University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

BACHELOR'S THESIS

Study programme/specialisation: Bachelor Datateknologi	Spring, 2021 Open
Author: Fredrik Seim Berge	
Programme coordinator: Supervisor(s): Krisztian Balog	
Title of bachelor's thesis: Creating a Move Recommendation Chatbot for the Facebook Messenger Platform	
Credits:	
Keywords:	Number of pages: 34 + supplemental material/other: ..... Stavanger, 15/05/2021 date/year

Abstract

The main goal of this thesis is to create a Facebook Messenger bot for the IAI MovieBot recommending system. The MovieBot is adapted to run on Messenger and Telegram simultaneously. Additionally, an adapted version of the Telegram is added to make the user experience of the two platforms as close as possible. The three versions are individually evaluated by test subjects to assess the success of implementation.

Contents

Chapter 1

Introduction

1.1 Background

Conversational systems are becoming an increasingly utilized technology for businesses and organizations worldwide [2]. Conversations that would otherwise require a human employee can through the use of conversational systems now be completely automated. Aside from the obvious advantages of saved costs and greater availability, automated systems also have other benefits over their human counterparts. The processing of complex queries by customers and retrieval of relevant information is one example where a machine can do the job much more efficiently than a human. The goal of any conversational system is to create a functional interface between human and machine.

The rise of conversational systems in recent years is driven in large part by development and progress in the underlying technologies, such as Natural Language Processing (NLP), Natural Language Generation (NLG) and Artificial Intelligence (AI). Natural Language Processing is a field that uses linguistics, computer science and artificial intelligence to make human language understandable for computers. Together with machine learning and AI, it is possible to create systems that can understand the intentions of users, and sometimes even predict what users want in the future. Natural Language Understanding (NLU) is a subset of NLP that is concerned with the intended meaning of text or speech.

Although conversational systems are known by most through the use of customer service chat bots, the technology can be applied to countless other settings, of which there are many examples. The Azure Health Bot is a service provided by Microsoft that helps healthcare organizations build conversational healthcare bots [1]. It comes with natural language capabilities, AI and medical information, with the option for customers to customize the bot to their own needs. The Rapid Response Virtual Agent [7] program was

launched by Google in 2020 in response to the pandemic. The program makes it possible for governments and organisations to quickly launch a chat bot that answer questions about COVID-19.

1.2 IAI MovieBot

The IAI MovieBot is a movie recommender system that uses a task-oriented dialogue to help users find a movie according to their preferences [11]. The system uses Natural Language Understanding and Natural Language Generation in a turn-based conversation with the user.

The conversation starts by asking the user for a preferred genre followed by a keyword to help narrow down the search space. It then recommends a movie along with some information and a selection of buttons. From here the user can select whether they want to accept, reject or show interest in the recommendation.

The system is currently available as a bot on Telegram, a messaging application with over 500 million active users. Facebook Messenger, also known as Messenger, is a messaging platform with over 1.3 billion active users, while Facebook as a social media platform has over 2.7 billion users [8]. In 2016 Facebook launched the Messenger Platform [10], a bot API for developers. According to Facebook, the Messenger Platform is aimed primarily at businesses for the automation of customer service. The Bot API offers a range of different templates and other UI components that makes it suitable for other types of conversational systems as well. By making the IAI MovieBot available on Messenger it will be able to reach many times the number of users currently on Telegram.

The goal of this thesis is to adapt the IAI MovieBot recommender system to Messenger while maintaining the same user experience on both platforms. If there are any limitations in the Messenger Platform that makes this difficult, then some changes may have to be made on the Telegram side. The solution should contain a multi-modal controller that can service both platforms simultaneously in one server instance. The final step is to get the Messenger application approved by Facebook, so that it can be published and made available to the public.

1.3 Goals

- **Adapt MovieBot to Messenger**. The MovieBot system is currently running as a bot on Telegram. The objective is to make the system run on Messenger by creating a separate Messenger controller.
- **Create multi-modal controller for messenger and telegram.** The Telegram version of the MovieBot is currently using polling, not a server and webhook setup. The goal is to run both Telegram and Messenger in a single server instance with the use of webhooks.
- **Get the app approved by Facebook**. When developing a Messenger application, it has to be approved by Facebook before it can be published and made available to users. The approval process ensures that the application is functioning as intended and that is it following Facebook's guidelines. As a part of the approval process a privacy policy also has to be created and connected to the application.
- **Create same user experience for messenger and telegram.** The Messenger Platform and Telegram differ in various ways in terms of functionality and UI components. The objective is to create the Messenger version as identical to the Telegram version as possible, and make changes to the Telegram version if necessary.

1.4 Contributions

My thesis partner disappeared about four weeks before the deadline. It turned out later that he had withdrawn from the thesis. For the first few months we worked together mainly on the design decisions for the Messenger implementation and we both explored and tested the different components available in the Messenger Platform. In terms of actual contributions to the project, he made the contents of the privacy policy and assisted in creating a method for retrieving information from a database.

Task	**Time Allocated**	**Fredrik**	**Niklas**
Design decisions	10%	60%	40%
Server code	15%	100%	0%
Messenger code	30%	90%	10%
Telegram code	10%	100%	0%
Approval process	10%	100%	0%
Privacy policy	5%	10%	90%
Thesis	20%	100%	0%

The code in this project is based on the IAI MovieBot, available on GitHub [5]. We worked on a fork of the original project and submitted pull requests when certain implementations were completed.

1.5 Outline

Chapter 2 - Technology and Tools
Chapter 2 introduces the main technologies and tools that are used for the project. Here, the developer tools and features for Telegram and the Messenger platform are described in detail.

Chapter 3 - Approach
Chapter 3 describes the approach taken and a problem analysis that gives an overview of the problems that has to be solved and proposes ways to solve them. The rest of the chapter is dedicated to the implementation of the solutions.

Chapter 4 - Results
In chapter 4 the different versions of the MovieBot are evaluated by and analyzed.

Chapter 5 - Discussions
In chapter 5 the overall success of the Messenger implementation is discussed and evaluated. Alternative solutions that could have been implemented are also discussed.

Chapter 2

Technology and tools

2.1 Telegram

Telegram is a free to use cloud-based messaging platform, available on android, iOS, macOS, Windows and linux. All that is required to sign up is a phone number.

Its development is motivated by belief in the right to privacy from government and corporations, while simultaneously trying to be the best messaging platform available. They are currently not monetizing the platform in any way, but there are plans to implement some limited monetization in the future to keep up with the rising costs. There are currently over 500 million active users on Telegram.

Telegram offers group chat, voice calls, group voice calls, video calls, channels and secret chat. Group chat can have up to 200,000 members, while channels are meant for broadcasting to a large number of people and can have unlimited members. Voice calls and secret chat use end-to-end encryption, while cloud chats use client-server encryption. It also supports file sharing up to 1.5 GB.

Conversations that use end-to-end encryption are not stored on the Telegram servers, only on the devices that are part of the conversation. Secret chat has the ability to set a self-destruct timer, deleting the conversation on all devices. If one device chooses to delete a conversation, the other side is forced to do the same.

All devices connected to the same account are synced through the Telegram cloud network. All cloud-based conversations are available on all devices, including shared files.

2.2 Telegram Bots

Bots in Telegram [9] are special accounts that can be set up by users. Bot accounts look identical to user accounts, and anyone can start a conversation with a bot. Users can find the bot they are looking for through the search function. Bots can not initiate conversations with users.

