Criterion C: Development

Introduction

Various programming techniques, software tools, and user interface considerations were used in the development of the Telegram bot in order to meet the defined success criteria. This section describes the techniques used, how they were implemented, and why they were chosen.

Techniques Used in Development

1. **Algorithmic thinking**
    a. Finite State Machine (FSM) for User Interaction.

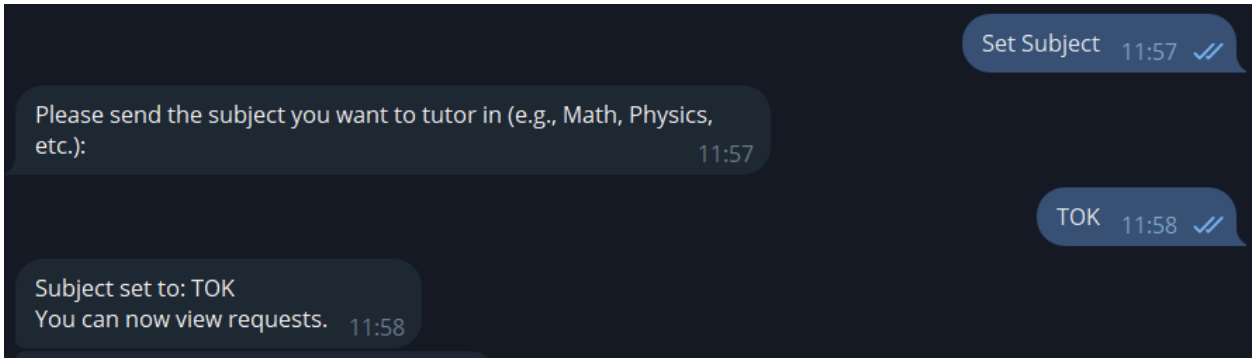
    State driven workflow management. For example:

    - The bot enters a state after the user clicks “Become a tutor” expecting a subject input.
    - The student waits upon sending a request for a tutor’s response.

```python
@bot.message_handler(func=lambda message: message.text == "Set Subject")
def set_subject(message):
    bot.send_message(message.chat.id, text: "Please send the subject you want to tutor in (e.g., Math, Physics, etc.):")
    bot.register_next_step_handler(message, receive_subject)
```

FSMs enforce logical train of thought (like setting a subject before seeing requests)

This meets the success criterion for usability that minimizes input mistakes.



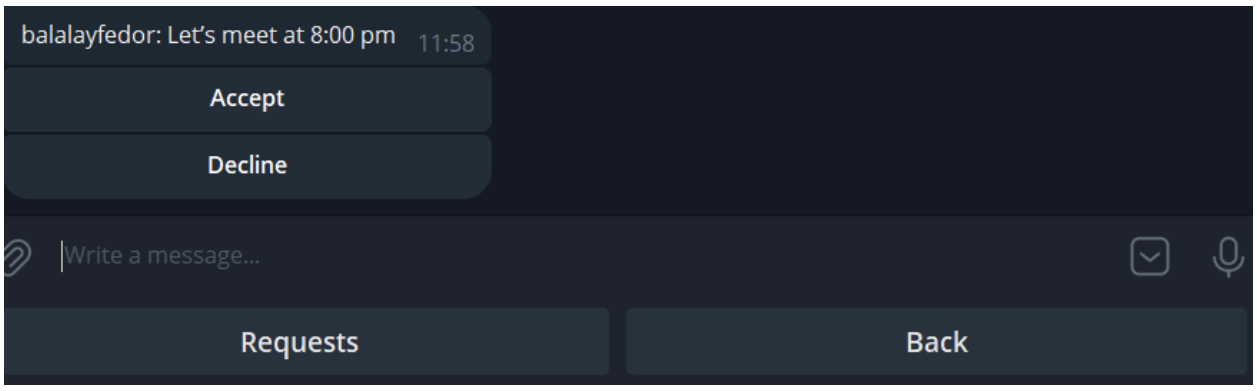
b. Callback Query Handling

Leveraging Telegram's inline keyboards and callback data, the bot manages real-time interactions such as accepting/rejecting requests.

```
markup = types.InlineKeyboardMarkup()
markup.add(types.InlineKeyboardButton( text: 'Accept', callback_data=f'accept:{order_id}'))
markup.add(types.InlineKeyboardButton( text: 'Decline', callback_data=f'decline:{order_id}'))

bot.send_message(recipient_id, text: f"New order from {sender_name}:\n{message.text}", reply_markup=markup)
bot.send_message(message.chat.id, text: "Your order has been sent.")
```

Callback queries are processed asynchronously, which is essential for managing multiple users, simultaneously. It proves the matching successful criterion in an efficient manner.



2. Data Structures

a. Relational Database (SQLite):

Structure:

- Users Table: User roles (mode), subjects and Telegram IDs.
- Orders Table: Tracks request statuses (Pending, Accepted, Declined).

```
def init_db():  1 usage
    conn = sqlite3.connect('tutors.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS users (
                      id INTEGER PRIMARY KEY,
                      username TEXT,
                      mode TEXT,
                      subject TEXT)''')
    cursor.execute('''CREATE TABLE IF NOT EXISTS orders (
                      id INTEGER PRIMARY KEY AUTOINCREMENT,
                      sender_id INTEGER,
                      sender_name TEXT,
                      recipient_id INTEGER,
                      message TEXT,
                      status TEXT)''')
    conn.commit()
    conn.close()
```

SQLite is a small, fast, and self-contained serverless relational database management system. The relational structure ensures its data integrity, fulfilling the dynamic updates success criterion.

b. Inline Keyboards as Interactive Menus.

```
@bot.message_handler(commands=['start'])  1 usage
def start_command(message):
    user_id = message.from_user.id
    reset_user_mode(user_id)
    markup = ReplyKeyboardMarkup(resize_keyboard=True)
    markup.add( *args: KeyboardButton("Find a tutor"), KeyboardButton("Become a tutor"))
    bot.send_message(message.chat.id,  text: "Welcome! Choose what you want:", reply_markup=markup)
```

Using predefined buttons decreases typos and directs users, increasing accessibility and error handling.

