COSC 4P02 FINAL REPORT

Chatbot application for COSC 4P02 Winter by Naser Ezzati-Jivan

Abstract

The Design Decisions and Team Management behind the Brock U and Canada Games chatbot application written for the COSC 4P02 course.

Grant Ferrier gf18fi@brocku.ca (6569388) Team Lead

Aldric Joya aj18gv@brocku.ca (6589865)

Arin Yaldizciyan ay16va@brocku.ca (6068431)

Justin Zhang jz16ig@brocku.ca (6217251)

Sabih Zubair sz18oj@brocku.ca (6552152)

Thanikash Kanagaratnam tk18il@brocku.ca (6586085)

Contents

Overview

Project Public GitHub: https://github.com/TheGrai/COSC-4P02

The goal of this project was to create a chatbot application that would be able to use a natural language processor to use machine learning to understand questions and respond in a natural way. The end result was to be a browser based chatbot that would be able to answer questions both about Brock University as well as the upcoming 2022 Canada Games hosted in Niagara. This document should give you an overview of the software in question ranging from how to use the software to design decisions and the process the development team used to accomplish this goal.

To aid in understanding this application, here is a brief overview of the chatbot application. The main application is a React frontend that interacts with a Django backend using a REST API. The Natural Language Processor (NLP) is a python library called Natural Language Toolkit (NLTK).

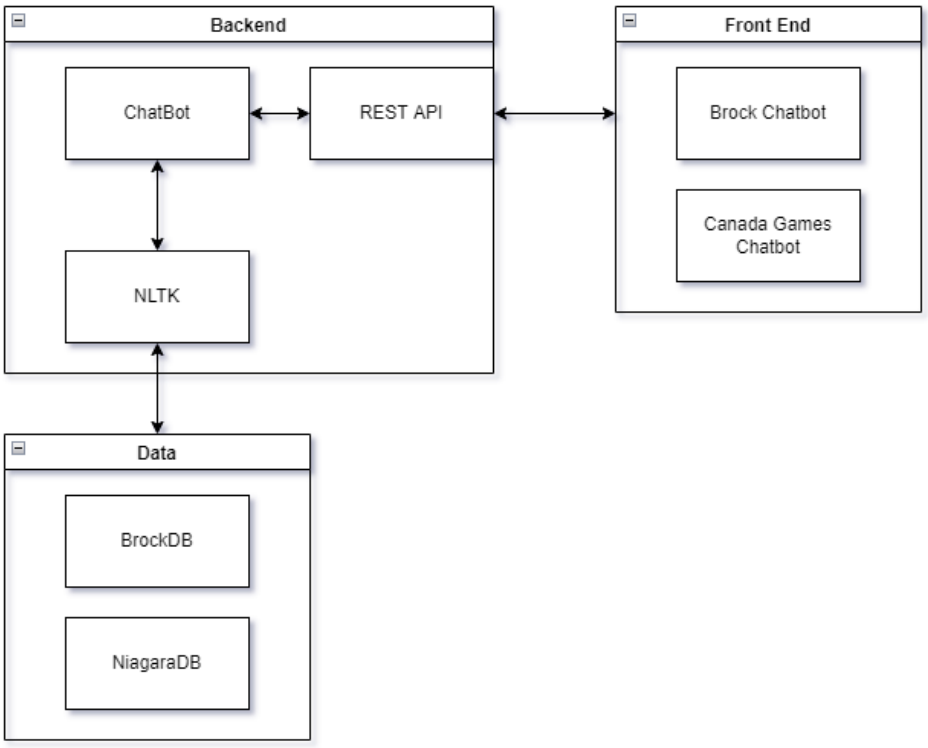

Figure 1 - Framework Overview

This system has been dockerized. Each main component has been made into its own container. This allows for easy deployment as well as future scalability.

Throughout this document the Canada Games may be referred to as the Niagara Games. This is due to the 2022 Canada Games being hosted in Niagara. In this document these two names are synonymous.

Instructions for Usage

General Chat

When the user first connects to the chatbot they will be welcomed with the welcome screen as shown in Figure 2.