Some bots have the ability to function inline, meaning they can be accessed directly from the input filed of any chat window by using the '@' symbol followed by the bot name. A response from the inline bot will be posted in the current chat window, making it possible to use bots in conversation with other users.

2.2.1 Communication

Messages and commands sent to the bot by the user are passed to the Telegram servers. The servers handle the communication with the Telegram API. Telegram and the application server communicates using the Telegram Bot API, a HTTP-based interface. The access token is a part of this communication to protect the bot from any unwanted access. The Bot API sends messages by POST requests to the application. The messages contain information about the user, such as user id, and a payload. Below is an example of a route for incoming Telegram requests.

```
@app.route('/{}'.format(telegram_token), methods=['POST'])
def respond():
    update = telegram.Update.de_json(request.get_json(force=True), bot)
    text = update.message.text.encode('utf-8').decode()
```

2.2.2 Bot setup

The creation of a new bot is done using one of Telegram's offical bots, BotFather [2.1]. After following the instructions provided by BotFather, the user receives an access token that will be used in communication with Telegram Bot API. When the bot has been created it will be public and available to other users.

2.2.3 Conversation and UI Components

Users interact with the bot through the chat window, just like they would with other users. The conversation is initiated by the user tapping the start button.

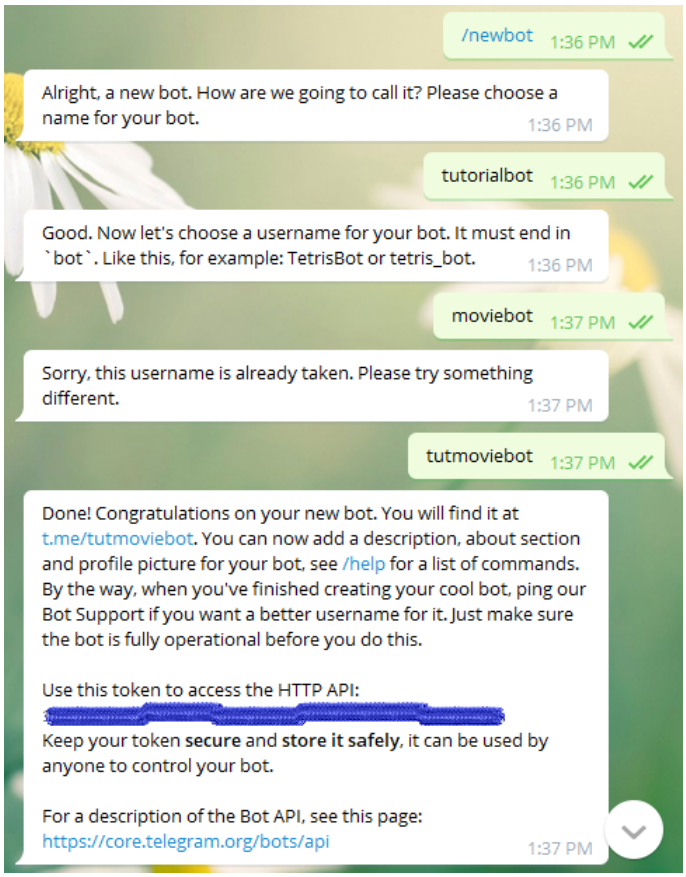

Figure 2.1: Botfather

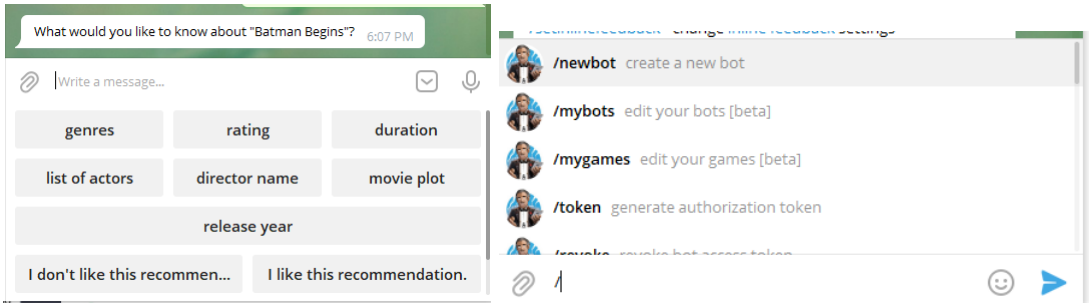

Figure 2.2: Keyboard with buttons and commands

A bot conversation can be entirely based on user text input, but Telegram also offers some different UI components that can be used to create a more streamlined user experience. Some of these include Bot Payments, HTML5-based games, keyboards and commands. For the MovieBot, only keyboards and commands are used.

Keyboards are situated below the text input field and contains a list of buttons. When a button is tapped, the text displayed on the button will be posted in the chat window and handled the same way as user text input.

Commands can have up to 32 characters and start with a '/'. Commands are posted in the chat window and passed to the bot. When entering '/' in the input field, a list of available commands is shown.

2.3 The Messenger Platform

The Messenger Platform is a tool for developers to create Messenger applications. The platform offers a range of different features for building the desired user experience.

2.3.1 Communication

To connect an application to Facebook, the Facebook server sends webhooks to the URL of the business server where the application is hosted. Then using the Send API the application can respond to a user on Messenger [6]. Every time a user starts a conversation with a Messenger bot, a page-scoped ID (PSID) is assigned to them. The PSID is unique to the Facebook page the bot is connected to, meaning that a user has different PSIDs for each bot they are in conversation with.

The Send API is the primary integration point with the bot and the Messenger Platform.

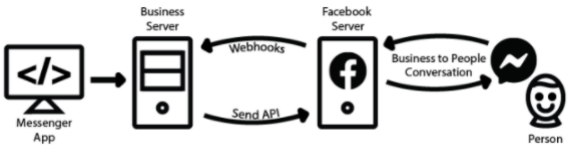

Figure 2.3: Facebook API Communication

It is through this API the bot can send conversation components.

The webhook is a single HTTPS endpoint which accepts POST requests, and is where the bot processes and responds to incoming webhook events. Events are sent from the Messenger Platform whenever an action occurs in a conversation with your bot.

```
@app.route('/', methods=['GET', 'POST'])
def receive_message():
    if request.method == 'GET':
        token_sent = request.args.get("hub.verify_token")
        return verify_fb_token(token_sent)
    else:
        output = request.get_json()
        action(output)
        return "Message Processed"
```

The Messenger Platform sends webhook events as POST requests to the webhook. There are multiple types of webhook types, but for the purpose of this project only the types message and postback are relevant.

A message event occurs when a user sends a text message to the bot, and a postback event occurs when a button of type postback is pressed. Shown below are examples of a message and a postback messages received from the Facebook server.

```
{'object': 'page', 'entry': [{'id': '105900111496592', 'time': 1620903054401,
'messaging': [{'sender': {'id': '3815517311849039'}, 'recipient': {'id': '105900111496592'},
'timestamp': 1620903054181,
'message': {'mid': 'm_r3h6v_AA9mdN8-MdifOe1I3xaOF52Cw83Go4QURcMlK53Ni_kEvOCDjh03f7cqol6JTXzqg
'text': 'batman'}}]}]}
```

{'object': 'page', 'entry': [{'id': '105900111496592', 'time': 1620903057290,
'messaging': [{'sender': {'id': '3815517311849039'}, 'recipient': {'id': '1059001114965
'timestamp': 1620903057212,
'postback': {'title': 'Seen it', 'payload': 'I have already watched it.'}}]}]}

The webhook object contains the PSID of the user and the recipient id of the page, along with the webhook type and its contents.

