Brock University Chatbot
Final Report

Marmik Bhatt 5939780 mb15nf@brocku.ca **[Leader]**
Tom Wallace 6482558 tw18dw@brocku.ca
Jakob Shortell 6280275 js17ya@brocku.ca
Aedel Panicker 6480297 ap18pu@brocku.ca
Hyejin Kim 6823116 hk19xt@brocku.ca
Liam Mckissock 4447751 lm09sr@brocku.ca
Lucas Kumara 6235295 lk16ce@brocku.ca

April 30th, 2022

Table Of Contents

1. Design Specifications and Methodologies

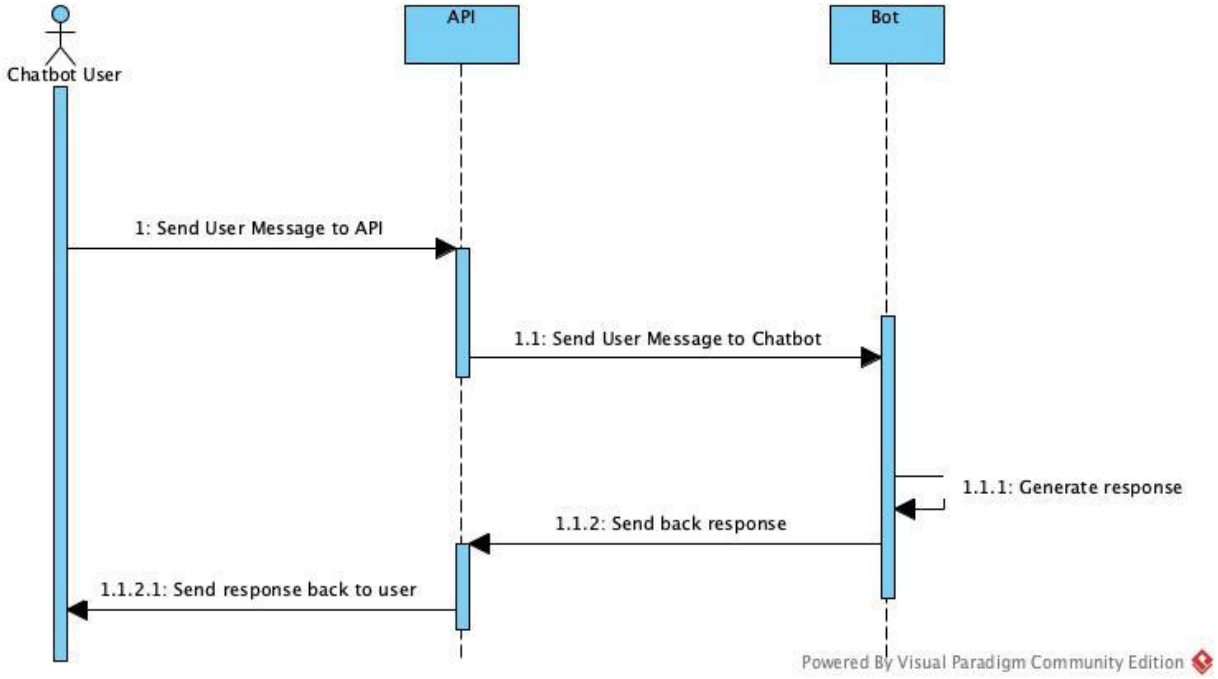
The project was followed on the basis of the scrum methodology. So this revolved around getting a product backlog set up based on the requirements document that was provided to us. Once this aspect was followed through we set up initial sprint backlogs based on components we felt were critical to the functionality of the project, such as setting up the front end, getting the basics of the NLP working along with setting up the database and scrapping. The process was broken down into 2-week sprints in which meetings were conducted on Tuesdays and Thursdays at 6:30 pm EST, which covered things such as progress, shifting around sprint backlog items and voicing any concerns the development team has encountered.

Sprints

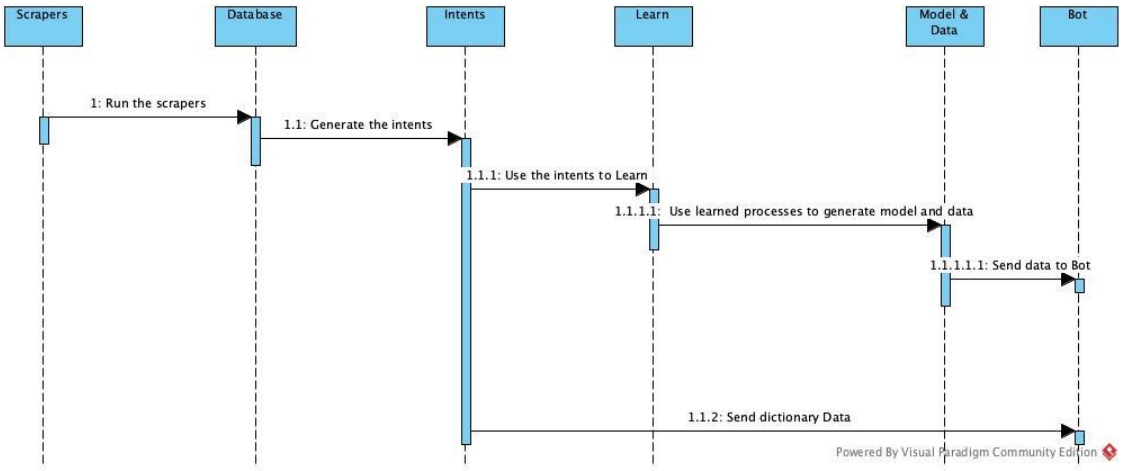
When beginning development on a given sprint, we divided up work based on what each group member was comfortable doing, as well as taking into consideration what members had already contributed. This helped ensure that tasks were given to the group members who were likely to be able to complete them, and helped cut down on time spent fixing issues while allowing more time for every member to complete their own tasks. In the first sprint, we didn't have any prior contributions to go off of, so members were assigned tasks from the backlog based on their ability as a programmer. Towards the last few sprints, our sprint backlogs got relatively small, so some members were assigned to help with reports and other non-development related tasks to remain efficient.

2. Diagrams

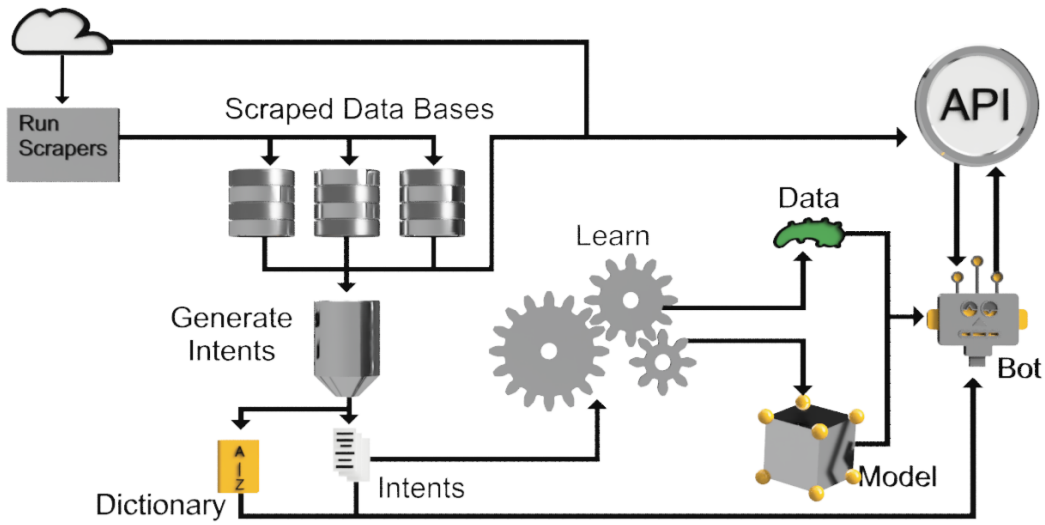
General Chatbot Sequence Diagram



Backend Sequence Diagram



Backend Diagram



3. Updated Requirements

Overall Description

A lot of the items were built into our initial backlog, as many of these fit under certain requirements that were mandatory.

