BROCK UNIVERSITY
FACULTY OF MATHEMATICS AND SCIENCE

COSC 4P02 - CANADA GAMES CHATBOT

By:
Eddy Su (6459705)
Aman Braich (6511679)
Manvendrasinh Rana (6137228)
Rikveet Singh Hayer (6590327)
Sager Kudrick (5919170)
Sawyer Fenwick (6005011)
Raghav Bhardwaj (6548580)

TEAM UNDEFINED PROGRESS REPORT

DEPARTMENT OF COMPUTER SCIENCE
ST. CATHARINES, ONTARIO, CANADA

March 28, 2022

TABLE OF CONTENTS

1. OVERVIEW

This report details the progression towards the chat-bot project over Sprint's 3(4th March) and 4(18th March). It provides insight into what parts have been completed and what parts need to be implemented. As we progress into the final sprints of this project, we have made considerable changes concerning the AI model. However, we now have a system that is finalized. Development in the Front-End system that has been consistent with what we had initially sought to be. The database system has been set up. Currently, we are working on transitioning the data from the Database to the AI via tables. Overall this project report is an update on the modifications/improvements made to the project and the new components that have been completed and enabled during this sprint as well as an overview of the overall completion of the project. For further information (e.g., project code), a link to the related GitHub page is available in Section 6.

1.1. Contributions

All Team Members worked together on Product Backlog, Sprint Backlogs and Reports.
All Team Members worked together to write scraping and testing code for scraping modules.
All Team Members contribute to the GitHub page, uploading their code to the appropriate branches (Front-End, Back-End, Production, Testing).
Rikveet Singh Hayer researched, designed and implemented the AI, and is our Team Leader.
Manvendrasinh Rana created the GitHub page.
Sager Kudrick and Sawyer Fenwick designed and implemented the MySQL Database.
Raghav Bhardwaj and Aman Braich designed and implemented the UI on desktop and mobile.
Eddy Su mapped out websites for scraping and the scraping itself.

2. DESIGN

2.1. System Design

The overall system has been finalized. The front-end implementation remains the same. When the user asks a question the front end sends the query with a call back function to the back end. The back end is structured as follows. All the input/output between Python components(Ai, Scraping, Database) and the node.js server is handled by Js Modules Handler. It uses the python-shell library that can spawn a python process. Instead of function calls system input and output are sent and receive data between the two processes. The library has an observer function ”on” with parameters ”message”, ”error” and ”pythonError”. For each request, the Modules handler returns a promise that is resolved or rejected depending on the type of message printed by the process. The Overall flow of the system can be seen in the following diagram.