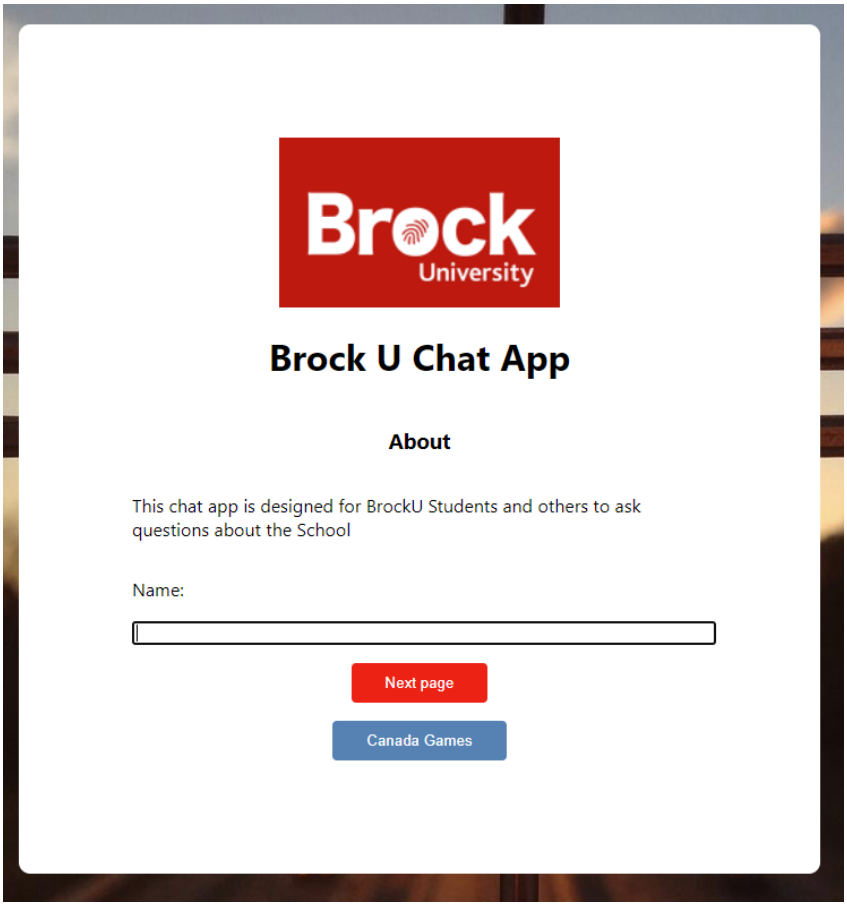

Figure 2 - Brock Welcome Screen

The user is then able to select which version of the chatbot they would like to access. If the user wishes to switch to the Canada Games chatbot they may select the button with “Canada Games”. If they are on the Canada Games version, they can change versions by clicking on the “BrockU” button as shown in Figure 3.