- Functionality
    - The chatbot is accessible through a web browser on most modern devices, including mobile phones, laptops, tablets and desktop computers.
    - The chatbot accesses external data through an internet connection to the main server and database.
    - This chatbot software is capable of having questions asked to it in the form of text or speech to text (built into most modern devices), and formulating a text-based response.
    - The software has an initial popup which makes its purpose clear to the user.
    - The software, as a web application, can be exited by closing the tab or window.
    - Should the chatbot be unable to answer a specific question due to complexity or any other reason, it will provide a link for the user to ask their question on Google.
- Brock-Specific Functionality
    - Users are able to access basic information about any course offered at Brock, including:

        - How the course is delivered (i.e.: 3 hours of lecture, 1 hour of labs per week), along with information on which instructor delivers each section.
        - A description of the course.
        - Course prerequisites.
        - Class times.
            - Note: While the scraper does retrieve the room, the chatbot is not currently configured to provide it as part of the response.
    - Important dates, such as withdrawal dates, are accessible.
        - Though currently asking for important dates only provides a link to Brock's Important Dates list, the scraper is functional and the groundwork exists for a more substantial implementation.
    - Transportation information is accessible.
    - General information about the Brock University campus and recent events are accessible, as well as any information otherwise accessible through the Brock University main page.
- Design Constraints
    - Language
        - The application and chatbot are available in English; however, it is designed so that it will be possible to add support for multiple languages.
    - Browser Support
        - The application functions on all frequently-used browsers, such as Google Chrome, Microsoft Edge, Firefox, etc. It has not been thoroughly tested on any non-standard browsers, so it is unknown if it will work or not.
    - Reasonably Accurate Language Processing & Responses
        - The chatbot will analyze keywords and attempt to supply a reasonably useful answer, regardless of any misspellings or poorly worded queries.
        - In the case the chatbot is wholly unable to determine what is being asked, it will supply a link to ask the question on google.
            - This is certainly not the ideal functionality, but the language processing system used makes it very difficult to work with incomplete queries.
    - Simple and Intuitive UI Design
        - The interface used for the text chat resembles those used for text messaging on mobile phones.
        - Users will be able to quickly determine how to send messages.
- User Documentation
    - Upon accessing the webpage, a popup occurs detailing types of information accessible via the chatbot.
    - The text box for user input shows default text saying "Type your question here", showing users where they must ask their questions.
- Assumptions and Dependencies
    - For the chatbot to function most effectively, the Brock University informational web pages it sources its information from must be accessible.
        - Most Brock-related information is stored in a database, updated whenever the scrapers are run (ideally daily). This information will be accessible if the websites are unavailable, but may not be up to date.

        - Information such as news/events at Brock and St. Catharines are retrieved live, and will not be accessible if the necessary web pages are down.
    - As a webpage, the chatbot will not be accessible without an internet connection.

Interface Requirements

- User Interfaces
    - Starting Menu
        - When opening the webpage, a popup is shown that details the types of interactions available to the user. It also serves to inform the user that all information given by the chatbot is retrieved from official Brock websites.
    - Usage Screen
        - Pressing the “I Understand” button on the starting popup brings the user to the primary screen of the application.
        - The text display between the user and the chatbot is visible, displaying the message history that the user may scroll through. The newest messages are displayed at the bottom. Excluding long multi-line messages, the screen can display approximately 7 messages at once.
        - Each message bubble indicates the time the message was sent, and whether it was sent by the user or the chatbot.
    - Input Bar
        - The input bar is located at the bottom of the screen. When not in use, it has text directing the user to ask their questions using the bar, with this text being faded so that it is distinguishable from typed text.
        - Any text inputted by the user can be sent as a message either by pressing the Enter key, or by pressing the Send button located to the right of the input bar.
            - When the message is sent, the text is removed from the bar. The text input remains engaged, so the user does not need to click on the bar again to enter additional messages.
        - For users accessing the website using a mobile device, tapping on the input bar will bring up the device’s onscreen keyboard. The position of the input bar will automatically adjust, so that it remains visible while the user types their message.
    - Button Menu
        - There are three buttons located at the top right of the screen.

            - The leftmost button is the ‘Help’ button, which reopens the initial popup displayed when the page was first opened.
            - The middle button copies the existing chatlog to the user’s clipboard in a text-based format.
            - The rightmost button opens a secondary menu allowing for the manipulation of certain settings.
                - Currently, the only option available is the option to change the webpage’s language, with the only language available being English. Were further languages to be added, they would be accessible through this menu.
- Hardware Interfaces
    - Computers
        - In this document, defined as any device with a physical keyboard and either a mouse or a trackpad.
        - The physical keyboard is used for the entry of text, with the “Enter” key being one option for the function of sending the message.
        - The mouse or trackpad is used to interact with the text box and buttons using the left mouse button. The right mouse button does not have any special uses.
        - The scroll wheel on the mouse can be used to scroll upwards or downwards through the message history.
    - Touchscreen Devices
        - Any functions that would be performed on a computer using the left mouse button are instead performed by tapping on the screen.
        - The device’s on-screen keyboard is treated identically to a physical keyboard in terms of input and functionality.
- Software Interfaces
    - The software works on a variety of operating systems, including macOS, Windows 10, Windows 11, iOS, Android, and Chrome OS. As a webpage, any incompatibilities are more likely to be browser-related than they are to be OS-related.
    - The software is implemented using a JavaScript (React)-based webpage, with a Python (Flask)-based backend.
- Communications Interfaces
    - The website maintains a consistent connection between the user and the host.
    - Most scraped website data is kept in a database on the hosting server. When the user asks a question related to this data, the information is retrieved from the server and formulated into an appropriate response.
    - The database is updated by running the code ‘run_scrapers.py’, which sequentially runs each website scraper and applies any updates to the database.

    - Some information (News and Events) is scraped upon user request, meaning it will not be retrieved if the websites they are sourced from are unavailable.

System Features

- Ability for the User to Ask a Question
    - The chatbot provides a method for the user to input a question (Typing in the input bar).
    - The chatbot allows the user to send the typed question, with some indication that the message has been sent successfully.
- Chatbot Generates an Appropriate Response to Any User Inquiry
    - After a message is sent, there is an indicator that the chatbot is preparing a response.
    - The chatbot system analyses the user's message, identifies keywords, and determines the information that must be relayed to the user.
    - The information is formulated into a properly worded response, and returned to the user as a message from the bot.
    - If the chatbot cannot understand what the user is asking for, or if the information is available, it will instead send a message indicating the issue.
- Opening Screen
    - The chatbot uses a popup upon opening the webpage that indicates what website the user is looking at, and what they will be able to do with it.
    - The popup has a button that, when clicked, brings the user forward to the main page by closing the popup.
- Main Screen
    - The main screen of the chatbot includes a chatlog, a user input bar, a menu composed of various buttons, and a button for sending an inputted message.
    - These elements are laid out according to generally accepted conventions seen in other chat applications.
        - The chatlog is in the center of the screen, with messages received from the chatbot on the left, and user-sent messages on the right.
        - The input bar is at the bottom of the screen, with the submit button to the right of it.
        - Any additional buttons are on a bar located at the top of the screen.
- Questions Which the Software Must Be Capable of Answering - Brock
    - The chatbot is able to tell the user information about exams.
        - If a particular course has an exam
        - If that exam takes place online or in-person
        - The room and time of the exam
    - The chatbot is able to answer questions about courses.

            - Note: Though this information is retrieved by the scrapers, the chatbot's language processing may need to be updated in order to properly supply this information.
        - Must be able to supply the course description
        - The duration of the course
        - Who teaches the course
        - If there are any labs, seminars or tutorials
        - Any prerequisites
    - The chatbot is able to answer questions about programs.
        - Whether or not a program exists
        - Description of a program
        - Requirements for a program
    - The chatbot is able to locate webpages on the Brock Website and supply them when asked, such as the Important Dates page.
- Questions Which the Software Must Be Capable of Answering - General
    - The chatbot is able to answer questions about transportation in the Niagara region
- Chatbot Responsiveness Indicator
    - When a user sends a message to the chatbot, a chat bubble will immediately appear with the user's message, with the input bar being cleared. This tells the user that their message has been sent properly.
    - In the time between the user's message being sent and the chatbot's response, a slowly blinking dot appears in the space where the chatbot's message would appear, to indicate that the bot is processing and will soon return a response.
    - This dot is animated, and disappears once the chatbot's response is sent.
- Menu
    - The button represented by a gear/cog icon on the top right of the screen opens a secondary menu allowing for the manipulation of certain settings.

        - Currently, the only option available is the option to change the webpage's language, with the only language available being English. Were further languages to be added, they would be accessible through this menu.
        - This menu appears as a dropdown originating from the icon it is accessed from.

- Chat Log Copy

    - This feature allows the user to copy the chat history to their clipboard and paste it wherever they choose.
    - This is performed using the middle button of the three buttons shown above.
- Clear Input Button
    - This feature allows the user to delete any text they have typed in but not yet submitted. It appears as an ‘x’ on the rightmost side of the text input box, next to the submit button.
    - This feature is only available when accessing the website through a mobile device.