For some webhook types, permission has to be granted by Facebook through an application process. Other types are currently restricted in some regions due to the new GDPR regulations.

2.3.2 Development

To get started developing on the Messenger platform one will need a Messenger Platform account, a Facebook page, Facebook application and a webhook URL.

2.3.3 App Setup

The first step is to set up a Facebook page. This is where the user will go to start a conversation with the Messenger bot. All that is needed to create a page is a regular Facebook account. The name chosen for the page is the name that will displayed on the bot. Then create the Messenger application in the Messenger Platform account and connect it to the Facebook page.

In order to communicate with the Facebook server API, the webhook URL has to be hosted somewhere. This can be achieved using flask and a port forwarding service like ngrok. To tell the facebook server where to send its webhooks, the URL needs to be added to Callback URL in the Messenger Platform settings 2.4.

A verify token is also added to verify that the webhook is authentic. The verify token is a string chosen by the developer and will be part of the webhook code. The token works as a password and has to be entered together with the callback URL. After pushing "Verify and Save" the server will verify the token and the webhook setup will be complete.

When a message is sent to the Send API from the server, a page access token is appended to the end of the POST request. This access token is unique to the application and is

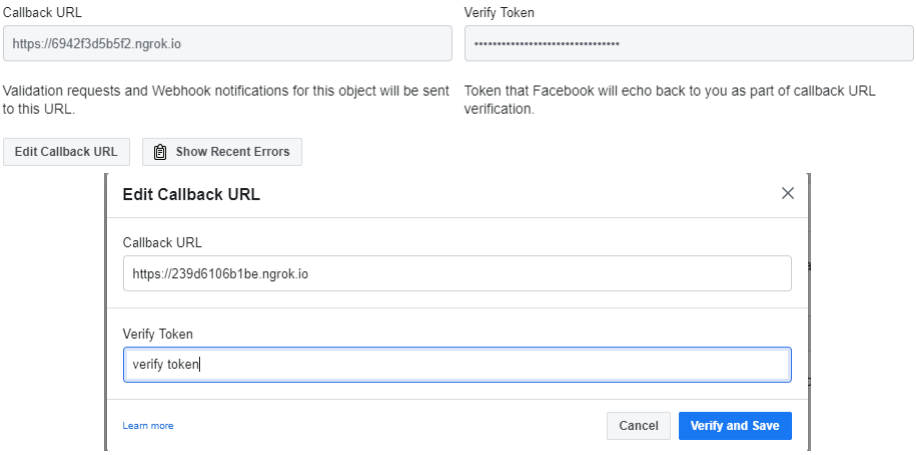

Figure 2.4: Verify Token and Callback URL

generated in the Messenger Platform settings. Anyone with access to this token could potentially impersonate the application, so it should be kept secret. If the token is compromised it is possible to generate a new token.

2.3.4 Conversation Components

The Messenger Platform includes a list of different UI components that helps developers tailor the user experience to their needs. Some of these components include attachments, quick replies, message templates, button template, sender actions, greeting text, get started button and persistent menu.

The button template contains a title and can hold up to three custom buttons listed vertically. When a button is pressed the button text is posted in the chat window. Postback buttons have different slots for payload and title text, so the payload that is passed to the API can be different than the text posted in the chat window.

Quick replies can contain up to 13 buttons that are posted as a horizontal vertical list. If the list overflows the window width, the list becomes scrollable. Quick replies can only contain postback buttons, but they also have the option to add an image to the buttons 2.5.

The persistent menu is a menu accessible through a user interface button in the bottom right corner of the chat window 2.6. The menu contains a list of up to twenty unchangeable postback or web URL buttons and it is always available to the user. If required, it

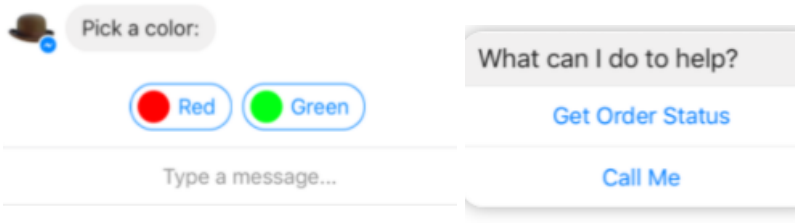

Figure 2.5: Quickreply and Buttons template.

is possible to disable user input when creating the persistent menu, making it the only way for a user to interact with the Messenger bot.

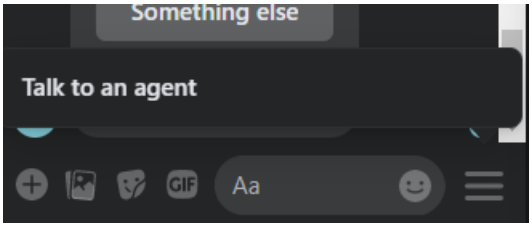

Figure 2.6: The persistent menu.

The get started button is displayed in the bot's welcome screen, before the conversation is started. It includes the title of the button, which will change language dependent on where the user is, and a payload that can be customized. The payload will usually be a command that starts the conversation.

The greeting text appears in the Messenger bot's welcome screen, above the get started button. The greeting can be customized in the Facebook page or sent as a POST request to the Send API like the other components.

The generic template is one of the many available message templates. It can include up to three buttons, a title, subtitle and image. The image and the title can be customized with an URL that opens in the browser or in a webview within Messenger.

Sender actions include the "mark seen" and "typing on" messages. Typing on displays a chat bubble until the bot sends a new message to the Send API, or after a certain time limit. Sender actions are typically sent after receiving a message from a user as a way of informing the user that the message has been received and is being processed.

2.3.5 Send API

The Send API is the endpoint for the requests sent by the business server. It is used for sending messages and conversation components to users, such as text, buttons and templates.

A message is sent as a POST request in JSON format, containing the page access token and a URI specific for the message type. Other requirements of the request body are dependent upon the message type, but all messages sent to a user require a PSID, to make sure the correct user receives the message. Messages such as greeting text and the get started button do not require a PSID, as they are not user specific, and are typically only sent once when the server is started.

The structure and contents of the sent JSON object tells the Send API what type of message it is receiving and what to do with it. It then posts the result in the chat window of the user specified by the PSID.

2.3.6 Approval

In order for the application to get out of developmental mode and be made available to users, it has to be approved by Facebook. This is done through an application process where Facebook staff test the application to make sure it functions correctly and follows the Facebook guidelines. Before the approval process can be started the owner of the Facebook account has to verify their identity. A URL linking to a privacy policy also has to be provided.

The Messenger Platform provides some submission guidelines to help with the approval process. The application has to available for external testing by Facebook, and instructions should be provided for how to test it. They also require a description of how the application is intended to be used, together with a screen recording of how it functions.

Chapter 3

Approach

3.1 IAI MovieBot

3.1.1 Dialogue Flow

When new users start a conversation with the bot, they are greeted with an introduction and a set of instructions. Pressing start initiates the conversation, and the bot responds with its first inquiry. At this stage the user can input a set preferences and receive a movie recommendation immediately. If the movie contains the same value for two different slots, there is the option to remove one and get a new recommendation 3.1.

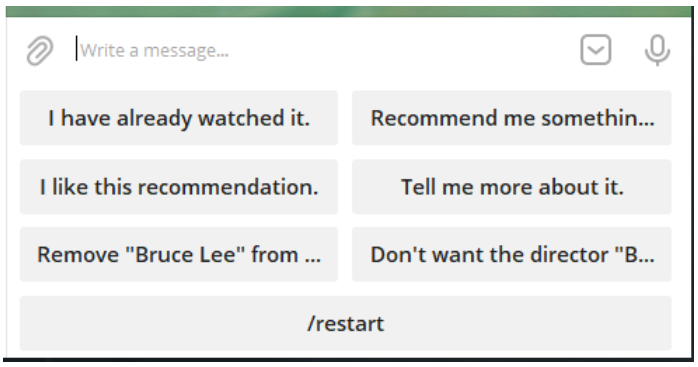

Figure 3.1: Option to remove slot.