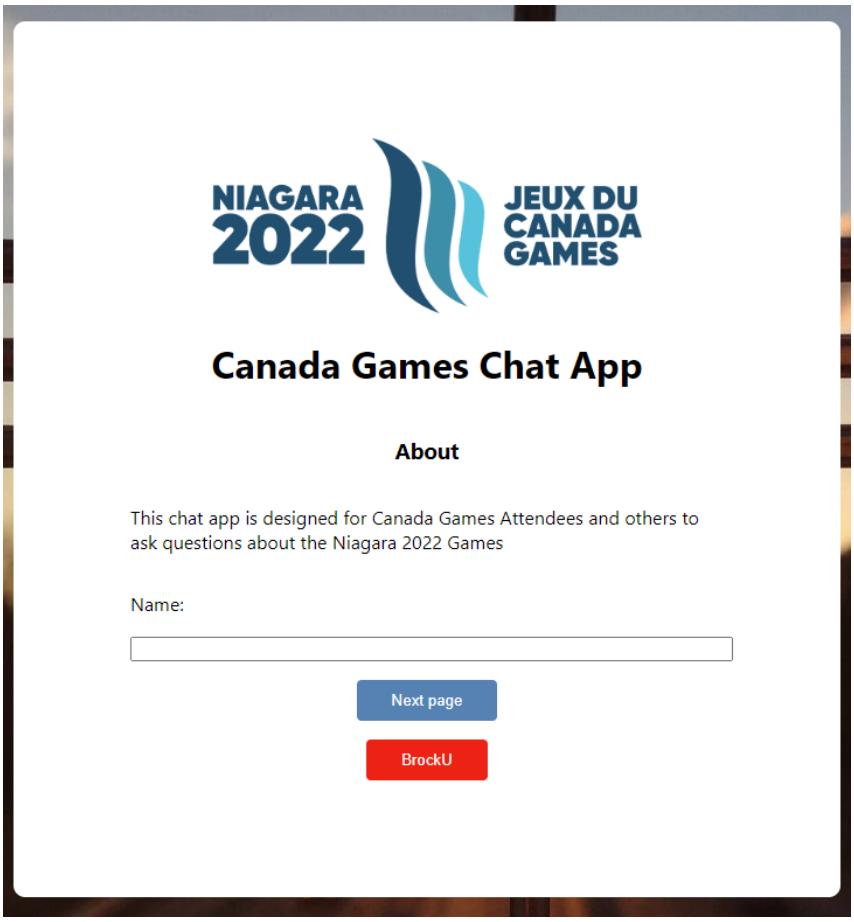

Figure 3 - Canada Games Welcome Screen

Once a user enters a name they should click on the “Next page” button to continue to the chatbot. They will then be greeted with a window similar to the one shown in Figure 4.

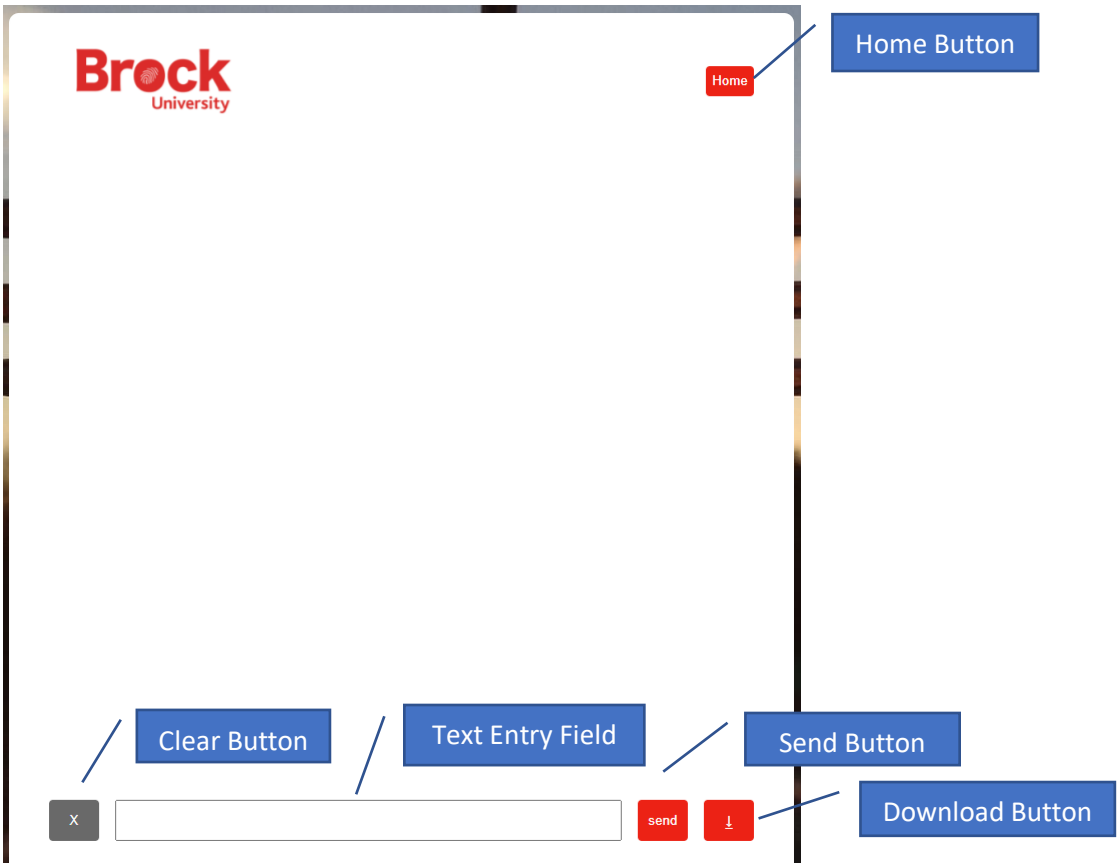

Figure 4 - Brock Chatbot

The user can now type into the text entry field and can either click on the send button or just hit enter to submit their message. The bot will then show an animated ellipse icon as it generates the response. Once the response has been generated it will be sent to the client as shown in Figure 5.