Other Non-Functional Requirements

- Performance Requirements
    - It is recommended that user interface interactions should take roughly 0.1 seconds; All interactions the user can perform will give some form of instant feedback.
    - It is recommended that more intensive tasks should take no more than 10 seconds to provide a response; All responses from the chatbot will generally take around 2 seconds to be received.
- Safety Requirements
    - There is little to no safety risk to a user, as there are no physical actions that must be taken, and does not involve any systems or information that could lead to damage or harm for the user or its system.
- Security Requirements
    - Operation of the system does not require any user data, nor does it permanently save any user data to the server. As such, there are no notable security risks involved in usage of the chatbot.
    - Conversely, user data will not be sent from the chatbot to another user.
    - The only data that should be preserved is any changes in the options menu.
- Software Quality Attributes
    - Adaptability
        - The system is able to be updated with new sources of information by adding new scrapers and new Intents to the natural language processing.
    - Availability
        - The system is able to be accessed by most devices that are able to use an internet browser.
    - Correctness
        - Any data returned by the system is retrieved directly from Brock’s systems without any alteration, to be as accurate as possible.
    - Flexibility

        - The system is reasonably able to understand user requests even with the presence of typos or grammatical errors, unless they are particularly egregious.
    - Interoperability
        - The system is able to efficiently and effectively retrieve data from Brock's websites.
    - Maintainability
        - The system is reasonably modular, meaning it is able to be updated with new scrapers, languages, and other features without a significant amount of trouble.
    - Portability
        - The system is likely to work on most commonly used operating systems and browsers.
    - Reliability
        - There is no situation where a user's actions will cause the system to crash, freeze, or become fully unresponsive when using the system as normal.
    - Reusability
        - The software is designed in a modular way, with components having the potential to be reused for different purposes, such as a chatbot for a different topic.
    - Robustness
        - The software is relatively resistant to purposeful attempts to 'break' it.
            - If disconnected from the internet, the webpage will take time attempting to send the message before saying "The chatbot is taking longer than expected."
            - If the user uses strange text, the chatbot will treat it as any other non-understood message.
            - If the user sends multiple messages, they will be answered in order of sending.
    - Testability
        - The components of the system are able to be tested separately or together.
        - Tests for the chatbot system are performed by inputting a message with expected output, and ensuring that this output is what the chatbot responds with.
- Business Rules
    - Because the software has no system for users to login or provide any other authentication, there is no need for any 'administrator permissions' or locking of features/testing.
    - Although the released system will function identically for all users, regardless of developer or Brock status.

Other Requirements

- Data Collection and Database Requirements
    - As previously noted, the information provided by the software is constantly updated by HTML scrapers which retrieve it from Brock's various websites on a regular interval, then save it to a database.

Development Framework and Sub-Projects

The development phase of the software can be broken down into sub-projects that can be worked on amongst multiple parties, then interconnected when complete.

- Main Input and Output
    - Comprises the visuals of the webpage, user interaction, and delivery of responses from the chatbot to the user.
    - Communicates a user's inputted string to the Language Processing system, and receives the response from the Response Generation system.
- Language Processing + Response Generation
    - The system by which the software interprets a user's inputted message, and chooses an appropriate message format, and determines the information that must be retrieved for the response. With this information, the system will formulate an return the response.
    - Receives the message from the Main Input and Output System, requests and receives information from the Data Collection and Database system, and sends the generated response back to the Main Input and Output System.
- Data Collection and Database
    - The system which collects and maintains up-to-date information on Brock using HTML scrapers.
    - Receives and fulfills requests for information from the Language Processing + Response Generation system.

4. Manuals

Introduction

The following manuals are intended to give detailed information about how to run and deploy the various parts of the chatbot application to make it easy for anyone who is interested to get it running. These instructions are accurate as of April 2021 and have been verified to work on Windows 11, Ubuntu 20.04, and both Intel-based and Apple Silicon Macs running macOS Monterey. For Apple Silicon Macs you will likely need to do some extra steps to get the project to run but we have included the instructions to fix the issues that we encountered.
This manual includes instructions on how we recommend setting up development and production environments and instructions on how an end user is expected to use it.

Development (Technical) Manual

Note: System Setup and Python Environment Setup should be completed prior to attempting to run any parts of the program and any components of the application should be run inside the python environment

- System Setup
    1. Clone or download the GitHub repository from https://github.com/jakobshortell/COSC4P02Chatbot
    2. Install Python (We used python 3.9)
        - Installation instructions can be found at https://www.python.org/
    3. Install Nodejs (We used version 16)
        - Installation instructions can be found at https://nodejs.org/
        - For macOS and Linux this can also be installed using homebrew (More information can be found at https://brew.sh/) by running the command brew install node
- Python Environment Setup
    1. Open a terminal and go to the api directory (COSC4P02Chatbot/chatbot/api)
    2. Create a new python virtual environment: python3 -m venv env
    3. Start the virtual environment
        - Mac and Linux: source env/bin/activate
        - Windows: .\env\Scripts\activate
    4. Install the requirements: pip install -r requirements.txt
        - For Apple Silicon Macs you will need to do some extra steps to make this work (Note: homebrew is required for these instructions)
            1) Install the TensorFlow dependencies (more information can be found at https://developer.apple.com/metal/tensorflow-plugin/)
                a) Run: conda install -c apple tensorflow-deps (Instructions for installing Conda can be found at the link above)
            2) Install tensorflow/io
                a) Install wheel: pip install wheel
                b) Clone the git repository: https://github.com/tensorflow/io
                c) Enter the io directory
                d) Run: python3 setup.py -q bdist_wheel --project tensorflow_io_gcs_filesystem
                e) Locate the .whl file in the dist directory
                f) Run: python3 -m pip install --no-deps <the .whl flies name>
                    i) Eg. python3 -m pip install --no-deps dist/tensorflow_io_gcs_filesystem-0.25.0-cp39-cp39-macosx_12_0_arm64.whl

g) Go back to the api directory

3) Install requirements for h5py
    a) Install pkg-config: brew install pkg-config
    b) Run: brew install hdf5
    c) Run: export HDF5_DIR="$(brew --prefix hdf5)"
    d) Run: pip install --no-binary=h5py h5py
4) Edit the requirements.txt file
    a) Replace: tensorflow==2.8.0 with tensorflow-macos
    b) Remove: tensorflow-io-gcs-filesystem==0.24.0
    c) Remove: h5py==3.6.0
5) Install tkinter
    a) brew install python-tk@3.9

- Run the Chatbot (Flask and React)
    1. Run Flask
        - Inside the api directory run: flask run
    2. Run React
        - Open a second terminal and from the chatbot directory (COSC4P02Chatbot/chatbot/)
        - Run: yarn install
        - Run: yarn start
- Scrapers

  The scrapers are automatically run if a request is made to an empty database table so follow these instructions if you want to manually run the scrapers or update the database.
    1. Inside the api directory with the virtual environment active run: python3 run_scrapers.py
- Training the AI

  Run this if you need to retrain the AI that is responsible for processing user input
    1. To generate the intents file run: python3 generate_intents.py
    2. After the intents file has been generated, retrain the bot by running: python3 learn.py
- Testing

  The testing harness automatically starts the bot so to run the testing harness only need to run the one program.
    1. Run: python3 tests.py
- Fixing errors you may see:
    1. PyEnchant error: "*The 'enchant' C library was not found and maybe needs to be installed.*"
        - Instructions for installing the library can be found at https://pyenchant.github.io/pyenchant/install.html

        - Note: If you are trying to install the library using apt run the command apt-get install libenchant-2-dev
    2. *Optional:* If you have an Nvidia GPU and wish to use GPU acceleration for TensorFlow you will need to install the CUDA Toolkit from https://developer.nvidia.com/cuda-downloads. Otherwise just ignore the errors and TensorFlow will automatically use the CPU instead.

Production (Installation) Manual

To make deployment as easy as possible the repository includes docker files for building and starting the containers for both the React and Flask parts of the bot. We used docker-compose to make it so the application could be built and run with minimal to no configuration needed.
The docker-compose.yml uses chatbot/Dockerfile.api and chatbot/Dockerfile.web to create the containers.

- Local
    1. Install Docker: https://docs.docker.com/get-docker/
        - On Linux Docker Compose needs to be installed separately: https://docs.docker.com/compose/install/
    2. Open a terminal and go to the base directory for the chatbot (COSC4P02Chatbot)
    3. Build the Docker containers by running: docker-compose build --no-cache
    4. Start the containers: docker-compose up
        - To run in headless mode use: docker-compose up -d
- Server deployment

  In the bots current state hosting options are limited due to the api needing over 8 GB of memory during initialization and over 5GB while running and docker-compose does not currently support using system swap space.
  We were successful in running the bot using Digital Ocean but it should run on services like Oketo if the memory usage was improved.
    - Digital ocean
        1. Create an account
        2. Create a droplet
            - Under choose an image go to marketplace and select Docker on Ubuntu
        3. Under choose a plan select memory-optimized and select the 16GB plan
        4. Then select create droplet
        5. Once the droplet has been created and is running open the web console or login via ssh
        6. Clone the COSC4P02Chatbot git repository
        7. Enter the COSC4P02Chatbot directory
        8. Build the Docker containers by running: docker-compose build --no-cache

9. Start the containers: docker-compose up
    - To run in headless mode use: docker-compose up -d
10. The bot can now be viewed by going to the droplets IP address

- Oketo

  As stated earlier the bot can not currently run on Oketo due to them not having any options with enough memory but we have included these instructions incase the bot is optimized in the future because deployment on Oketo is free and easy to use.