If the conversation is restarted, the process of finding a movie starts over. In this case, the user is asked about a genre, or combination of genres. To narrow down the number of possibilities, the user is then asked to provide some keywords. The keywords can be words connected to the movie in any way, such as plot, actors and directors.

After a recommendation has been revealed, the user has the option to get more information about the movie, or get a new recommendation. If choosing to get more information, a new keyboard is presented, showing a list of details that can be expanded 3.3. When one of the buttons is pushed, it is removed from the keyboard while the rest of the options remain. The system keeps track of all the movies that has been recommended in a session to avoid recommending the same movie more than once. Pushing the movie name or image of the recommendation template opens the Imbd page for the movie.

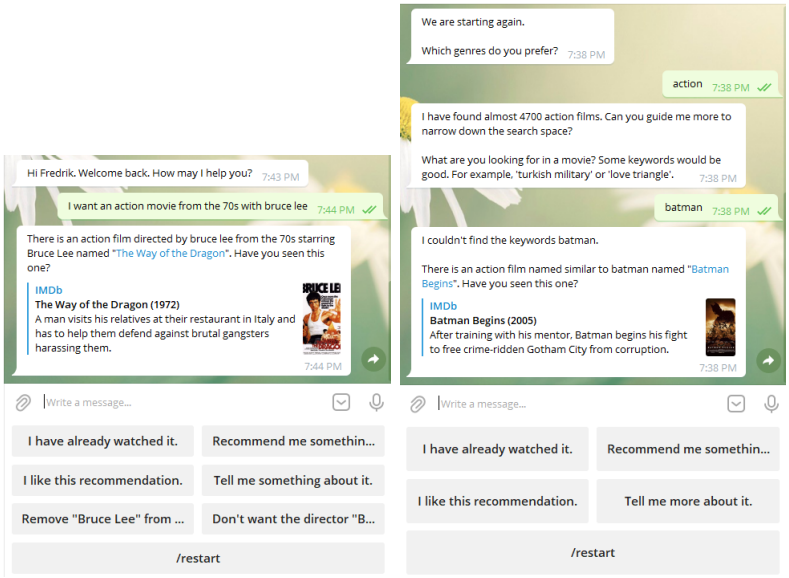

Figure 3.2: Movie recommendation

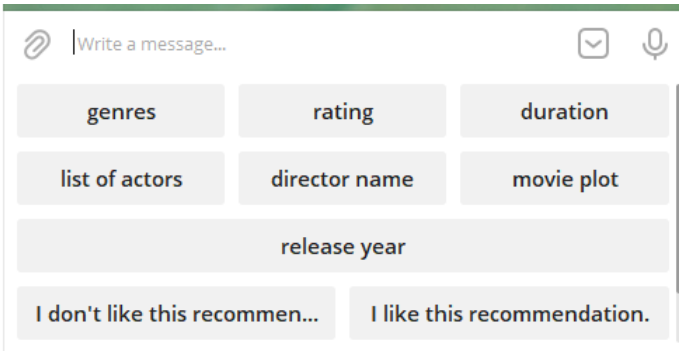

Figure 3.3: More information.

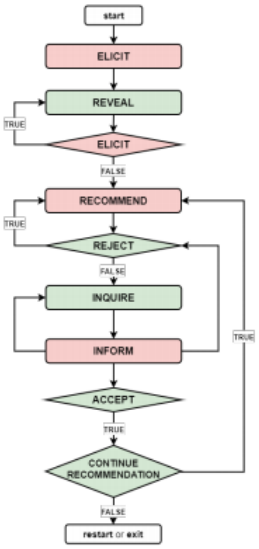

Figure 3.4: Dialogue flow.

3.1.2 Functionality

IAI MovieBot uses a task-specific dialogue flow [4] to make movie recommendations 3.4. The agent is the interface between the user and the recommender system, where user utterances are taken as inputs, and agent utterances are presented as outputs.

The IAI MovieBot is currently available as a bot on Telegram. Here, agent utterances are presented as text or buttons, dependent on where it is in the dialogue. In the same way, user inputs can be text or the return of a pushed button.

When the agent utterance output contains a list of options, the options are presented as a keyboard located below the text input field 3.5. A press of a keyboard button returns the button text as user input in the chat window, which is then passed as a user utterance to the agent. The system takes the input, generates a response and returns the response as an agent utterance.

The Telegram bot is not using a webhook and a server to receive and send updates, instead it uses polling. With polling, the application periodically connects to the Telegram servers to check for updates.

Movie recommendations are presented as a markdown of the Imbd page of the movie and

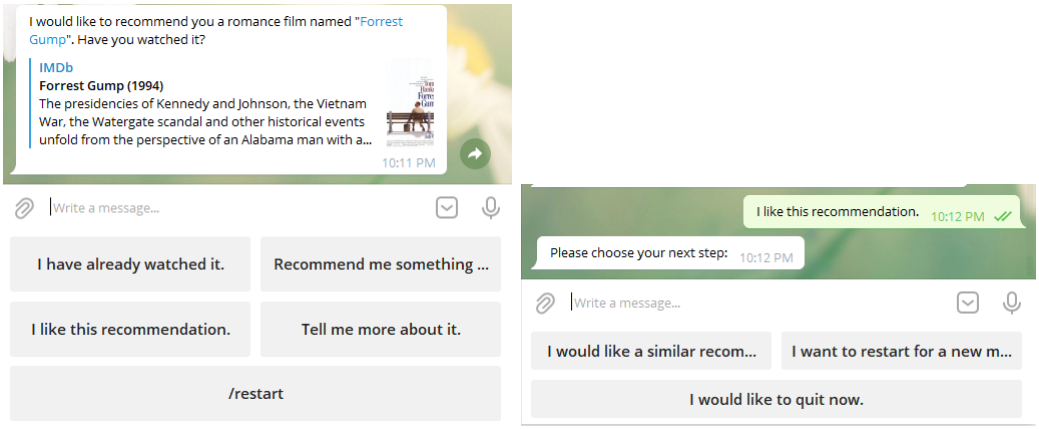

Figure 3.5: Keyboard with buttons

a list of keyboard buttons. Telegram gets the URL for the imdb page from the agent utterance.

3.2 Problem Analysis and Requirements

The goal is to adapt the IAI MovieBot code to the Messenger Platform and Telegram in such a way that both can be serviced at the same time using a multi-modal controller, and to make the user experience of both platforms as identical as possible. Additionally, the Messenger Platform application should be approved by Facebook.

In Telegram, the keyboard located below the user input field creates a clean user experience by avoiding posting buttons in the chat window. Additionally, the keyboard is removed from the conversation when it is no longer used. The Messenger Platform equivalent of the keyboard would be the persistent menu, but it is as the name suggests, persistent, meaning it will always be present and contain the same buttons throughout the conversation.

Since the Messenger Platform does not offer a keyboard with the same functionality as that of Telegram, the options have to be presented with the use of the button template or quick replies. Button templates are posted in the chat window and have a limit of maximum three buttons, which means the options will either have to be contained in multiple templates, or some options removed. Another issue with the button template is that the buttons are only able to show roughly half the text length as that of the buttons in Telegram.

