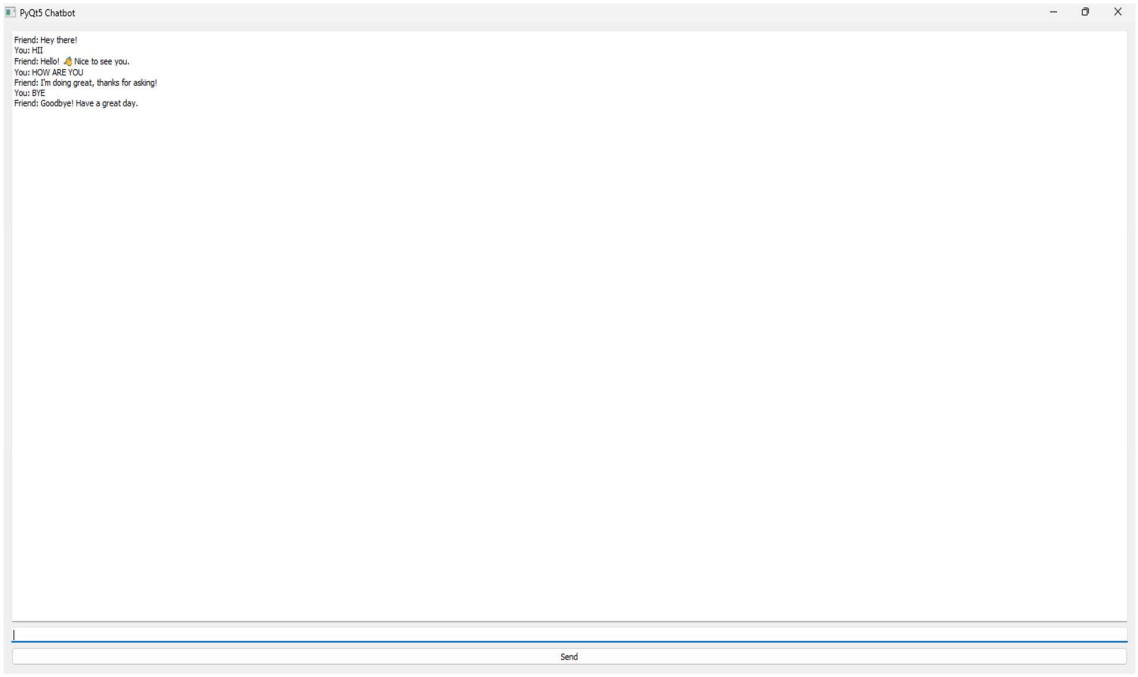
**Coding :**

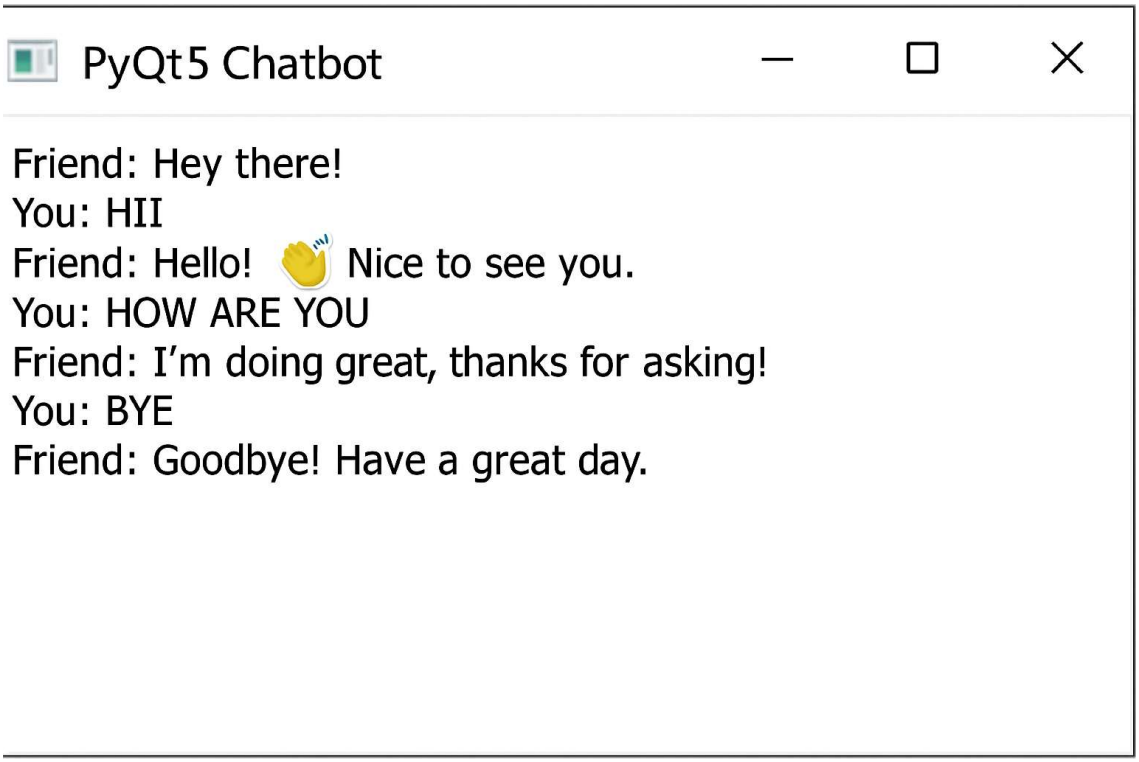
```
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QTextEdit, QLineEdit,
QPushButton
import sys

def bot_reply(msg):
    text = msg.lower()

    if any(word in text for word in ["hi", "hello", "hey"]):
        return "Hello! 👋 Nice to see you."
    if "how are you" in text:
        return "I'm doing great, thanks for asking!"
```

```
    if "your name" in text:
        return "I'm your friendly PyQt chatbot."
    if "about yourself" in text or "who are you" in text:
        return "I'm a simple app built in Python with PyQt5 — here to chat with you!"
    if "bye" in text or "goodbye" in text:
        return "Goodbye! Have a great day."
    return "Hmm… that's interesting!"

class ChatWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("PyQt5 Chatbot")
        self.resize(400, 400)

        layout = QVBoxLayout()
        self.chat_area = QTextEdit()
        self.chat_area.setReadOnly(True)
        self.entry = QLineEdit()
        self.send_btn = QPushButton("Send")

        layout.addWidget(self.chat_area)
        layout.addWidget(self.entry)
        layout.addWidget(self.send_btn)
        self.setLayout(layout)

        self.send_btn.clicked.connect(self.send_msg)
        self.entry.returnPressed.connect(self.send_msg)

        self.chat_area.append("Friend: Hey there!")

    def send_msg(self):
        text = self.entry.text().strip()
        if not text:
            return
        self.chat_area.append(f"You: {text}")
        self.chat_area.append(f"Friend: {bot_reply(text)}")