  1. Create an Oketo account
  2. Select Launch Dev Environment
  3. Select the github tab
  4. Paste the repository URL
  5. It will automatically do the rest and give you a URL for the bot

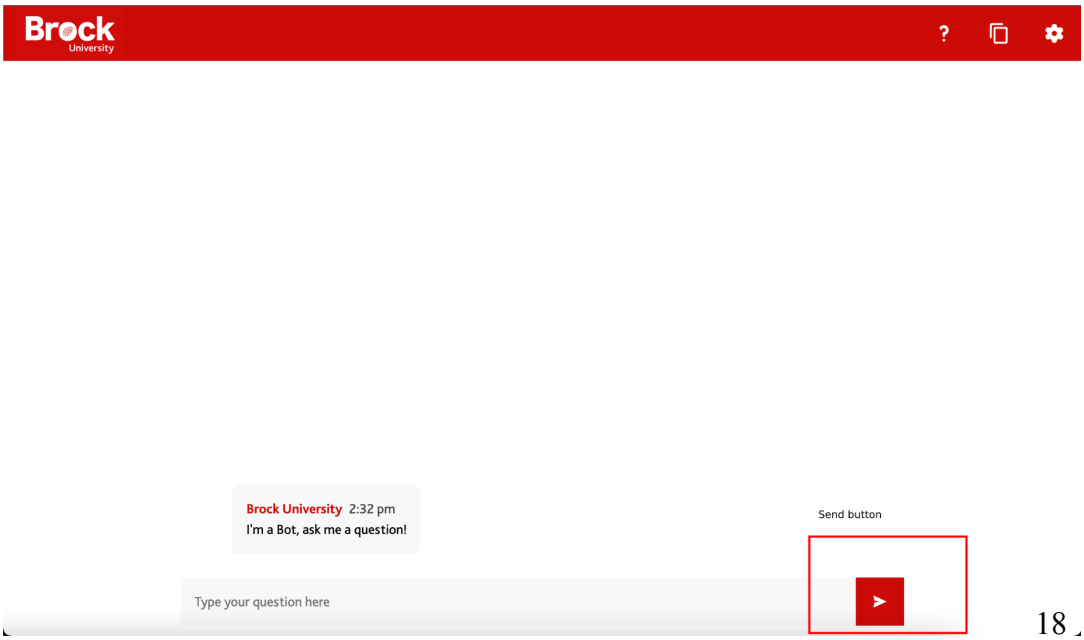
End-User Manual

Introduction: The Brock University Chatbot is hosted for ease of access to the end-user. So no installation needs to be done for users to access the website as long as they have access to the internet.

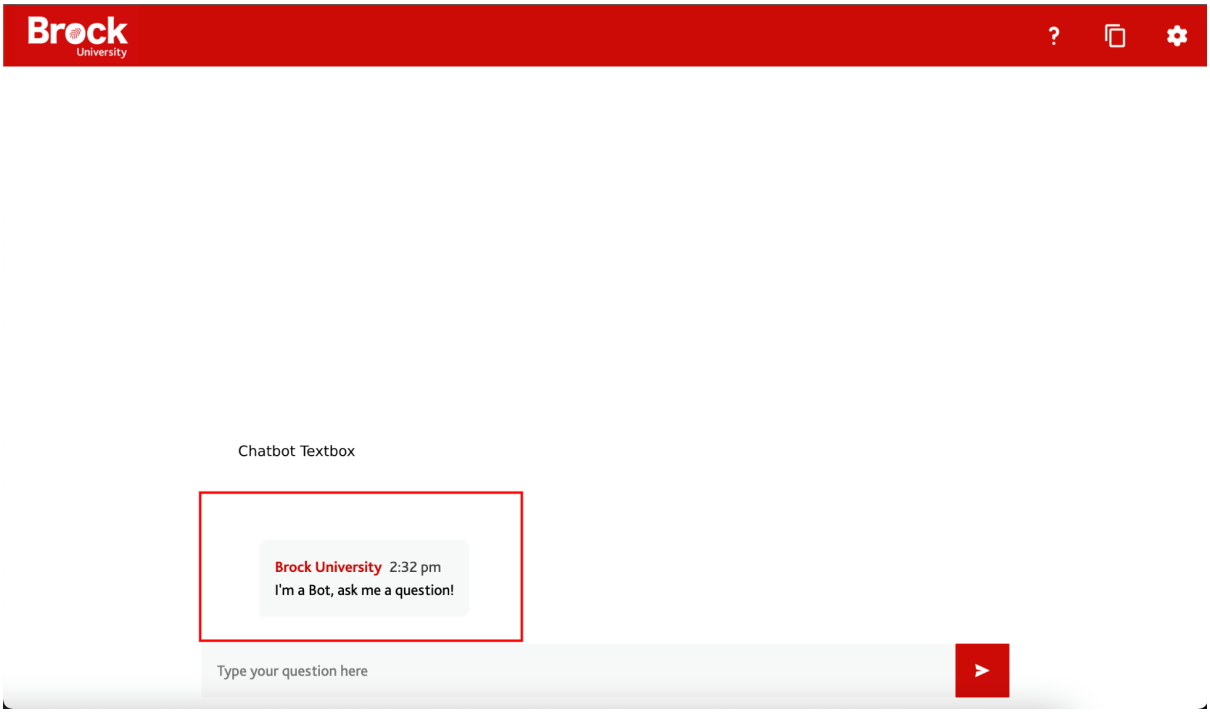
The link to the website is as follows: https://brockuchatbot.jakobshortell.com/ (This link will only be active until the end of May 2022)

In the following, we will be going into the other components within the chatbot, that the user can refer to if need be. Within this section, we will be going over the specific layout by labelling each major component within the chatbot.

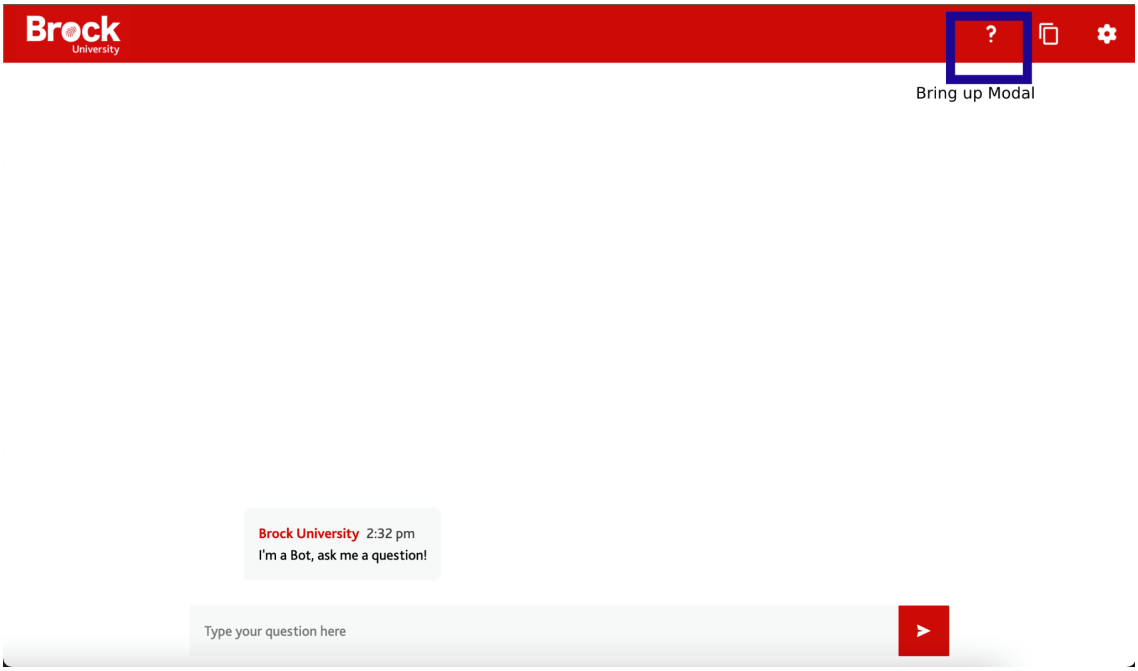
1. This represents the send button.