The movie recommendation is posted together with a keyboard, creating the same issue of double posting as described above. By using the generic template in the Messenger Platform, it is possible to combine the movie recommendation along with three buttons in the same post.

For the multi-modal controller, Telegram and the Facebook messenger application should run in the same server instance. As the original Telegram application is using polling to retrieve updates, the code has to be rewritten to use a webhook instead. Methods for the webhooks of both applications will then be added to the same flask server, each using a separate controller class.

The approval process will be started when the Facebook messenger part of the code is finished.

3.3 System Architecture

The system consists of a Flask server, the recommender agent, a Telegram controller class, a Facebook messenger controller class and a helper class for the messenger controller.

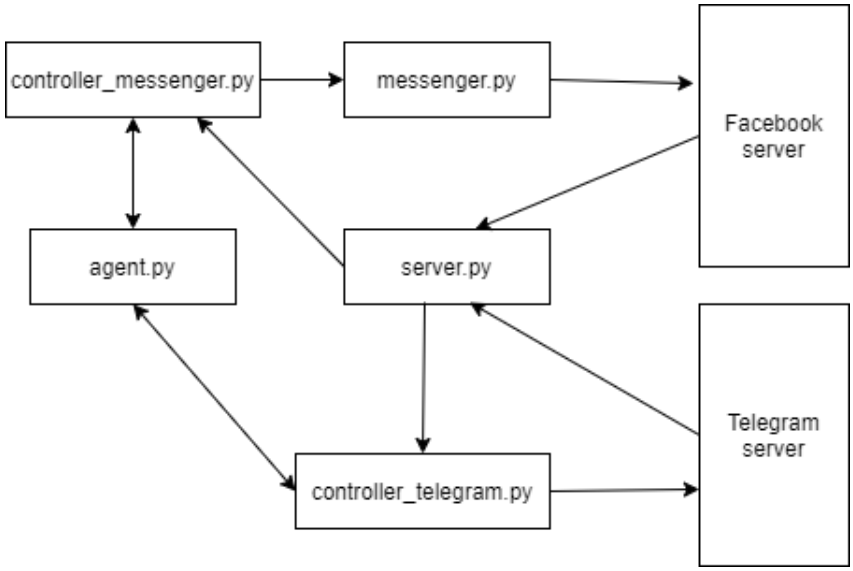

Figure 3.6: System Architecture

The server receives post requests from both the Facebook server and the Telegram server. The relevant information is then extracted from the message, and sent to one of the con-

trollers. The controller passes the input to the agent and decides what to do with the response. A message is then sent from the controller to the Telegram or Facebook API, where it is presented to the user. In the case of the Messenger controller, a helper class is responsible for sending post requests to the Facebook server.

3.4 Adding Facebook Messenger as a Platform

3.4.1 Server and webhook setup

A separate server file was created to handle incoming requests from the Facebook and Telegram servers. Requests coming from the Telegram server is routed to the respond method, which uses the telegram python library to update the message text directly. If the incoming message is a command, the method invokes another method in the Telegram controller. It was done this way to reduce the amount of changes made to the original Telegram controller.

```
@app.route('/{}'.format(telegram_token), methods=['POST'])
def respond():
    update = telegram.Update.de_json(request.get_json(force=True), bot)
    text = update.message.text.encode('utf-8').decode()

    if text == "/start":
        controller_telegram.start(update, True)
    elif text == "/restart":
        controller_telegram.start(update, True)
    elif text == "/help":
        controller_telegram.help(update, True)
    elif text == "/exit":
        controller_telegram.exit(update, True)
    else:
        controller_telegram.continue_conv(update, True)
    return 'ok'
```

The webhook for Telegram is set in the run method upon server startup. The URL for the webhook is located in the configuration file and is passed to the run method. Webhook setup is done using the telegram library with the URL and the Telegram access

token.

```
def set_webhook(webhook_url):
    webook = bot.setWebhook('{URL}{HOOK}'.format(URL=webhook_url, HOOK=telegram_token))
    if webook:
        return "webhook ok"
    else:
        return "webhook failed"
```

Requests coming from the Facebook server is routed to the receive message method. When the webhook URL is initially set up in the Messenger Platfrom settings page, a GET request is sent to the URL to confirm the verify token. When the setup is complete the Facebook server is ready to send POST requests. When a POST request is received, user id and payload is extracted from the object and passed to the Messenger controller.

During the development process the server was hosted using ngrok. Ngrok forwards the local port to an external URL that can then be used as a HTTP server.

```
def run(config, webhook_url):
    controller_telegram.execute_agent(config)
    controller_messenger.execute_agent(config)
    set_webhook(webhook_url)
    app.run(host='0.0.0.0', port=environ.get("PORT", 5000))
```

3.4.2 The Messenger controller

The Messenger controller class is based on and similar to the Telegram controller. For each new user that starts a conversation with the Messenger application, a separate agent object is created with the user's id. Whenever a new message is received from a user, the payload is passed as a user utterance to the agent. One of the main responsibilities of the controller is to handle the resulting agent utterance and decide what to do with it.

There are three types of agent responses that has to be separated and handled differently by the controller, text, user options and the movie recommendation. First, the agent response is checked for user options. This is simply done by checking if the user options list is empty. Then it checks if it's a movie recommendation or just a list of user options. If it's neither the response is text.

```
def send_message(self, user_id, payload):
    self.continue_dialogue(user_id, payload)
    if self.user_options[user_id]:
        buttons = self.user_messages[user_id].create_buttons(self.get_options(user_id))
        if "**" in self.agent_response[user_id]:
            self.movie_template(user_id, buttons)
        else:
            self.user_messages[user_id].buttons_template(buttons, self.agent_response[user_id
    else:
        self.user_messages[user_id].text(self.agent_response[user_id])
```

Once the type of response has been decided, the message is converted to a JSON object and sent to the Send API. Communication with the Send API is handled by the messenger class.

3.4.3 Send API communication

In the messenger class the JSON objects for button, text and templates are generated and sent to the Send API. The class contains methods for each message type. Below is the code for sending the button template.

```
def buttons_template(self, buttons, text):
    template = {
        "recipient":{ "id": self.user_id},
        "message":{
        "attachment":{
            "type":"template",
            "payload":{
            "template_type":"button",
            "text":text,
            "buttons":buttons
            }
        }
        }
    }
    return requests.post(self.button_template_uri, json=template).json()
```

The contents of the object is passed from the messenger controller. It is then sent as a POST request to the Send API. The POST request includes the URI required for the button template, the access token and the object in JSON format.

```
self.button_template_uri =
    'https://graph.facebook.com/v2.6/me/messages?access_token='+self.token
```

3.4.4 Adapting the user experience

In attempting to create the same user experience for Messenger as that of the original Telegram version, the main limitation to overcome is the lack of an integrated keyboard option in the Messenger Platform. Agent inquiries and user text input will be identical on both platforms.

Two types of postback buttons were looked at for the presentation of user options, quick replies and the button template 3.7. Quick replies can contain up to 13 buttons, and are presented as scrollable horizontal list with title and button text. The buttons template is presented as a block with a list of buttons listed vertically.

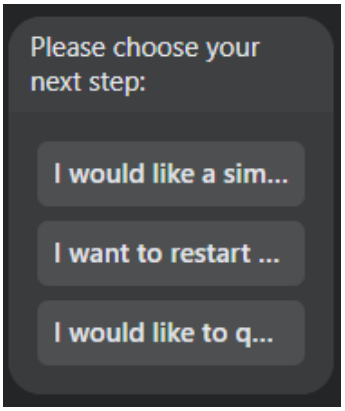

Figure 3.7: Buttons template

While quick replies has the ability to represent all the user options, without having to remove any functionality, most of the buttons will be hidden behind the scroll. Additionally, the button text length is too short to represent the options in a clear manner. Because of this, it was decided that the buttons template was the better option.