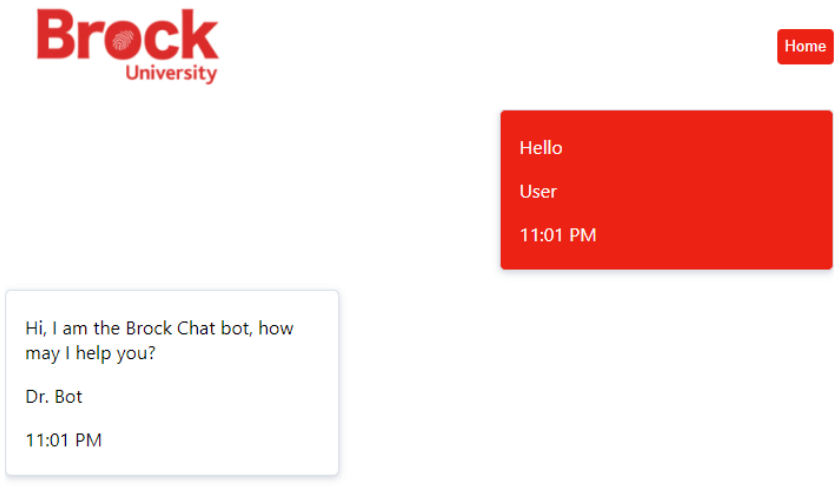

Figure 5 - Brock Bot Response to Hello

As you can see, each message bubble has a user as well as the time that it was sent. This will help clarify when the messages were sent.

Clearing the Text Input

If the user wishes to clear the text input, they may click on the clear chat button as shown in Figure 4. This will reset the text entry field.

Downloading Chat Log

If the users wishes to download the chat log, they may do so by clicking on the download button as shown in Figure 4. This will prompt the user to save a CSV file. From the example above here is the generated CSV when the user requests to download the chat.

body	username	timestamp	ownedByCurrentUser	brockVer
Hello	User	11:01 PM	TRUE	TRUE
Hi, I am the Brock Chat bot, how may I help you?	Dr. Bot	11:01 PM	FALSE	FALSE

Table 1 - Chat CSV

Example Queries

Here are some good example questions to ask the bot to see some of the capabilities of the chatbot.

Due to our limited data, you will only be able to Query the courses offered in the 2022 spring timetable from A-C as well as exams from A-C. This is explained more in the Database section of this document.

Brock University Chatbot

- Hello bot
- Do I need to wear a mask to class
- When is my last lecture
- Who is the prof for cosc 2p03
- When does the course apco 1p50 start
- What are the labs for biol 1p91
- Is there a seminar for CHYS 3P10
- What can you tell me about the accounting program

Canada Games Chatbot

- Greetings
- Where do I find info about the games
- How long till the games start

[The following questions do not have the complete data, but they will still return valid search terms. Once the games are running and there is a database that stores this data the bot can use it generate proper responses.]

- How many medals does ontario have
- How many gold medals does BC have
- When is the next basketball game
- When does new brunswick participate in the swimming event next

Team Process

Scrum

During our project we used the agile method Scrum. This mean that we worked in two-week sprints. Our original goal was to follow the plan laid out in our project proposal as shown in Table 2.

Timetable

Week of	Goals
Jan 17	- Generate user stories - Create Product Backlog - Create Sprint Backlog
Jan 24	- Infrastructure Setup - R&D
Jan 31	- Start Development (First Sprint)
Feb 14	- End of first sprint. - Release of version 1
Feb 28	- 2nd Sprint
March 14	- End of 2nd sprint. - Release of version 2
March 28	- 3rd Sprint
April 11	- End of 3rd sprint - Release of version 3
April 18	- Final Release

Table 2

Our outlined plan was to have 3 main sprints. This would help break up our backlog into three main sections. The first sprint consisted of mainly Research and initial development of the system. This was done so that we would have a greater understanding of the scope of the project and what was needed to be done to create a successful product. The second sprint was where majority of the work would be done. This was where the different aspects of the system would be worked on, tested and implemented. Lastly, the final sprint was where all the different parts of the system; Front-end, Back-end and Database would be put together to function as one whole system.