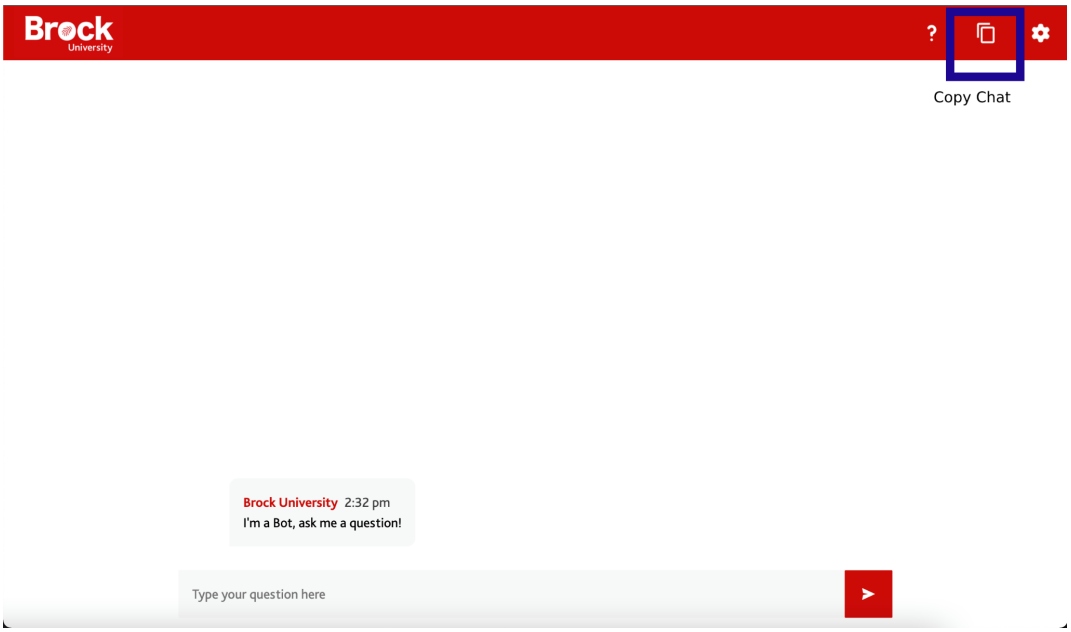
2. This represents the Chatbot text button.



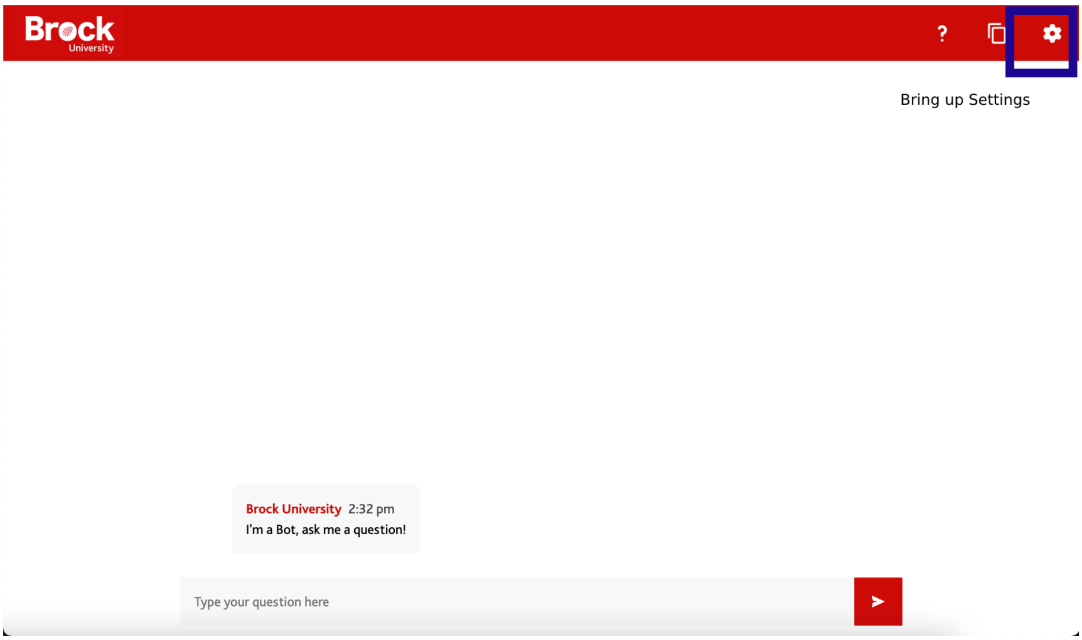
3. Brings up Chatbot information.



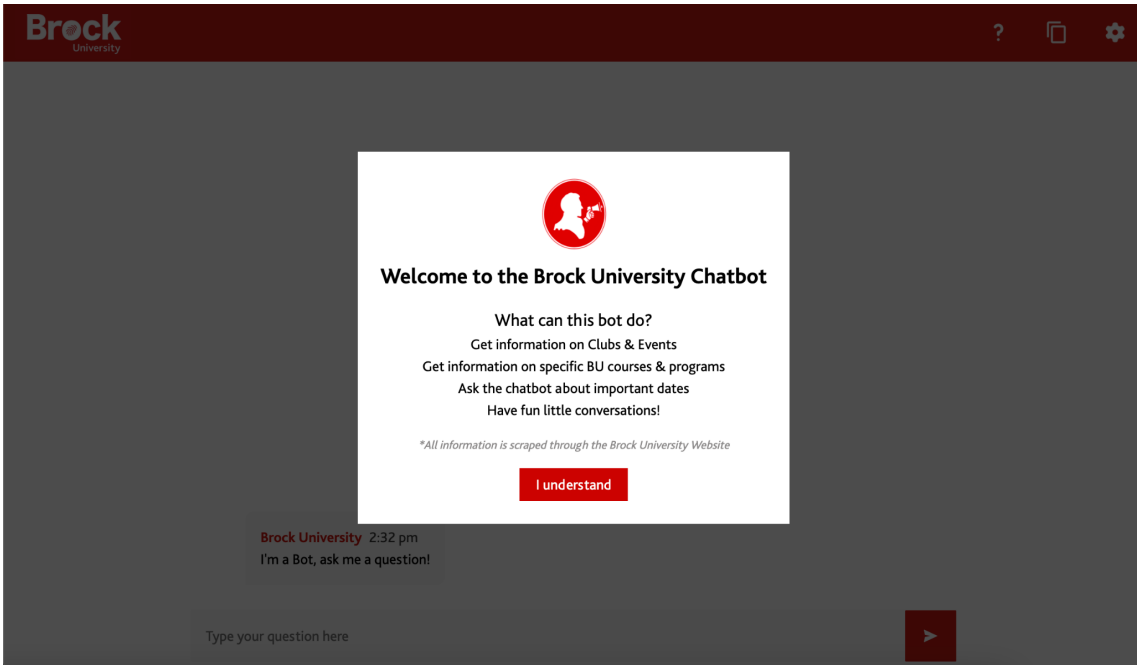
4. Brings up Chatbot Copy.