        self.entry.clear()

app = QApplication(sys.argv)
win = ChatWindow()
win.show()
sys.exit(app.exec_())
```

**Output:**


localhost:8888/notebooks/Untitled9.ipynb?kernel_name=python3
UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.
Don't show anymore
jupyter Untitled9 Last Checkpoint: 31 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel
Trusted
Python 3 (ipykernel)
Run
self.setLayout(layout)
self.send_btn.clicked.co
self.entry.returnPressed
self.chat_area.append("F
def send_msg(self):
text = self.entry.text()
if not text:
return
self.chat_area.append(f"
self.chat_area.append(f"
self.entry.clear()
app = QApplication(sys.argv)
win = ChatWindow()
win.show()
sys.exit(app.exec_())
In [ ]:
In [ ]:
PyQt5 Chatbot
Friend: Hey there!
You: hi
Friend: Hello! Nice to see you.
Send


PyQt5 Chatbot
Friend: Hey there!
You: MI
Friend: Hello! Nice to see you.
You: HOW ARE YOU
Friend: I'm doing great, thanks for asking!
You: BYE
Friend: Goodbye! Have a great day.
Send




**Explanation**

## 1 Imports

```
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QTextEdit,
QLineEdit, QPushButton
import sys
```

- `PyQt5.QtWidgets`: Contains the GUI components.
  - `QApplication`: Manages the whole Qt application.
  - `QWidget`: Base class for all UI windows.
  - `QVBoxLayout`: A vertical layout manager (arranges widgets top → bottom).
  - `QTextEdit`: A multi-line text box (for showing conversation).
  - `QLineEdit`: A single-line text box (for user input).
  - `QPushButton`: A clickable button.
- `sys`: Lets us use `sys.argv` (needed to start the Qt app).