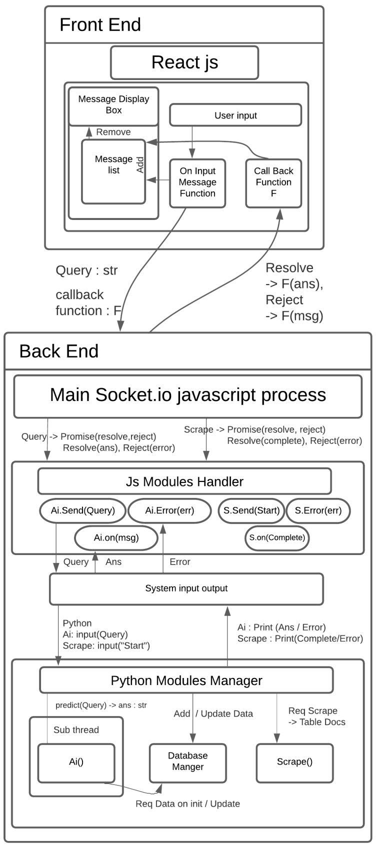
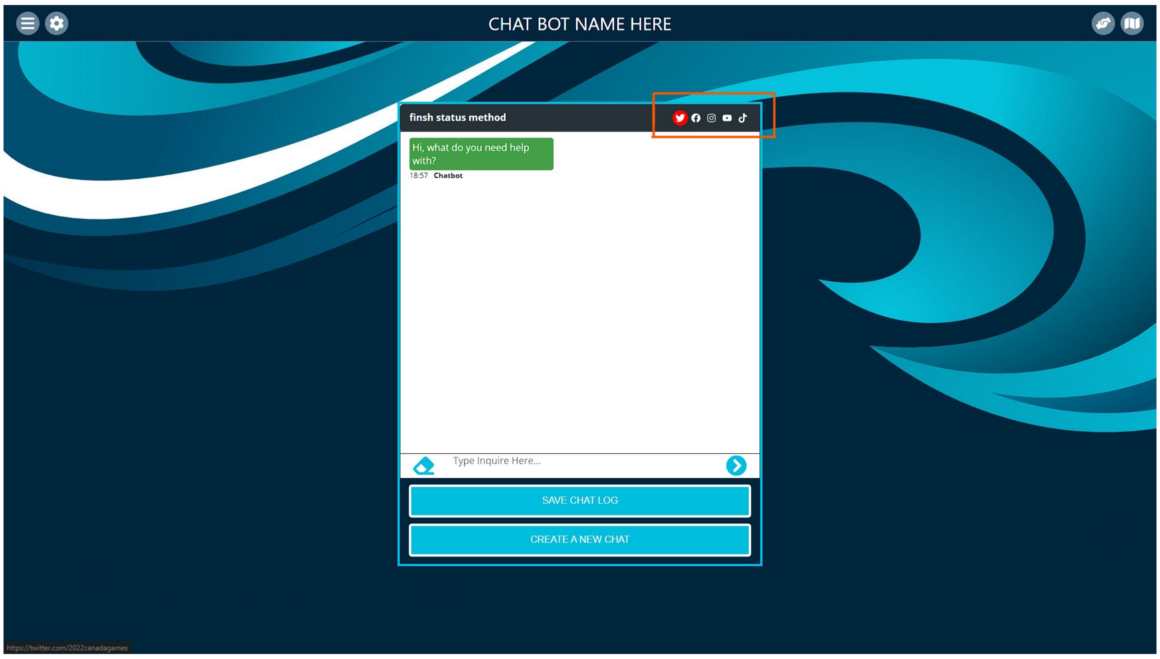

Front End
React js
Message Display Box
User input
Remove
Message list
Add
On Input Message Function
Call Back Function F
Query : str
callback function : F
Resolve -> F(ans), Reject -> F(msg)
Back End
Main Socket.io javascript process
Query -> Promise(resolve,reject) Resolve(ans), Reject(error)
Scrape -> Promise(resolve, reject) Resolve(complete), Reject(error)
Js Modules Handler
Ai.Send(Query)
Ai.Error(err)
S.Send(Start)
S.Error(err)
Ai.on(msg)
S.on(Complete)
Query
Ans
Error
System input output
Python Ai: input(Query) Scrape: input("Start")
Ai : Print (Ans / Error) Scrape : Print(Complete/Error)
Python Modules Manager
predict(Query) -> ans : str
Add / Update Data
Req Scrape -> Table Docs
Sub thread
Ai()
Database Manger
Scrape()
Req Data on init / Update

Figure 2.1: System Design

3. CHANGES

3.1. Website

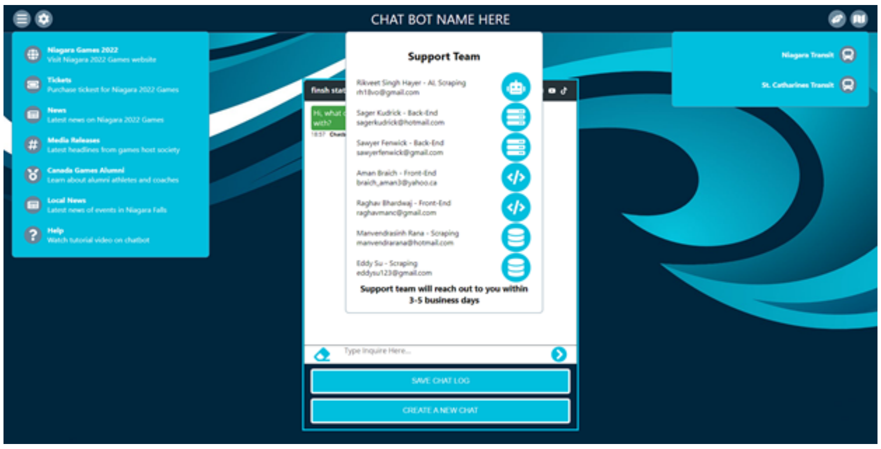
We have added a few more features to the front end to make it more user friendly and accessible on all platforms. One of the features added was the links for social media so a user can easily access any of the available media they would like to use. By clicking any of the 5 icons it will direct the user to the appropriate social media.