5. Brings up Chatbot Settings

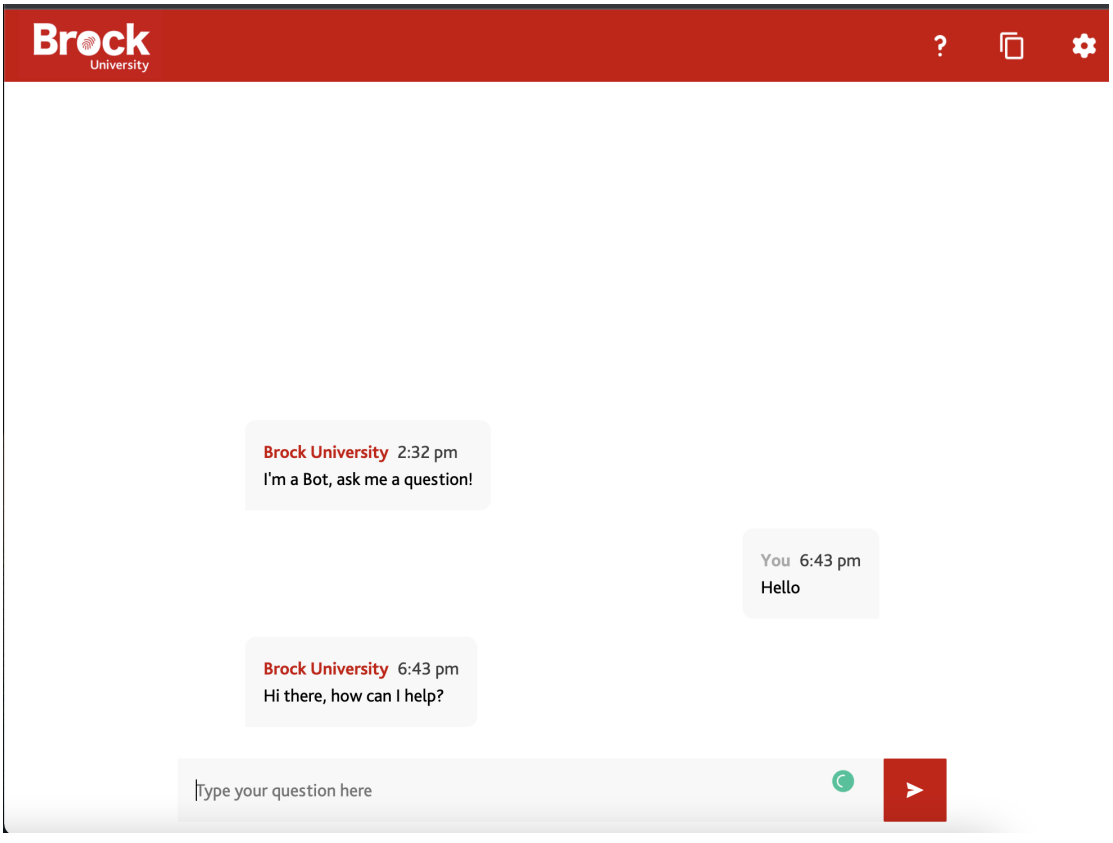


6. The Instruction Modal

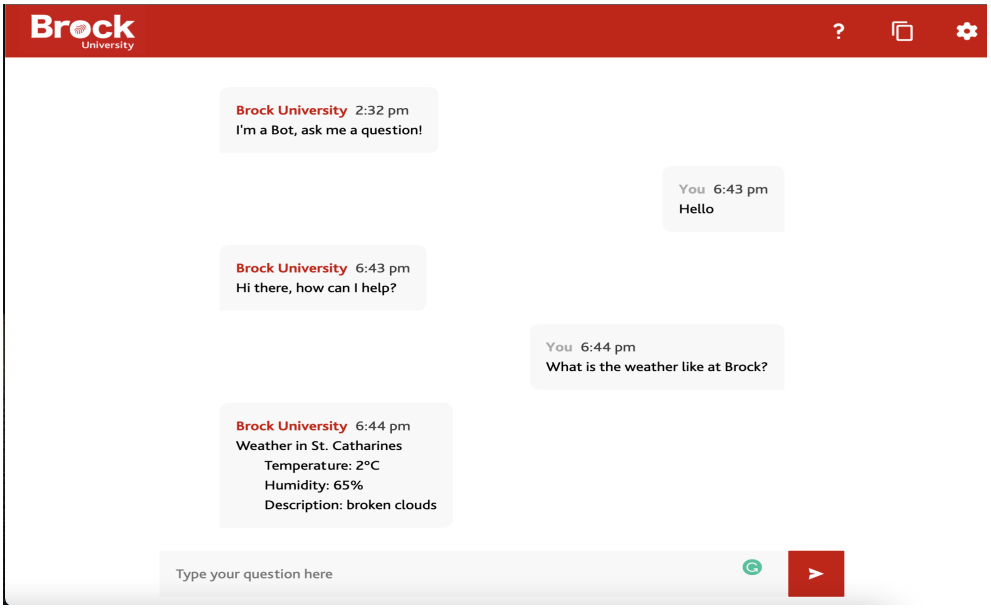


Now we will provide screenshots of the chatbot in use…

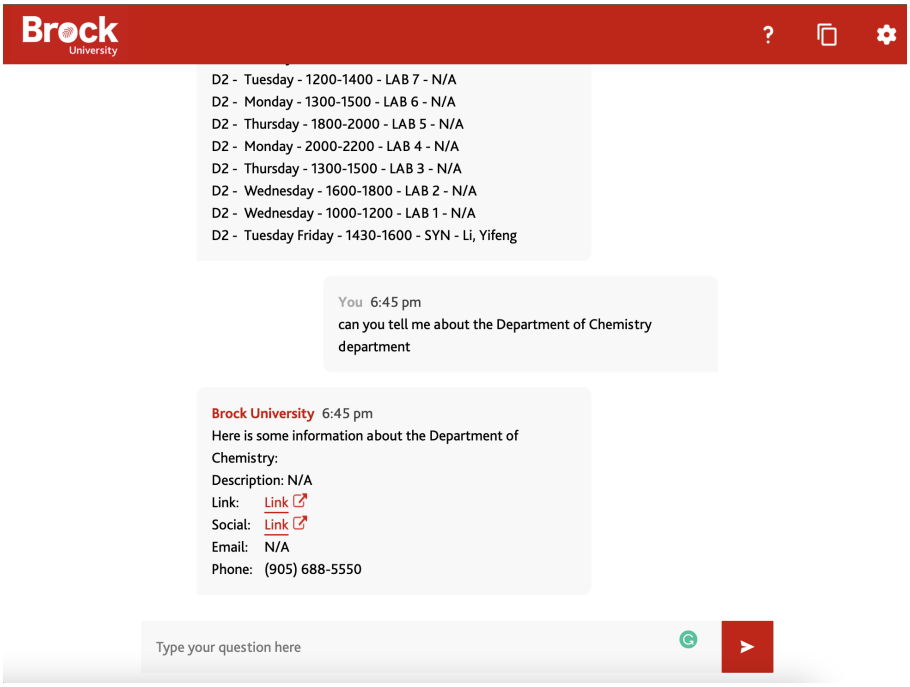
7. Saying hello to the Chatbot.



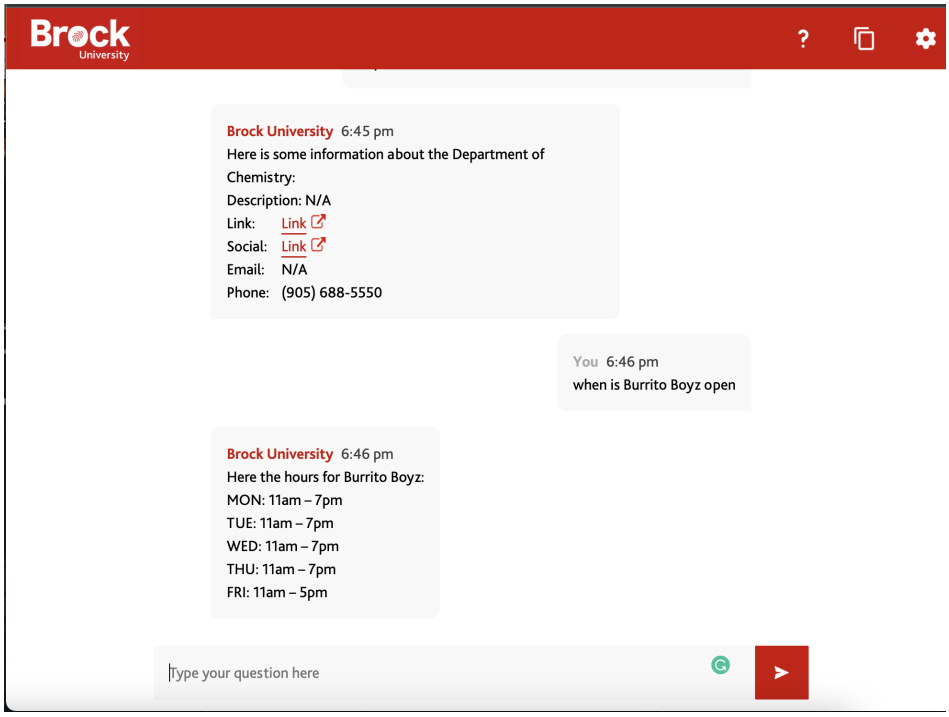
8. Asking what is the weather like at Brock?



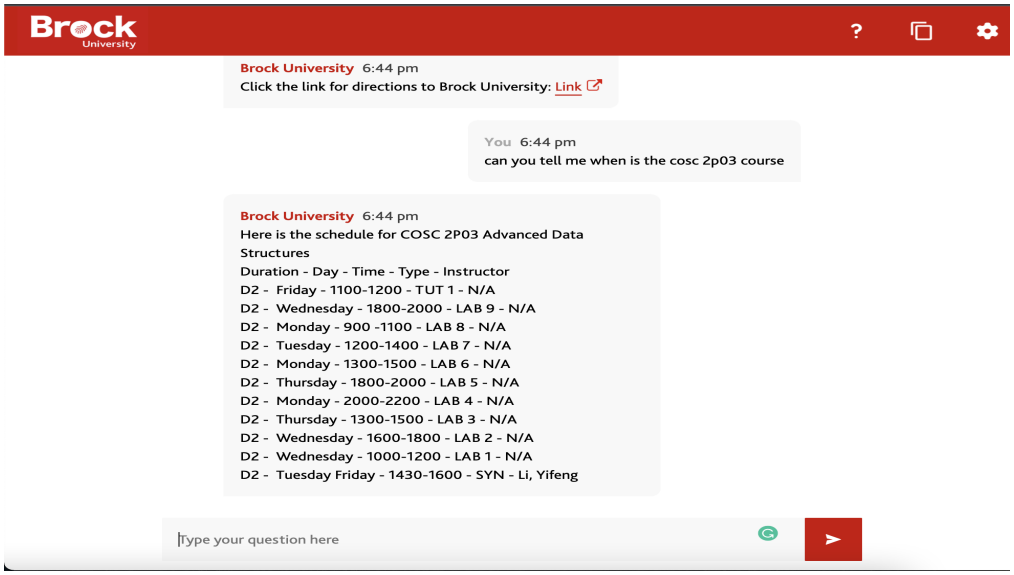
9. Asking about the Chemistry Department?