The persistent menu was considered for functions such as "restart" and "quit", but at the time of development this option had been temporarily disabled by Facebook while they

are adapting the platform to the new General Data Protection Regulation (GDPR) [3]. The "mark seen" and "typing bubble" were also disabled for the same reason.

The movie recommendation is presented along with user options. To avoid posting two times in the chat, a template containing both the movie and the buttons is preferable. The template can only fit three buttons, so "Tell me something about it." and "/restart" was removed. "/restart" is was deemed unnecessary, since it is always available as a user command. Movie rating and duration was integrated into the template to compensate for the removal of the "Tell me something about it." option.

The movie recommendation is presented with the generic template message. As this is not automatically generated based on the URL, like in Telegram, the contents of the template has to be manually retrieved and added. Title, rating, duration, summary, image url and the imdb link is from the database with the use of a separate method. The method uses the database already present in the MovieBot code and gets the information based on the movie id.

```
def movie_info(self, movie_id, user_id):
    for row in self.lookup().execute(f'SELECT * FROM movies_v2 WHERE ID="{movie_id}"'):
        self.info[user_id] = {
            "title": row[1],
            "rating": row[4],
            "duration": row[6],
            "summary": row[10],
            "image_url": row[9],
            "imdb_link": row[12]
        }
```

The movie id is retrieved from the agent response, where the IMdb page id is the same as the movie id used in the database, as shown below.

```
I would like to recommend you an action film named similar to batman named
**"[Batman Begins](https://www.imdb.com/title/tt0372784/)"**. Have you watched it?
```

```
def get_movie_id(self, response):
    if "/tt" in response:
        start = response.find("/tt")
        movie_id = response[start+3:start+10]
        return movie_id
```

In the template generation, rating and duration is appended to the title text. The first three items in the user options list are added as buttons to the template.

```
def movie_template(self, user_id, buttons):
    title = self.info[user_id]['title'] + " " \
    + str(self.info[user_id]['rating']) + \
        " " + str(self.info[user_id]['duration']) + " min"
    self.user_messages[user_id].template(
        buttons[0:3], self.info[user_id]['image_url'],
        self.info[user_id]['imdb_link'], \
            self.info[user_id]['summary'], title)
```

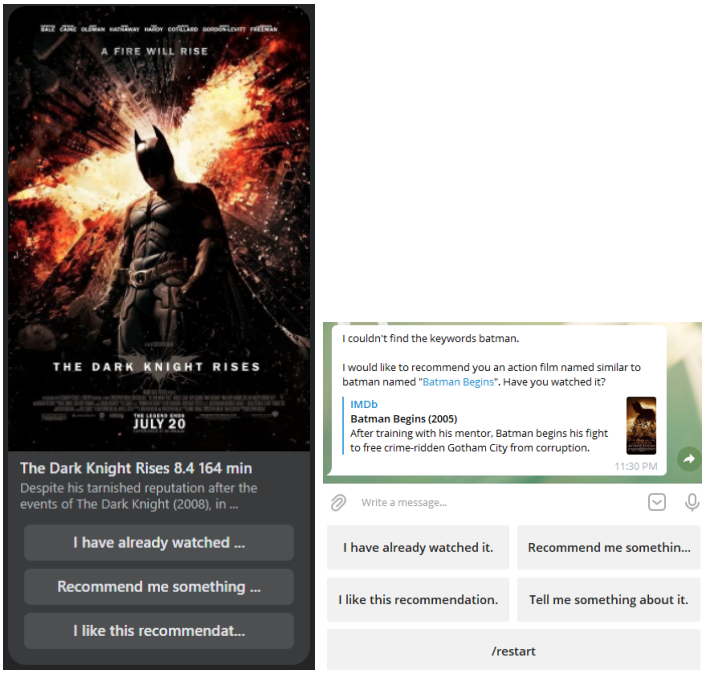

Figure 3.8: New vs original movie recommendation.

To fix the problem of text overflowing in the button text, a function was added that creates shorter versions of the options, while still sending the original text string back to

the agent 3.9. The title is set to the shorter version of the user option, while the payload contains the original string. When the button is pushed, the title is posted in the chat window, while the payload is sent to the Send API and eventually passed to the agent.

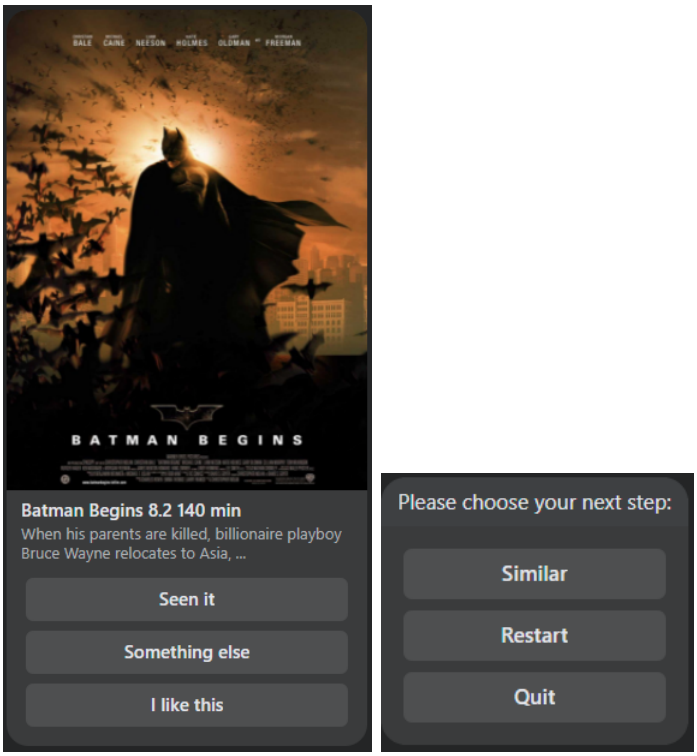

Figure 3.9: Shortened button text.

```
for option in self.user_options[user_id].values():
    for item in option:
        options.append({"button_type": "postback", "title":
            self.shorten(item), "payload": item})
return options
```

The greeting text was added to the welcome screen. When "get started" is pushed the user is presented with information regarding the logging of conversations, and the option to accept the conditions or start without storing any data. If "Accept" is pushed, conversations will be stored for this user. There is also an URL button, linking to the full privacy policy on github pages. The user has the ability to stop storing the data and delete all stored conversations by typing "/delete". When "Start" is finally pushed, the

actual conversation with the bot is started.