Additionally, with our three sprints we planned to run our “daily stand-ups” on Monday and Friday of each week. This would be a good time to bring up any issues that we were facing as well as go over our findings and development breakthroughs. We also implemented using Discord as our main method of communication. This allowed us to post research findings as well as have the ability to reach out anytime of the day to our peers.

Sprints

During one of our first meetings, we had a meeting where we came up with the user stories. These were created in our OneNote which we then transferred over to ZenHub which was our Scrum board software. We broke the user stories down into categories to try and cover all the main aspects of the chatbot application. Below are the categories and user stories of our first backlog.

Ask a question

- As a user I want to be able to ask a question.

Reply Stimulus

- As a user I want to be able to see that the chatbot is working on a reply.

Chat Display

- As a user I want to be able to see the last 50 messages.
- As a user I want to access a menu to change settings on the chat bot.
- As a user I want to see a general welcome page with an interface for interacting with the chat bot.
- As a user I want to be able to click a button to clear the chat box
- As a user I want to know if the chat bot can not understand my question

Interaction

- As a user I want to be able to quit the application at any time

Scraper

- As a user I want to be able to know information about Brock courses
- As a user I want to I want to know information about summer games events.

Canada Games Questions

- As a user I want to be able to know information about volunteer opportunities.
- As a user I want to be able to ask questions about Athletes
- As a user, I want to be able to choose to use the chat-bot in either English or French.
- As a user I want to be able to ask the scores for sports
- As a user I want to be able to ask about scheduling of sports

Brock University Questions

- As a user I want to ask questions about Exams
- As a user I want to ask questions about specific Courses
- As a user I want to ask questions the Brock website

General Questions

- As a user I want to ask questions about how to get to places

Portability

- As a user I want to access the chat bot through my phone, computer, laptop, or tablet
- As a user I want to be able to know about questions
- As a user I want to be able to see the log of the chat and be able to save it to my device

Developer

- As a developer I would like to have a preliminary framework set up
- As a developer I would like users to use natural phrases to obtain information
- As a developer I would like to have a preliminary User Interface design

Since this list was created very early on in the development cycle, we added more user stories as the first sprint which was very R&D focused as we understood both the SRS as well as our potential framework and the development that it would require.

Also, in the first sprint each user was tasked with different areas of R&D. The three main areas were the NLP, Docker/Backend, Frontend. Once we had a good week to research, we met up to go over our findings in our daily meeting and then we voted on which solutions we thought would best fit our needs.

The second sprint delt with finally settling on our chosen framework and then starting development on each component. In this sprint we had each of the team members working on different things ranging from the docker containers, Django, databases, frontend and the NLTK. Due to our team having five members we tried to share the load with were people had their strengths.

In the third sprint our velocity started to suffer as due to midterms and other course assignments starting to creep up. This meant that some user stories from the second sprint were caried over to the third. Thankfully we did not realise that we would actually have the time for four sprints which helped our third sprint help catch us up to our original seclude. By the end of the third sprint, we had a simple working prototype of the system.

In the fourth sprint we started to finalize the design and spruce up the front end. Additionally, we also started to work on training our chatbot to answer more questions and support more data. Unfortunately, due to our planning decisions we thought we would have access to more data from Brock or Canada games so we never got a scrapper running so we worked on compiling data manually into a database so that our chat bot would be able to function.

What we learned from Scrum

One main thing we noticed was that due to our initial user stories being somewhat vague we were not able to really complete user stories per sprint. Plus, due to the fact that we did not have a fully working system by the end of sprint 3 it was very hard to complete stories like “As a user I want to be able to know information about volunteer opportunities”. These were all user stories that could only be accomplished once a working system was created. In most agile development cases it works better when you have very basic versions of a software released and then you slowly add more and more features to the application. In our case we finished most of our user stories near the very end of our development cycle.

However, even if our user stories were not completed as quickly as the “scrum” process would have liked each user story had several tasks that were talked about in our daily stand-ups. These tasks were managed by each individual as we used the stories as our main goal but had our own tasks to accomplish those stories. Later on, we learned we could use stories like “As a developer I would like to have an API” which were more development stories than final product stories. This meant we were able to accomplish these user stories at a faster rate during our early development phase. Unfortunately, this methodology was learned when we were already in our later development phase.