3. Software Tools

a. pyTelegramBotAPI Library

```
import telebot
from telebot import types
from telebot.types import ReplyKeyboardMarkup, KeyboardButton, InlineKeyboardMarkup, InlineKeyboardButton
TOKEN = "7805831024:AAGLXM2dinEKiV1jTBbzgU8sRWYiahO2MjQ"
bot = telebot.TeleBot(TOKEN)
```

The library makes it easier to develop Telegram bots via integrated decorators (e.g., @bot) to enable rapid prototyping. This decision fits with the usability success criterion.

b. SQLite3 for Database Management

```
    conn = sqlite3.connect('tutors.db')
    cursor = conn.cursor()
    cursor.execute( sql: "UPDATE users SET mode = NULL, subject = NULL WHERE id = ?", parameters: (user_id,))
    conn.commit()
    conn.close()

    bot.clear_step_handler_by_chat_id(user_id)
```

At a school-level project, you do not need to set up a server like SQLite. It supports 24/7 availability as a means to provide minimal dependencies.

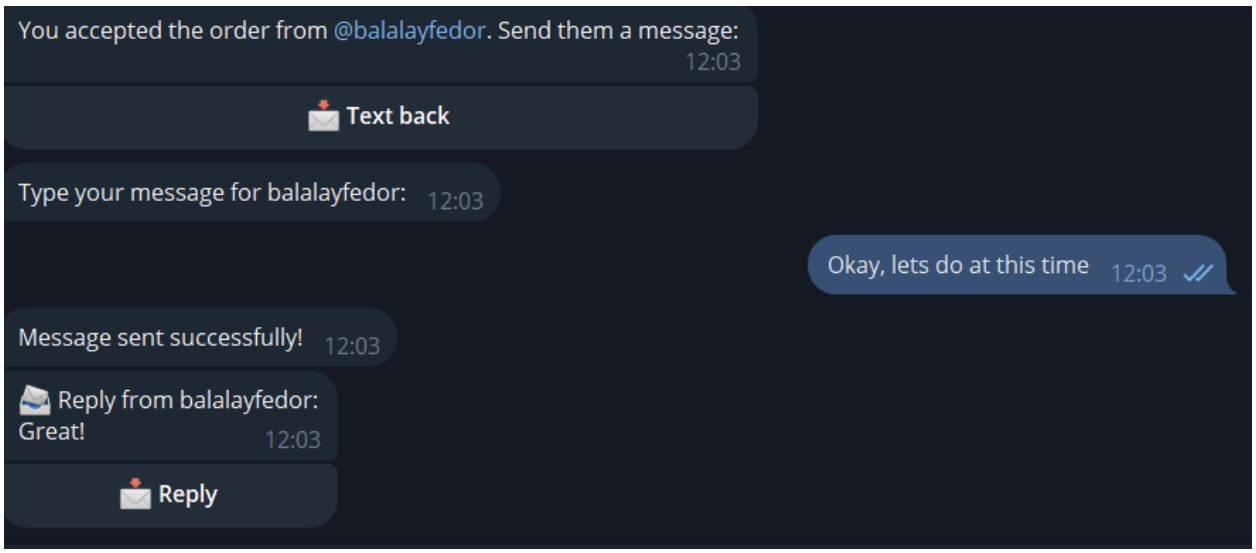
4. User Interface Design

a. Bidirectional Messaging System

```
@bot.callback_query_handler(func=lambda call: call.data.startswith("textback"))
def initiate_textback(call):
    _, student_id, student_name = call.data.split(':')
    tutor_id = call.from_user.id

    msg = bot.send_message(tutor_id, text: f"Type your message for {student_name}:")
    bot.register_next_step_handler(msg, send_to_student, student_id=int(student_id))
```

This allows for direct communication between the tutor and student, satisfying the efficient matching criterion.



5. Error Handling Techniques

a. Graceful Failure for Blocked Users

```
def send_to_student(message, student_id):  1 usage
    try:
        tutor_name = message.from_user.username or f"Tutor {message.from_user.id}"
        markup = InlineKeyboardMarkup()
        markup.add(InlineKeyboardButton( text: '📩 Reply', callback_data=f'reply:{message.from_user.id}:{tutor_name}'))

        bot.send_message(student_id,
                         text: f"📩 Message from {tutor_name}:\n{message.text}",
                         reply_markup=markup)
        bot.send_message(message.chat.id,  text: "Message sent successfully!")
    except Exception as e:
        print(f"Error sending message: {e}")
        bot.send_message(message.chat.id,  text: "Failed to send message. The student might have blocked the bot.")
```

Avoids crashing and reports address to troubled tutors, provides reliability (success criteria).

Conclusion

The techniques above were selected for their alignment with the project's success criteria:

- Algorithmic thinking (FSMs) and data structures (SQLite) ensure efficient matching.
- Software tools (pyTelegramBotAPI) and UI design (inline keyboards) prioritize usability.
- Error handling guarantees robustness and 24/7 accessibility.

By combining these techniques, the bot delivers a scalable, user-friendly solution for peer tutoring.