Hi, what do you need help with?
18:57 Chatbot

Figure 3.1: Social Media Links

We also decided to split up the menu to make a more even and symmetrical looking screen. We have added a pop-up menu with all transit information on the right along with the support pop up button which will include all the developer names, roles, and emails.


CHAT BOT NAME HERE
Niagara Games 2022
Visit Niagara 2022 Games website
Tickets
Purchase tickest for Niagara 2022 Games
News
Latest news on Niagara 2022 Games
Media Releases
Latest headlines from games host society
Canada Games Alumni
Learn about alumni athletes and coaches
Local News
Latest news of events in Niagara Falls
Help
Watch tutorial video on chatbot
Niagara Transit
St. Catharines Transit
Type Inquire Here...
SAVE CHAT LOG
CREATE A NEW CHAT

Support Team

Rikveet Singh Hayer - AI, Scraping
rh18vo@gmail.com

Sager Kudrick - Back-End
sagerkudrick@hotmail.com

Sawyer Fenwick - Back-End
sawyerfenwick@gmail.com

Aman Braich - Front-End
braich_aman3@yahoo.ca

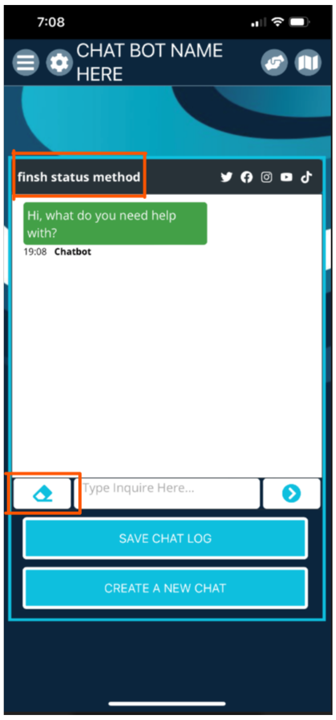
Raghav Bhardwaj - Front-End
raghavmanc@gmail.com

Manvendrasinh Rana - Scraping
manvendrarana@hotmail.com

Eddy Su - Scraping
eddysu123@gmail.com

Support team will reach out to you within 3-5 business days

Figure 3.2: All New Links



Hi, what do you need help with?
19:08 Chatbot

Figure 3.3: Buttons and Media

On the left, we have the general menu pop-up which will help the user find links to, the Games website, active news relating to the games, where to buy tickets and a helpful video that will direct them to YouTube. We also have the settings icon menu which is still to be implemented on the website. The clear input button was also moved away from the send button to avoid any accidental message clears or sends, making it a smoother user experience when using the chatbot. Lastly, a status method has been implemented so the user knows what is going on with the chatbot. It will read ”waiting for user input” or ”Generating a response” depending on the situation of the conversation.

3.2. AI

Haystack framework was implemented instead of a basic Google/Tapas implementation. This framework also supports tabular q/a as discussed in the previous report. This frameworkloads the tables into memory. Each table has a key, header, content and metadata. The following dictionary is an example document in the document store.

```
{"doc_x": {
  "id": "canoe"
  "content": "Pandas data-frame of the table",
  "content_type": "table"
  "meta": {
    "title": "Canoe table",
    "section_title": "Info regarding what is stored",
    "url": "Url of scraped page"
}}}
```

This framework has a document store that stores all docs. These docs can be updated using the id. To handle multiple tables the framework has a table retriever. This retriever uses a document store, deepset/bert-small-mm retrieval-question encoder for queries, deepset/bert-small-mm retrieval-passage encoder for passages, deepset/bert-small-mm retrieval-table encoder for handling table info and metadata fields of the document. using this info the retriever returns k tables from the document store based on their ranking for a particular query. Then the best table selected from the prediction's results and the Table-Reader component of the framework is used. This component uses google/tapas-large-finetuned-sqa model for answering a query on the table. It has an accuracy of 72% on Microsoft Research Sequential Question Answering (SQA) Dataset. The result of the table reader is returned as an answer. There are four possibilities for the result. Correct/Incorrect answer, incomplete answer and no answer. Only the no answer case can be handled beforehand. When the answer is empty we can return the URL from metadata to help the user find more info on the question.