Development Process

During the development process we set about to try and stay as organized as possible. The lead to any changes done for any aspect of the system, would have to be done on separate branches in the GitHub, named with the specific feature being worked on. When done with those specific features intended for the branch, we would then create a Pull Request. These pull requests would then need to be approved by another group member to then be fully merged into the main branch. This allowed for our work to be read and approved by other group members, allowing for multiple eyes looking for anything that may stand out. We could then discuss with each other how it works and why it was done in that way. We decided on this as we would like to lock out commits to main directly. This allowed for a more organised and safe way to develop without interfering with each other on the different features of the whole system.

Another system we utilised was creating a #notes-and-resources thread in Discord as well as setting up early a OneNote for the Team. This would hold things like general info, meeting notes, initial product backlog, project documents, and R&D research. This would also hold any helpful links to the tools that we ended up using. Things like video tutorials and helpful links would be stored in either of these two places.

GitHub wikis were also a great tool that assisted us in our development. Due to some of the complexities of our development we were able to create wiki pages that outlined some of the complex set up tasks that were need in our development cycle. Mainly setting up Docker as well as importing our databases. We created these wiki pages to help aid our peers in setting up and configuring the external tools for development. This meant that there would be no setup confusion or misconfigured dev environments for each of our team members.

Testing

Our methodology of software testing was development testing. As we are all coding different parts of the system, we would be testing these changes as we go on. Looking for any bugs or changes that need to be made to further better the pieces. Due to the way we required pull requests to bring in the commits into the main branch, we would then ask each other to look through the changes and test out the feature to see what would need to be changed, and new issues were created to fix them. In the later sprint we then got to a point where the full system had to be tested, which lead us to run system-level testing. This is where when the different components of the system were put together near the end of development to test how well the different aspects of the system worked when put together as one whole working system. We tested things like the API and the communication between the different components of the system.

Design

General Design Decisions

In one of our early meetings as we were discussing possibilities of languages and libraries to set up our framework, we also discussed what type of dev tools we were going to use. When we settled on using the NLTK we decided to use Django as our main backend to keep our language spread to just python. This meant we decided to use PyCharm as our main development environment.

Another early design decision was to use Docker as a way to containerize our system. This was decided on as a way to make sure there every developer in the team would have the same versions of the software. An additional reason was that since this was going to be a web application, Docker would allow for us to scale and deploy the system fairly easy.

We settled on using PostgreSQL as it was a popular free SQL database.

For our front-end we decided on React as it is a very popular JS framework. None of us had worked with react before so we also thought it prudent to learn it as it will be useful later on in our careers.

Framework

After our initial R&D phase was over, we settled on our final framework. This was laid out similar to the framework as shown in Figure 1. We used Django as our backend. This would run the chatbot and communicate with the database. Since both Django and NLTK use python, it was quite easy to configure

NLTK as a Django app. Django also has its own DBMS that server data as models which were quite easy to inject into the NLTK to enable answers to be received from the database. We then would use a REST API to communicate with the React frontend which is a popular JS library.

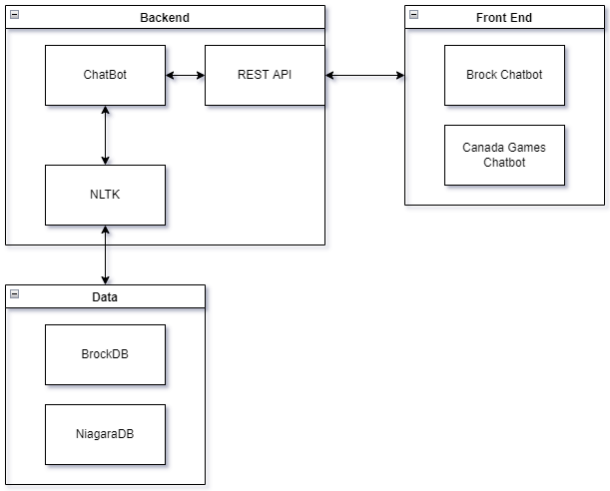

Figure 6 - Framework Overview

Having the framework laid out in such away meant that we were able to containerize our system as shown in Figure 7. We had a backend, web/frontend, BrockDB, and NiagaraDB container.