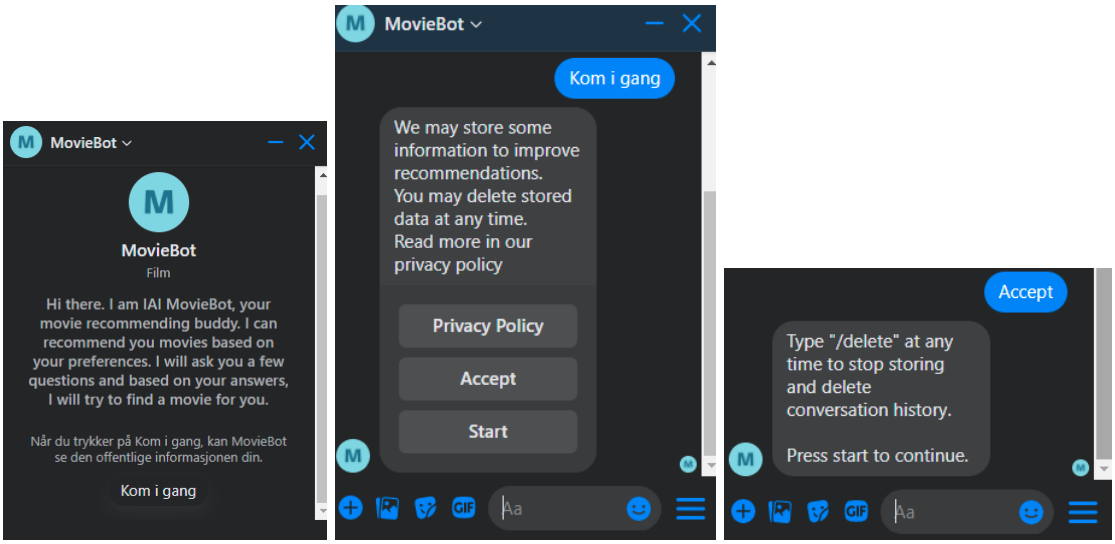

Figure 3.10: Greeting, privacy policy and deletion.

3.5 Logging and Personal Data Management

The IAI MovieBot has the option to store conversations between the users and the bot. Due to regulations on personal data storage, it is required to inform the user of what data is stored and for what purpose. Facebook also requires that a privacy policy be added to the app. The text below is taken from Facebook's GDPR pages [3].

> **Data controller**
>
> A company is a data controller when it has the responsibility of deciding why and how (the "purposes" and "means") the personal data is processed.
>
> Under the GDPR, data controllers have to adopt compliance measures to cover how data is collected, what it's used for and how long it's retained. They also need to make sure that people can access the data about them. Data controllers must ensure that data processors meet their contractual commitments to process data safely and legally.

Data processor

A company is a data processor when it processes personal data on behalf of a data controller. Under the GDPR, data processors have obligations to process data safely and legally.

Messenger

On the Messenger platform, Facebook is a data controller in most cases as conversation between people and businesses is considered on-platform activity. As the data controller, we handle personal data as described in our Data Policy. Please note, even in instances where Facebook is a data controller, your business may also be considered a data controller under the GDPR.

A privacy policy was created following the Facebook guidelines and a GitHub page was added to the GitHub repository, linking to the policy.

The logging of conversations was done with functions already available in the original code, with the addition of an if statement dependent on if the user has accepted the privacy policy. A function for retrieving which movies were liked and already seen for a particular was added.

```
def load_user_data(self, user_id):
    user_history_path = self.path + 'user_' + user_id + '.json'
    self.load_data[user_id] = {}
    if os.path.isfile(user_history_path):
        with open(user_history_path) as json_file:
            data = json.load(json_file)
            for conversation in data:
                for movie in conversation["Context"]:
                    self.load_data[user_id][movie] = conversation["Context"][movie]
        }
```

3.6 Generalizing MovieBot to Multiple Messaging Platforms

To get the MovieBot to run on both platforms, first the Telegram controller had to be configured to use a webhook instead of polling, as described in the server setup. Code was also added to the Telegram controller and the configuration file to make the use of

a webhook optional.

Due to the limitations in the Messenger Platform and the choice to remove some of the user options, changes also had to be made to the Telegram application to create a similar user experience. Code was added that offers a configurable option to remove the same buttons as those in Messenger and display a shorter button text.

```
remove = False
if self.configuration['MESSENGER_MODE']:
    for option in list(self.user_options[user_id].values()):
        if "Tell me" in option[0]:
            remove = True
if remove:
    reply_keyboard = self._recheck_user_options(
        deepcopy(list(self.user_options[user_id].values())[0:3]))
else:
    reply_keyboard = self._recheck_user_options(
        deepcopy(list(self.user_options[user_id].values())))
```

The code above only displays the first three user options in the list, if the "Tell me more about it." option is present.

In Telegram, it is not possible to have different values for payload and title in the buttons, making the implementation of the shorter button text option slightly more complicated. The agent utterance is first converted to the short version and presented as buttons. Then, when a button is pushed, the value is converted back to the original string and passed to the agent as a user utterance.

3.7 Facebook Approval Process

The Facebook approval process was started when the Messenger application was functional and deemed ready for external testing.

The pages messaging permission was the only necessary permission that had to be requested. As this is the most basic permission required to interact with users, a very simple description was given.

Instructions for how to test the bot with a link to the Messenger application was provided. Since the only way to test an unpublished application externally is through a test

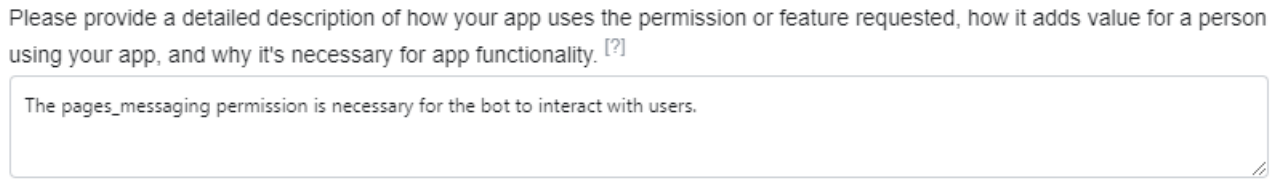

Figure 3.11: Description of permission.

user, this was created and the credentials added to the instructions. Finally, a screen recording showcasing the use and functionality of the MovieBot was uploaded.

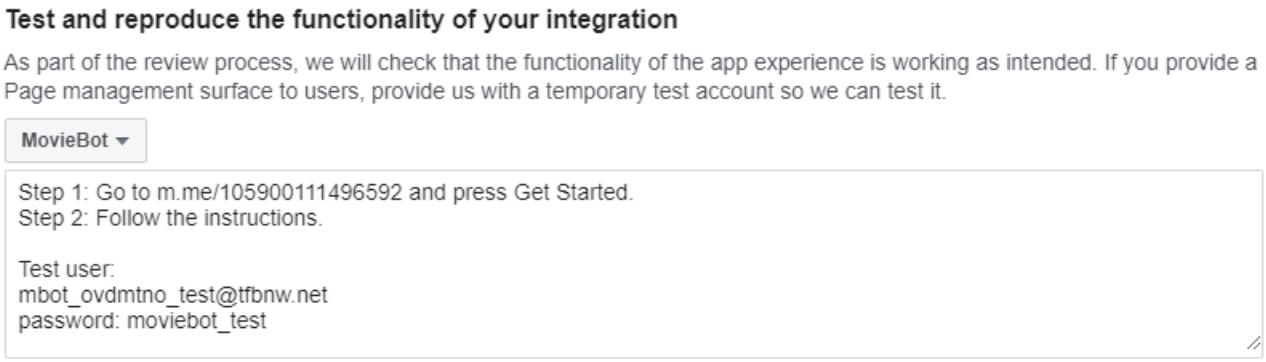

Figure 3.12: Instructions for testing.

Before the application can be submitted for review, the user of the Facebook account has to verify their identity. Following the instructions provided, a government ID was uploaded, but the verification failed. This was attempted multiple times. One possible explanation is that Facebook has reduced capacity for review due to the COVID-19 pandemic.