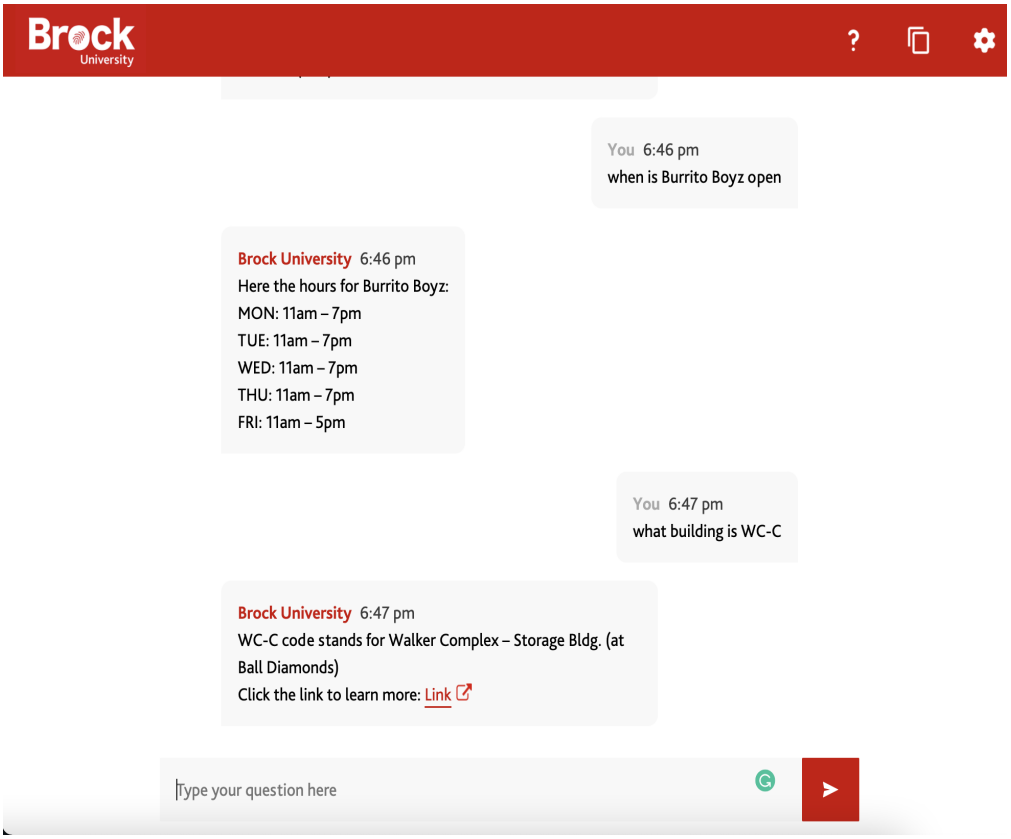
10. When is Burrito Boyz Open?



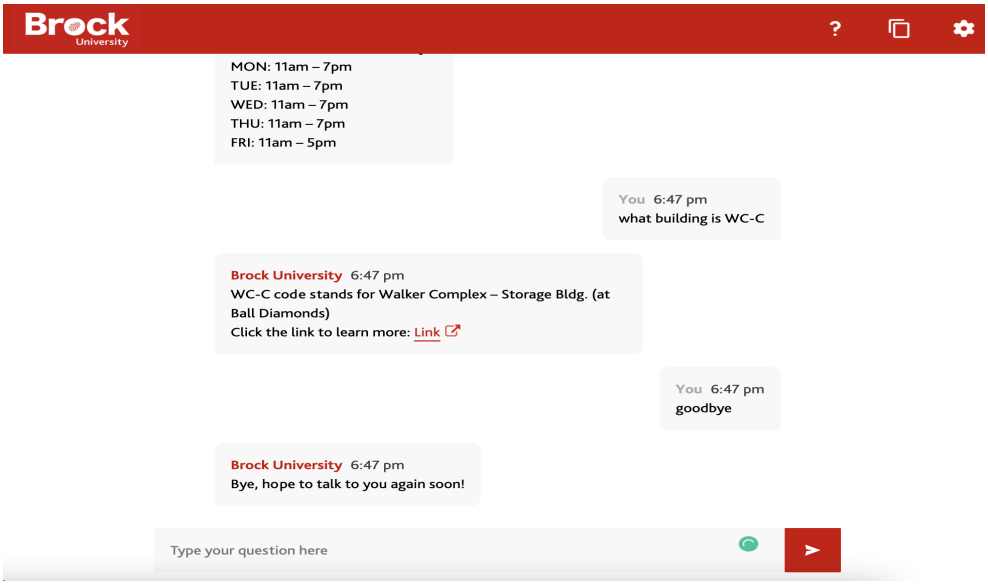
11. Tell me about COSC 2P03?



12. Where is the building based on the building code?



13. Saying Goodbye to the Chatbot?



5. Front End

The front end of our application was built using React. This library was chosen for its modularity and ease of use regarding the creation of front end user interfaces. React allowed us to break our code down into components, which allowed the separation of code based on relevance to the part of the UI it affected.

The front end was all built out of a single App component composed of many others. The App component served as a parent to our entire application which allowed us to pass callback functions around to various other child components as well as allowed us to easily combine other components together to create a cohesive user experience. These child components were made up of various sections of the application that we split apart in order to organize our code. The other components we had were as follows:

- Header
    - Represented the top header of our web application. This contained the Brock University logo as well as a number of selectable buttons
        - Get help
            - Brings back up the introductory modal for the user to reread
        - Copy chat log to clipboard
        - Settings
            - Change language
- Message Container
    - A container that held various instances of Message and displayed/arranged them in a way that was clear to the user.
- Message
    - A chat bubble that would appear when the user or bot would send a message. The formatting would depend on which user was sending the message.
- Input
    - The singular input field at the bottom of the screen that the user would interact with to ask questions to the chatbot.
- Modal
    - An introductory modal that appears upon initially entering the site. This can again be brought up by clicking the ? icon in the header
- ActiveButton
    - Simple indicator that appears after the user has typed a message to inform the user that the application is working and has not crashed.

The styling of the front end was heavily inspired by the Brock University official website as our goal was to ensure that our final product looked very close to being an official Brock application. While most of the styling was taken directly from the site, some additional assets and custom features had to be styled from scratch to try and match the rest of the site. Colour schemes and modern/minimalist design choices were made in order to ensure these features did not stick out in a way that detracts from the overall feel of the site.

Lastly, the front end also interacts with the backend through the usage of various POST requests containing the user input. The request data is very simple. It is a simple JSON format of two strings containing the user message and the language. eg. {“userMessage”: “hello”, “language”: “en”}

6. Scraping

Our application makes use of a variety of different scrapers to gather data to provide to the user. The scrapers were created to target specific web pages on the Brock University website and extract data that the chatbot could then use later. The backend API would interact with the scrapers in order to get the information that was to be formatted and sent back to the user. The various pieces of data that were scraped are outlined below.

- Courses (Database)
    - Program code
    - Course code
    - Description
    - Restrictions
    - Prerequisites
    - Corequisites
    - Grade replacements
- Course Offerings (Database)
    - Program code
    - Course code
    - Title
    - Duration
    - Lecture dates
    - Lecture times
    - Course type
    - Instructor
    - Section
    - Location
- Programs (Database)

    - Name
    - Description
    - Prerequisites
- Contact Information (Database)
    - Department
    - Name
    - Email
    - Title
    - Phone extension
    - Office
- Exams (Database)
    - Program code
    - Course code
    - Duration
    - Section
    - Exam date
    - Start time
    - End time
    - Location
- Departments (Database)
    - Name
    - Website
    - Phone extension
    - Email
    - Social Media
    - Description
- Clubs (Database)
    - Name
    - Email
    - Description
- Buildings (Database)
    - Name
    - Code
    - Link
- Restaurants (Database)
    - Name
    - Description
    - Hours of operation
- Transportation (Database)
    - Region name

    - Description
    - Routes/Map
- Important Dates (Database)
    - Occasion
    - Term
    - Stakeholder/Type
    - Date
- Brock News (Live)
- Brock Events (Live)
- Niagara News (Live)
- St. Catharines Weather (Live)

For each source of data we scraped, each scraper was used to get a good variety of relevant information that a user might want to ask for. Depending on the information being obtained, the data would either be stored to a database or provided to the user directly. These two categories of scrapers were appropriately referred to as database scrapers, and live scrapers.

Database Scraper

A database scraper is defined by whether or not the scraper stores the data into a database. This category of scraper is used to describe the scrapers that pull data from sources that update infrequently thus making the information safe to store and reuse repeatedly.

These scrapers implemented a fetch() method, which retrieved the data from the source, updated the relevant contents of the database, and then returned data that could be used. These scrapers also implemented a get() method which retrieved data from the database and returned it for use.