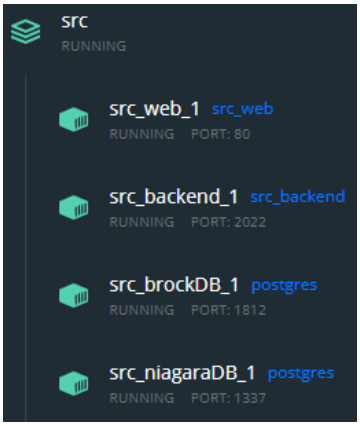

Figure 7 - Docker Containers

NLTK

Training

The NLP is trained using TensorFlow so that it can predicate user intents in a dynamic environment. Training data is consisted of topics (intents) and patterns (possible user inputs). Every topic in the training data corresponds to a bunch of patterns. The patterns are filtered (lemmatized, lowercased, and special characters are deleted) and tokenized into lists of words. Each word list has its corresponding topic. The bot is trained over epochs so that it can get the user's intents as precisely as possible. The more diverse data it has for training, the more likely it will get the true intent of the user.

Predicting user intents

To predict the user's intents, the program first tokenizes the message given by the user into a bag of words, where the words are lemmatized and special characters are ignored. Using the TensorFlow model generated by the training process, the program tries to match possible intents that passes a certain error threshold. The results are then sorted by confidence and the most fit one is picked for the response generation.

Response generation

In cases like greetings or asking for a website, the database part is skipped entirely, as the responses from the bot will not include anything queried from the database. In such cases, a random response is picked from the predefined response list for the topic.
On the other hand, in cases such as asking for information about a course, we first try to get the course code by matching a regex (regular expression) pattern. We then use the course code to query relevant information from the database and finally, compose them into a complete sentence based on how the user asks the question.

Docker

As stated in the framework section, we split the application into four main containers backend, web/frontend, BrockDB, and NiagaraDB, respectively. Each of the containers required their own Docker file. The main goal with configuring our Docker containers was to require the least amount of manual work. This meant that we had to figure out a way to automatically migrate the database models when firing up the containers. Since we were working with a multi-container set up, we needed to configure the system to start up the database first before the backend. Otherwise, there would be race conditions that would cause errors. Additionally, we had to set up inter-container communications by configuring ports so that the containers could talk to one another.

REST API

As the backend was built on Django. We used a REST api library for Django to facilitate the requests. REST api's are extremely intuitive and simple ways of interacting with clients on a variety of devices. They work on the premises of defined endpoints and request types. The endpoints are essentially routes which can be accessed by a client, and these routes define the structure of accessing data from the backend. For example. The backend may define a route at www.IP_ADDRESS_OF_SITE.com/api/course (this is a fake url), and this route has programmed requests that it can handle. It may accept a GET request with a particular format for the body, which can then be processed and have data sent back to the client. The type of request tells the back end what the client wants to do. Get requests mean the client wishes to access data, Post requests usually provide data for the backend to handle, it is up to the back-end engineers to choose the request type most appropriate for the functionality. In our version of the api, there is a route to get a response from the chat bot. This route expects json in the body of the message, where the client's message is stored. This is then processed by the backend and sent to be processed by the NLP. The response is then sent back to the client to be displayed.

Since this API is technically public, any client that wishes to support the functionality can follow the rules of the endpoint and receive the full functionality. This means apps on any native platform can interact with the API. If required however, the api can be secured with access tokens or account authentication to only allow specific clients to communicate.

Front End

The front end was built entirely on the React Framework. The React framework is a popular modern framework for building web and native applications quickly. The framework works on the basis of a rendering engine and a set of user built and provided components. The idea behind react is to create components that represent many of the views in your application, and then render them at will using the rendering engine. These components are quite versatile and can accept data, react to events, and update at will. The first iteration of our front end was designed as a barebones implementation of all the core requirements. This included a text box for typing, messages, and two different pages (one for brock and one for the summer games respectively). This allowed us to get the basic idea for what the webpage would look like, what features to implement, as well as how to integrate it with our back end.

After a sprint meeting with the TA some new changes were suggested, they were to redesign the UI somewhat to have a cleaner more refreshed look. The next sprints were spent on implementing the remainder of the core features such as the downloading of the chat logs, the "…" to let the user know the response was being processed, as well as the full interaction with the API. These sprints also were centred around a new UI. The API interaction is quite simple, as a message is sent to the endpoint on the backend, the format of this request is simple as it stores the message in json within the body of the request. The front-end then waits for a response, during this waiting period it shows the typing animation to the user. When the response arrives, the front end reads the json body and creates a new message bubble to display to the user. This functionality is also repeated on the summer games page.