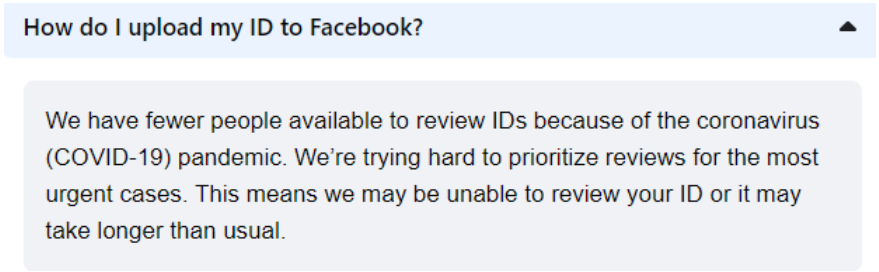

Figure 3.13: Reduced review capacity.

Chapter 4

Results

The IAI MovieBot was successfully adapted as a functional Messenger bot. Due to the problems with identity verification, the approval process could not be started in time. The Messenger bot is fully functional in development mode and can be modified and tested in a development environment. Anyone with access to the code can start the approval process at a later stage.

A multi-modal controller was implemented, allowing a single server instance to service both platforms simultaneously. Until the Messenger application eventually gets approved, it is still possible to run Telegram in its original state.

The final objective of the thesis was to create a user experience that was as close as possible to that of the original. The original Telegram bot, the Messenger bot and the adapted Telegram bot should therefore be tested separately to assess the success of this goal.

4.1 User Experience

To get a qualitative measure of the success of the adapted Messenger version of the MovieBot recommender system, all three versions of the systems were tested by test subjects.

The questions for the evaluation survey was taken from the master's thesis for the MovieBot [11]. As the Messenger application is not available for testing by the public, the testing had to be done in person. This made it difficult to recruit a large number of people, due to the current pandemic situation. The result was three participants for

each version.

- **Questions**
  - Q1. The system is effective in recommending movies.
  - Q2. The system is able to understand my requirements.
  - Q3. The system is able to adapt to changes in my requirements.
  - Q4. I understand what my options are during the conversation.
  - Q5. The conversation with the system feels like a normal human-human conversation.
  - Q6. My experience with the system was overall enjoyable.
  - Q7. I would use this system in the future for movie recommendations.

4.1.1 Feedback

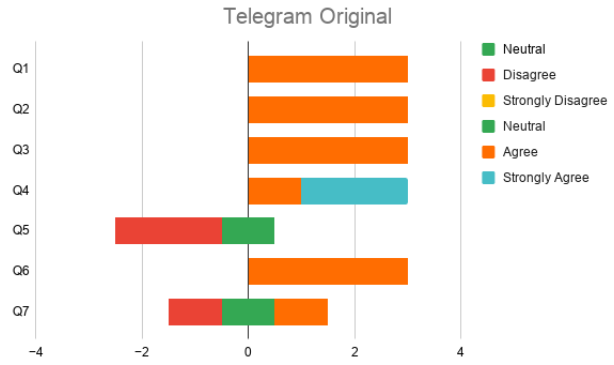

Figure 4.1: User feedback for the Original Telegram version.

4.1.2 Analysis

With only three participants for each version, it is difficult to draw any conclusions based on the feedback. The Messenger version performed slightly worse on average. An important thing to note is that the difference between the versions are mostly reflected in questions 6 and 7. The other questions should perform close to or identical across all versions.

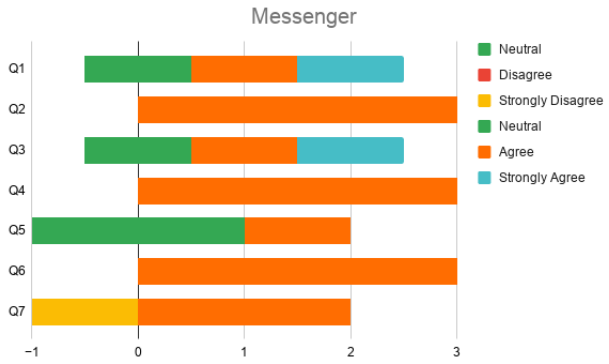

Figure 4.2: User feedback for Messenger.

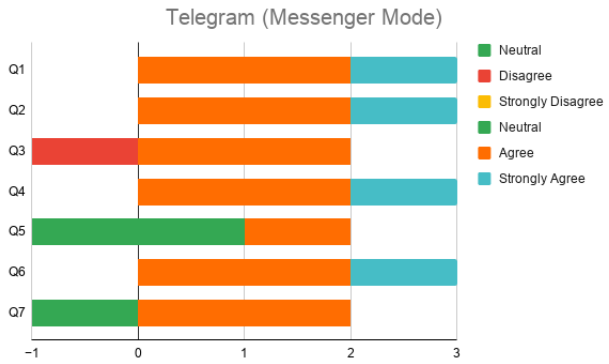

Figure 4.3: User feedback for the adapted Telegram version.

Chapter 5

Conclusions

The overall goal of creating a Messenger application for the IAI MovieBot was successful. Apart from getting the application approved by Facebook, all objectives were completed.

The adaptation of the MovieBot recommending system to Messenger was done with two main components, the Messenger controller and the server. The controller is the interface between the recommending system and the server, while the server connects the controller to Facebook.

The multi-modal controller was implemented as specified in the goals, using a single server instance to run both Telegram and Messenger simultaneously. As the Messenger application required a server and webhook setup by default, all that was needed was to add a route and webhook setup for Telegram to the server. Testing was done to make sure that users from both messenger platforms could use the system at the same time.

The Facebook approval process was halted due to issues with identity verification.

The user experience for the Messenger application could not be made identical to the Telegram version, due to limitations in the Messenger Platform. In addition to limitations in the development tools and features in the Messenger Platform, the overall user experience is better in Telegram. For instance, keyboards in Telegram can be removed when they are no longer needed, as opposed to buttons and templates in Messenger that are posted in the chat window permanently.

Some of the implemented solutions could potentially have been done differently. The decision to avoid posting multiple components in the chat window for a single agent response created some limitations in terms of what components could be used, and how.

For instance, quick replies could have been used together with the movie template, displaying all user options.

Bibliography

[1] Azure Health bot overview. https://www.microsoft.com/en-us/research/project/health-bot/. Accessed: 2021-05-12.

[2] Conversational Systems report. https://www.grandviewresearch.com/industry-analysis/conversational-systems-market. Accessed: 2021-05-12.

[3] GDPR. https://ec.europa.eu/info/law/law-topic/data-protection_en. Accessed: 2021-05-12.

[4] IAI MovieBot, A Conversational Movie Recommender System paper. https://arxiv.org/pdf/2009.03668.pdf. Accessed: 2021-05-12.

[5] IAI MovieBot github. https://github.com/iai-group/moviebot. Accessed: 2021-05-12.

[6] Messenger Platform introduction. https://developers.facebook.com/docs/messenger-platform/introduction. Accessed: 2021-05-12.

[7] Rapid Response Virtual Agent overview. https://cloud.google.com/solutions/contact-center/covid19-rapid-response. Accessed: 2021-05-12.

[8] Social Networks users. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. Accessed: 2021-05-12.

[9] Telegram Bots developers. https://core.telegram.org/bots. Accessed: 2021-05-12.

[10] The Messenger Platform developers. https://developers.facebook.com/docs/messenger-platform/. Accessed: 2021-05-12.

[11] Javeria Habib, Shuo Zhang, and Krisztian Balog. Iai MovieBot: A conversational movie recommender system. In *Proceedings of the 29th ACM International Conference on Information Knowledge Management*, CIKM '20, pages 3405–3408, 2020.