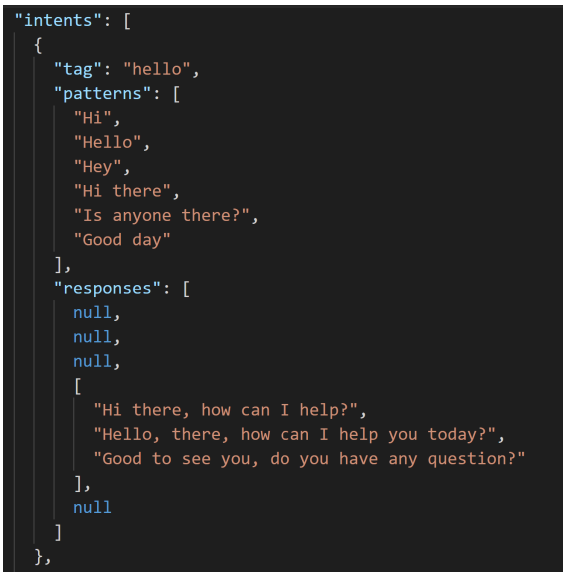
Live Scraper

A live scraper is any scraper that doesn't store the scraper information into a database and instead immediately provides the data to be processed and used. These scrapers usually gathered information that changed frequently rendering the storing of data and retrieving it again at a later date, nonsensical.

Unlike the database scrapers, this category of scrapers did not have a fetch() method and simply used a get() method to retrieve from the source and return data for use. This was to ensure consistency among all scrapers and prevent writing individual function calls for each scraper instance.

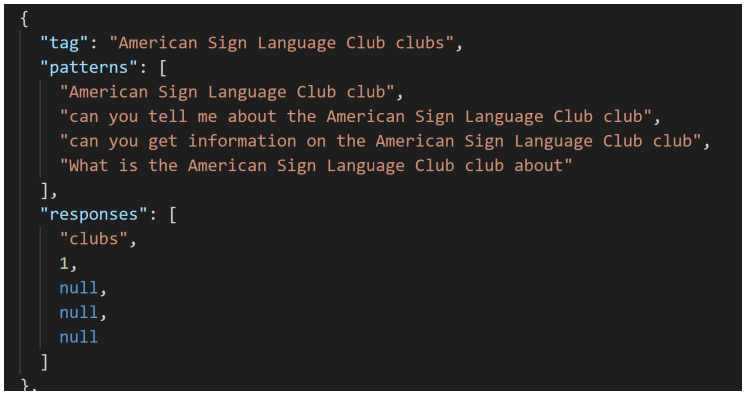
7. Generating Intents

Intents are a list of tags that hold a series of patterns and responses. The chatbot uses these intents to figure out what the user is asking for and what information is needed to respond to said user. Here are some example intents:



```
"intents": [
  {
    "tag": "hello",
    "patterns": [
      "Hi",
      "Hello",
      "Hey",
      "Hi there",
      "Is anyone there?",
      "Good day"
    ],
    "responses": [
      null,
      null,
      null,
      [
        "Hi there, how can I help?",
        "Hello, there, how can I help you today?",
        "Good to see you, do you have any question?"
      ],
      null
    ]
  },
```

If a user types in “hello” to the bot, the bot will respond with one of the predefined greetings, as shown.



```
{
  "tag": "American Sign Language Club clubs",
  "patterns": [
    "American Sign Language Club club",
    "can you tell me about the American Sign Language Club club",
    "can you get information on the American Sign Language Club club",
    "What is the American Sign Language Club club about"
  ],
  "responses": [
    "clubs",
    1,
    null,
    null,
    null
  ]
},
```

In this example the user would ask about the “American Sign Language Club”, The bot would then respond with the table name “clubs” and the index wherein the information for that club is located.

8. Natural Language Processing

In our Chatbot application we make use of natural language processing using such python libraries as NLTK, TensorFlow and Numpy. The system is trained on a file of intents, this file

contains tags that hold a series of patterns (questions that could be asked) and responses. The intents are generated by information pulled from the scraped SQL databases.

Here is how the system is trained by running the learn.py file:

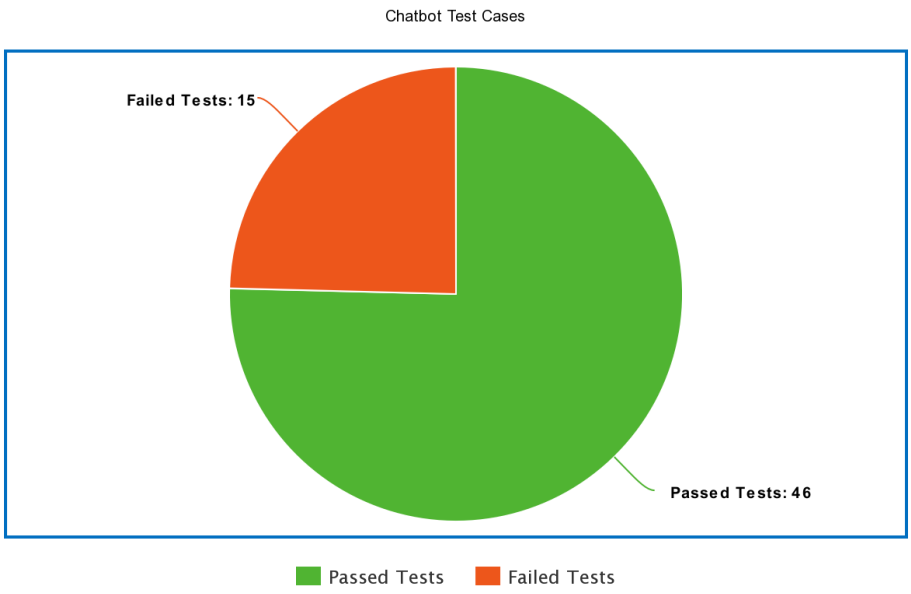
- First the intents are loaded into the system.
- Then the patterns are tokenized (the sentences are split into individual words).
    - Example: “what is the weather like today” is tokenized into to [“what”, “is”, “the”, “”weather”, “like”, “today”]
- The words are then stemmed (each word is trimmed down).
    - Example: “going” is stemmed into the word “go”
- The stemmed words are then matched to a Numpy ordered array that is used for training the bot.
- This information is then dumped into a compressed pickle file called “data”.
- Finally the bot is trained using TensorFlow. This outputs a few “model” files.

The bot runs by doing the following steps:

- First the bot checks for spelling errors in the question and attempts to correct them. The spell checker ignores certain words like program codes, building codes and contact names.
- Second the data and model is loaded into memory and TensorFlow and are initiated.
- Third the question is then tokenized and each word is then stemmed.
- The bot then matches the stemmed words with the stemmed words from the intents patterns. If they match, one is added to the bag and then the bag is ordered using Numpy.
- The bot then looks for the one with the highest probable match and checks if it is also over the preset confidence level. If so the bot matches the label with the tag in the intents and gets the corresponding response.

9. Testing

For our testing, we created several blackbox tests of our natural language processing. This was done by creating an automated test harness and writing tests containing user inputs we want to be able to anticipate. The list of tests was not a comprehensive list of all possible inputs on all types of data, but simply included a variety of inputs from the user that we wanted our application to be able to handle and compare the actual and expected outputs.



The diagram above shows the number of passed and failed tests that we had carried out. Of those 15 tests that failed, we noticed that 12 of them were tests involving grammatical errors. This is something valuable to note as it showed that most of the tests where grammar was correct, retrieved the correct data and helped us narrow down an area of our application where natural language processing was not working as well as we wanted.

We believe the case for 80% of our failed tests involving grammatically incorrect inputs can be attributed to the number of different ways an input can be given with poor grammar (incorrect spelling; use of periods, commas, capitalization, etc.) and that our chatbot, while capable of handling poor grammar in some form, was incapable of interpreting all inputs given in the test cases.

Fortunately, our testing seemed to be useful and we were able to tweak our settings to have our chatbot pass the majority of our test cases. Of the test cases we wrote, we were able to test for these pieces of data:

- Greetings
- Farewells
- Building codes
    - Name to code
    - Code to name
- Clubs
    - Emails
    - Descriptions
- Course

    - Description
    - Prerequisites
- Departments
    - Phone Extension
    - Email
    - Description
- Exams
    - Info
- Programs
    - Description
    - Prerequisites
- Restaurants
    - List
    - Description
    - Hours
- Transportation
    - Region
- Brock News
- Niagara News
- Niagara Events
- St. Catharines Weather

These tests do not go over every single piece of information that can be requested, however the tests written were created to ensure that the natural language processor was able to reliably respond to user input of that kind with the data relevant to the request.

Example Test Case

```python
class GeneralTests(unittest.TestCase):
    '''Tests regarding basic communication with the chatbot.'''

    def test_greeting(self):
        '''Saying a greeting to the chatbot.'''
        self.assertEqual(
            process_message('Hello', 'en'),
            {
                'table_name': None,
                'index': None,
                'associated_indexes': None,
                'messages': [
                    'Hi there, how can I help?',
                    'Hello, there, how can I help you today?',
                    'Good to see you, do you have any question?'
                ],
                'attributes': None
            }
        )
```