## 2 Chatbot Reply Function

```
def bot_reply(msg):
    text = msg.lower()
```

- Defines a function `bot_reply()` to decide the chatbot's response.
- `msg.lower()` converts the user message to lowercase → easier keyword matching.

```
    if any(word in text for word in ["hi", "hello", "hey"]):
        return "Hello! 👋 Nice to see you."
```

- If the message contains **hi / hello / hey**, return a friendly greeting.

```
    if "how are you" in text:
        return "I'm doing great, thanks for asking!"
```

- Checks if `"how are you"` appears in the message → returns a custom reply.

```
    if "your name" in text:
        return "I'm your friendly PyQt chatbot."
```

- If the user asks for the chatbot's name, reply accordingly.

```
    if "about yourself" in text or "who are you" in text:
        return "I'm a simple app built in Python with PyQt5 — here to chat
with you!"
```

- If the message asks for details ("about yourself" / "who are you"), send info about the bot.

```
    if "bye" in text or "goodbye" in text:
        return "Goodbye! Have a great day."
```

- Detects goodbyes and responds politely.

```
    return "Hmm… that's interesting!"
```

- Fallback response if none of the above keywords match.

## 3 Chat Window Class

```
class ChatWindow(QWidget):
```

- Defines a new class `ChatWindow` that inherits from `QWidget` (a window).

### `__init__` Method (Constructor)

```
def __init__(self):
    super().__init__()
```

- `__init__`: Runs when a `ChatWindow` object is created.
- `super().__init__()`: Calls the `QWidget` constructor to set up the window.

```
self.setWindowTitle("PyQt5 Chatbot")
self.resize(400, 400)
```

- Sets the window's title and size (400×400 pixels).

#### Create Layout and Widgets

```
layout = QVBoxLayout()
```

- A **vertical box layout** that stacks widgets vertically.

```
self.chat_area = QTextEdit()
self.chat_area.setReadOnly(True)
```

- `QTextEdit`: Where the conversation appears.
- `setReadOnly(True)`: Prevents the user from typing inside it.

```
self.entry = QLineEdit()
self.send_btn = QPushButton("Send")
```

- `QLineEdit`: Input box for typing messages.
- `QPushButton`: Button labeled "Send."

#### Add widgets to layout

```
layout.addWidget(self.chat_area)
layout.addWidget(self.entry)
layout.addWidget(self.send_btn)
self.setLayout(layout)
```

- Adds the widgets to the layout (chat → entry → button).
- Attaches the layout to the window.

#### Connect Signals (Events)

```
self.send_btn.clicked.connect(self.send_msg)
self.entry.returnPressed.connect(self.send_msg)
```

- When the **Send** button is clicked → call `self.send_msg`.
- When the **Enter** key is pressed in the entry box → also call `self.send_msg`.

**Start Conversation**

```
self.chat_area.append("Friend: Hey there!")
```

- Adds the first line to the chat area.

### `send_msg` Method

```
def send_msg(self):
    text = self.entry.text().strip()
    if not text:
        return
```

- Gets the user's text from the entry box and trims whitespace.
- If the box is empty → stop.

```
    self.chat_area.append(f"You: {text}")
```

- Shows the user's message in the chat area.

```
    self.chat_area.append(f"Friend: {bot_reply(text)}")
```

- Calls `bot_reply(text)` to generate a response → adds it to the chat area.

```
    self.entry.clear()
```

- Clears the input box after sending.

## 4 Run the Application

```
app = QApplication(sys.argv)
```

- Creates a Qt application object (required by PyQt to manage events).

```
win = ChatWindow()
win.show()
```

- Creates a `ChatWindow` instance and shows it.

```
sys.exit(app.exec_())
```

- Starts the Qt event loop (waits for clicks, typing, etc.).
- `sys.exit` ensures a clean exit when you close the window.

## Summary of Flow

1. User types a message → presses **Send** or Enter.
2. `send_msg()` gets the text → appends it → calls `bot_reply()`.
3. Reply is added to the chat area → entry cleared.
4. The app continues running until the window is closed.