4. TESTING

As we near the end of this project, one of the most important and often overlooked aspects of an application like this is its reliability and security. When referencing the project requirements, we can see what this application can and cannot do; particularly, what kind of data we should accept, store, and present.

While there currently aren't any material test cases, we have been considering how to integrate testing and authentication into our future sprints to ensure we validate, test, and authenticate our software. We will be doing this by:

• Creating test cases for inserting, updating, and retrieving information from the database.

• Creating specific procedures for inserting, updating, and retrieving information from the database. Since there will be some interfacing between a users input string and the database, we should ensure there are no scenarios where bad actors can negatively affect the application; such as SQL injection, ext.

• Creating test cases for interacting with the AI to ensure proper output is achieved, such as: properly responding to invalid requests, responding to scenarios where information is not available, and responding with information.

• Looking into an automated workflow for testing these sub-processes of the application.

• Ensuring the functionality of the application is complete and fully satisfies user requirements.

• Ensuring the application and its functionality is what the user actually wants.

5. SPRINTS

5.1. Meetings

All members of our team are present for meetings, as it is vital to get everyone's input and expertise. We hold formal meetings twice a week and communicate throughout the week over a dedicated Discord Server/Microsoft Teams chat.

5.2. Sprints

Within a sprint, team members would choose activities to contribute to, moving the sprint forward and iteratively building the application. Team members chose their activities for the sprints, and we will include individual names next to the activities they worked on. Our team has opted to use Miro.com to visualize our sprints, as well as keep track of the progress/-completed tasks, and can be viewed by clicking on the ”Sprint Backlog” link at the end of this document in Section 6.

The Tasks for the Sprints during this period were focused on creating Scraper modules to scrape the websites, implementing the Front End that was designed in Sprints 1 and 2, and beginning to train the AI on our data.

5.2.1. Sprint 3

FRONT END:

• Create a greeting message for session start

• Create a Reset Messages Button

• Create a Send Message Button

• Create a Support Button

• Create a Help Button

• Create a Tickets Button

• Dynamically change the input field size in relation to text entered but keep width fixed

• Make CSS appealing for Desktop/Mobile

SCRAPING:

• Create Scraping Module for Athletes

• Create Scraping Module for Sports

• Create Scraping Module for Medals

AI:

• Deploy the AI (Implement the Ai framework to work with scraping data)

• Setup auto develop and deploy on command configuration (Implement script to load data and initialize Ai on the data)

5.2.2. Sprint 4

FRONT END:

• Create a Button to Print/Download the Chat Log

• Add Links to local Transit websites

• Add a link to local News websites

SCRAPING:

• Finalize Scraping Canada Games Website • Finalize Scraping Gems Pro

• Finalize Scraping Niagara Transit Website

BACK END:

• Create an interface for the server-side to submit questions and receive responses

AI:

• Return Human readable Output for question answer

5.3. Future Sprints

Our future and final sprints for this project will be focused on testing the system. This will include creating test classes for each of our modules/methods and ensuring they all return correct results. We have written more in-depth about testing the system in 4.

6. PROBLEMS

- Transfer of data between JavaScript and Python : Solved using Python-Shell library which uses sys input and output to transfer data in-between Python and JavaScript processes it supports text and JSON type info.
- Handling multiple tables: The table receiver component of the Haystack framework has resolved this issue.
- Handling large tables: The table size is very large regarding events. The tuples can go up to 2000 entries. Therefore events and sub-events are combined to create a single table reducing the size of tables but increasing the number of tables. This trade-off is valid because it increases the accuracy and reduces query time.

7. LINKS

GitHub
Sprint Backlog
Haystack framework
Deepset models for retriever
Tapas model for reader