

COSC 4P02 Final Report

Brock University

and Canada Games

Chatbot

Greg Pogue, Madeline Janecek, Joel Jacob, Sam Langdon,
Brendan Park, Kylee Schram

30th April, 2022

Table of Contents

Table of Contents	1
1. Introduction	3
Application Scope	3
Tech Stack	3
Contributions	3
2. Project Management	6
Scrum Framework	6
Sprint Breakdown	6
3. Issues Encountered	9
Back-end-related Issues	9
Front end- related Issues	9
NLP-related Issues	10
Testing-related Issues	10
Deployment Issues	11
4. Manual	12
Web Scraping and Data Preprocessing	12
Frontend	14
NLP	16
Server	19
Deployment	19
Testing and Coverage	19
5. Future Works	21
Appendix A: Diagrams	22
System Structure Diagram	22
System Class Diagram	23
System State Diagram	24
System Sequence Diagram	24
Use Case Diagram	25
Entity-Relationship Diagrams	26
NLP Pipeline	27
CI/CD model	29
Appendix B: Project Management	30

	30
Product Backlog	30
Sprint Backlog	30
Appendix C: UI Design	31
Images	31
Components	32
Appendix D: Testing	34
Test Case Outlines	34

1. Introduction

GitHub: <https://github.com/janecekm/COSC4P02Project2022>

Hosted: <https://chatbot4p02.herokuapp.com/>

Application Scope

This project we developed is a web-based chatbot application that accommodates two distinct modes: one for Brock University and the other for Canada Games. We wanted to be able to provide a single website, where the user can ask questions that they have regarding that particular domain, i.e. Brock University or Canada Games, and be provided with meaningful answers. This may be a link to the resource or an actual answer so that the user doesn't need to hunt for that information.

Tech Stack

Technology	Usage
Python	Backend, Data cleaning
Selenium	Web scraping
React / Javascript	Frontend
Node Package Manager	Frontend
Flask	Server
SpaCy	Natural Language Processing
SQLite	Database
Flask-SQLAlchemy	Querying Database
PyTest	Testing
Coverage	Coverage
Docker	Containerization
Heroku	Deployment

Contributions

Greg Pogue

- Contributed to: Database, Back-end, Documentation
- Achievements:
 - Populating and preprocessing data for database
 - Solved database querying and routing keywords from NLP
 - Modularized back-end code to assist with Canada Games Expansion
 - Assisted with Docker containerization and deployment to Heroku
 - Organize and structure project management tools

Joel Jacob

- Contributed to: Hosting, Containerization, Testing, GitHub automation
- Achievements:
 - Developed the pipeline in Heroku to continuously deploy code.
 - Worked on Containerization of app using Docker to easily package and deploy web app
 - Used Pytest modules for Python, and developed unit test cases for the application.
 - Developed GitHub workflow to test on push and pull requests with main.

Madeline Janecek

- Contributed to: Front-end, NLP, Hosting
- Achievements:
 - Assisted with setting up Heroku to deploy code
 - Added to existing match patterns to accommodate a larger range of questions
 - Response generation for questions where the database is not involved or when information can not be retrieved from the database
 - Worked with React to develop UI

Sam Langdon

- Contributed to: NLP, Documentation
- Achievements:
 - Assisted with data requirements development
 - Creation, debugging and improvement of numerous match patterns for the NLP system
 - Integration of NER into the NLP pipeline
 - Documentation of NLP code (docstrings, additional commenting, NLP focussed README.md content)
 - General documentation creation/editing

Brendan Park

- Contributed to: NLP
- Achievements:
 - Integration of effective spell correction as preprocessing step to NLP pipeline

- Debugging to various backend issues leading to more accurate responses
- Assisted with building match patterns
- Integrated NLP into front-end and flask

Kylee Schram

- Contributed to: Database, Testing, Web-scraping
- Achievements:
 - Writing test case outlines for unit and integration tests
 - Ran and documented tests
 - Wrote some web-scraping scripts
 - Researching pytest, python assert, and .contains() for testing
 - Completing Offering table for the database including the SQLAlchemy model and preprocessing
 - Assisted with database querying and population

2. Project Management

Scrum Framework

We closely followed the scrum framework to track our product development. Our development process had requirements, design, develop, and testing phases. We developed user stories from the SRS and stored them as items in the product backlog, refining them over the project duration.