Database/Scraping

BrockuDB

In the early stages of this project, we decided because of project creep we were just going to manually create a database to get a preliminary chatbot working. That choice was made with the hope that we would eventually receive data from our client later on. This meant that the data the bot is currently able to query is only from the 2022 Spring Schedule on courses from the A-C range. On the exam side of things, the bot is able query the 2022 exam schedule from the A-C range.

Easy DB Access/Updates

Even though the bot does not have a scraper, the bot is able to fully comprehend the users question and provide an answer based off of the database that it has. In the backend there is at Brock admin page at /admin that allows for us to manage the database for the chatbot as shown in Figure 8. Due to security reasons we have disabled the admin ports on our production server.

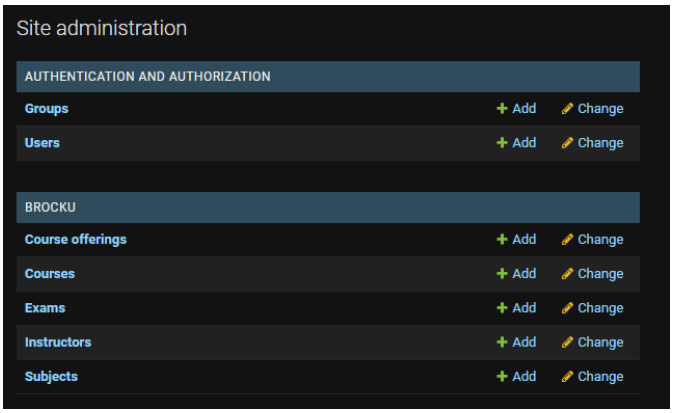

Figure 8 - Database Management

This site administration page allows for us to mange users and the DBs. This means that if we were provided with all the courses, it would be very easy for us to import the courses into chatbot.

CanadaGamesDB

Due to scope creep, we were not able to put any data into the Canada games DB. However, just like with the Brock side of things, if we were able to get easy access to the data our bot would be able to query the database quite easily. As you can see with the answers from the Niagara chat bot. It is able to understand the question. It will respond with what it understands. For example, the question "How many medals does Ontario have?" will return "scoreON". If we had access to the scores, we would be able to return the score of Ontario.

CSVs

If you would like to view the current data that is in our chatbot application, it is viewable on our importing data wiki. There are links the CSVs that contain the database we used to produce our chatbot application. [Data-Import Wiki]

Installation and Running

If you would like to run this application on your own machine our GitHub wiki is very helpful. https://github.com/TheGrai/COSC-4P02/wiki This project is very simple if you already know docker. You should be able to just create a Docker Cloud instance from running the docker-compose file. Please note, that in order to have the data to make the chatbot work you will need to import the data into the database. There is a provided wiki on how to accomplish that.

If you are new to Docker, the easy way would be to install docker desktop. Once installed open the GitHub project in IntelliJ PyCharm Professional and then follow the wiki on the GitHub.

Contributions

These are the general Contributions of our team members. More details are in the progress reports as well as all the git logs in GitHub.

Aldric Joya:

- Front-end and NLTK Specialist
- Front-end design and UX guru
- NLTK – Niagara games specialist.
- Worked on more accurate NLTK response generation.

Arin Yaldizciyan:

- Front-end Specialist
- Helped with initial infrastructure design ideas
- Research on frameworks for use on backend, database, as well as front-end
- API structure and message protocol

Grant Ferrier:

- Full-stack developer
- Configured API from frontend to backend to NLTK.
- Helped with Docker and Framework setup and System configuration.
- Assisted with NLTK response generation.
- Team Support

Justin Zhang:

- NLTK Specialist
- Configuration of Preliminary NTLK settings.
- Worked on NTLK processing to work with small brock university sample database.

Sabih Zubair:

- Database Design
- Gathered Brock information

Thanikash Kanagaratnam

- Full-stack developer
- Configure docker container implementation of the Cloud Container Framework.
- Configure intercommunication between docker containers in the Docker instance.
- Database configuration and creation.
- Worked on more accurate NLTK response generation
- Team Support (wiki guru)