Sprints: 2 weeks long

Scrum Meeting Schedule: **Monday** and **Friday**

Sprint Reviews: Biweekly Monday

Sprint Retrospective and Planning: Monday after Sprint Review

Software: ClickUp and GitHub

Other Agile principles we followed:

- One-day spikes where we worked together to understand and organize a problem
- Pair programming helped us learn and solve problems together
- Epics - large user stories that required multiple sprints to complete

Sprint Breakdown

Sprint 1

Front end mockup

- Designed user interface in Adobe XD and laid out major components using the React JavaScript library so that they could be implemented in subsequent sprints

NLP research

- Different NLP python packages/libraries
- Different methods of spell correction in other NLP chatbots
- Creating lists for different key inquiries like course information, program information, exams/timetable and greetings to allow for matching of key terms

Primitive database

- Database relations and design
- Produced an entity-relationship diagram from the design
- Database created in PostgreSQL

Data gathering

- Initial web-scraping with Python Selenium
- Gathered course information, such as, prerequisites, cross listing, offering, lab times and so on, from different websites, and formatted it into a JSON file, so that we can use it for the database.
- Scraped the data about Academic advisors which is also formatted into JSON format.



Sprint 2

Building out features

Connecting subsystems together

Integrate database with Flask

- Implement as an embedded database in SQLite
- Connect database to Flask backend using SQLAlchemy

Data preprocessing

- Take scraped data and clean it such that the format is compatible with the database

Text-To-SQL/NLP Research

- Researched popular implementations and problems with Text-To-SQL
- Generated responses with a base template consisting of the extracted keywords
- Code-cleanup and improvements
- NLP integration to front-end via Flask

Added functionality to front end components, including:

- Query input area
- Message and response display with thinking animation and auto-scrolling
- Menu with font increase, font decrease, help, and mode switching options

Sprint 3

Preprocessing and populating database with course and offering info

Create a docker container for the application to assist in server deployment

NLP to Database querying:

- Research into SQLAlchemy querying methods
- Route keywords to query template

NLP:

- Evaluation of NER for detection of locations, people
- Formatting of extracted keywords for use by database/querying
- Creation of spelling correction system

Develop splash page with options for Brock University and Canada Games

Testing and deployment research

Sprint 4

Database querying using NLP keywords

- Filter keywords to match database formatting

Abstract SQLAlchemy data models from server class

Spell correction preprocessing layer added to NLP pipeline

NLP:

- Integration of spell checker into the NLP pipeline
- Addition of link responses for general questions and difficult questions where we are unable to get an answer from the database for the given query
- Debugging/improvement of keyword extraction patterns

- Implementation/integration of NER for detecting peoples' names
- Generation of text responses using data returned from the database for a selection of queries

Produced unit test cases for different functions within the application

Research testing automation

Deployment pipeline to Heroku from GitHub main branch

Sprint 5

Course table updates

- Adding additional information (title, restriction, and format)

Offering functionality

- Complete query options for Offering table
- Generate responses to Offering queries

Exam functionality

- Preprocessing exam data
- Add queries for Exam table
- Generate responses to Exam queries

NLP

- Expanding response generation capabilities
- Increasing keyword extraction scope
- Enhancing spell check

Automate testing

Automate deployment

Sprint 6

NLP

- Adding support for program questions, building information
- Added support for some Canada Games-based questions
- General bug fixes and improvements to resolve issues found during testing
- Modularized NLP matching and response generation code

Front-end

- Splash page improvements
- Made links clickable
- Copy-editing of various site texts

Database

- Creation of the Canada Games database
- Addition of building names, program info to Brock database
- General bug fixes and improvements to data quality

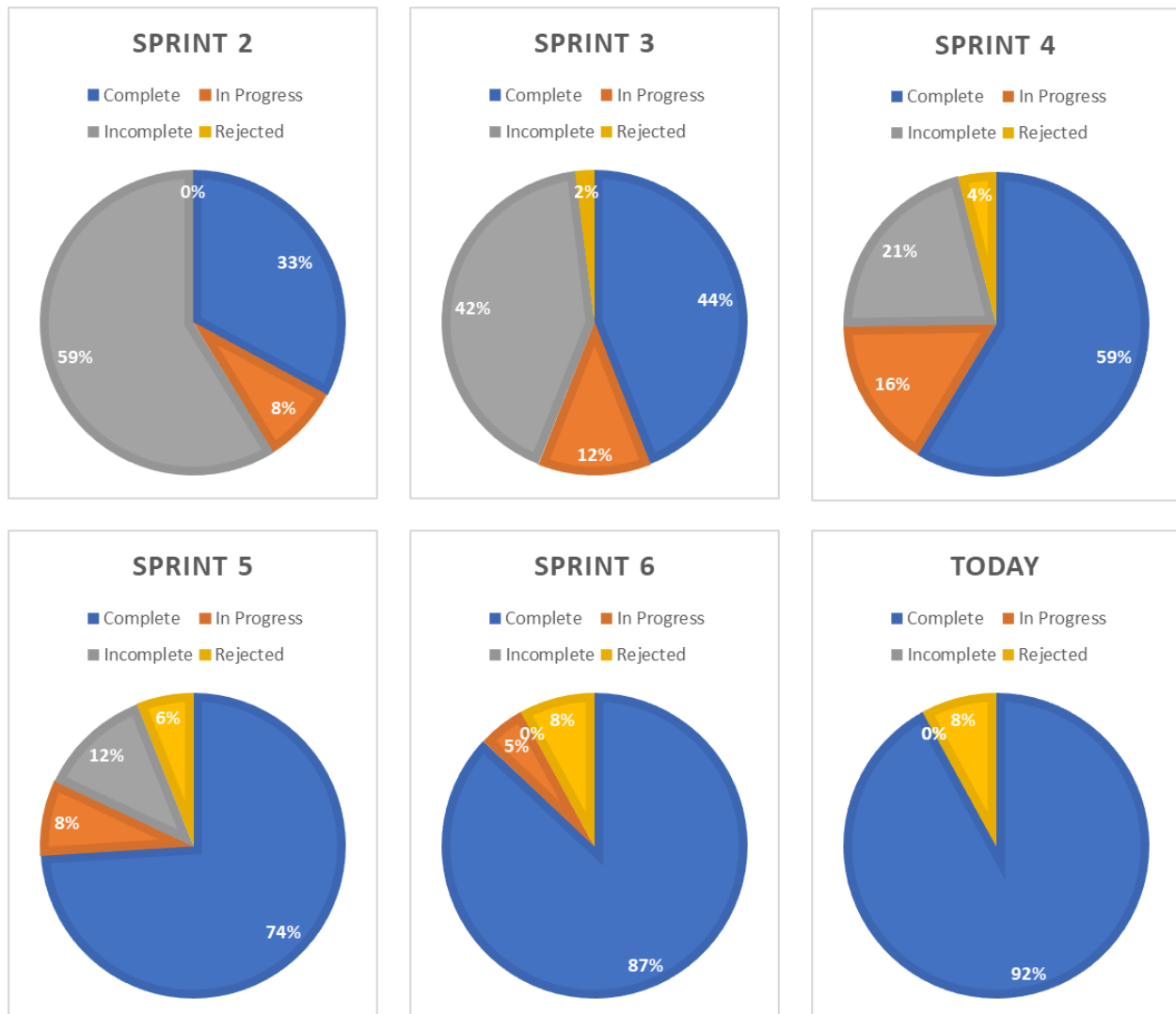
Testing

- Modularization of testing files
- Expansion of testing coverage

Performance Measurement

In the Project Proposal, we outlined the criteria for tracking progress. By Progress Report 1 we would have 40% user stories complete. By Progress Report 2 we would have 80% of user stories complete. These deadlines fell under Sprint 3 and Sprint 6 respectively.

The following charts show our user story completion progress over each sprint.



3. Issues Encountered

Back-end-related Issues

- Web scraping
 - Dynamic Scraping
 - One major issue with scraping the Brock websites was the dynamic nature of the pages, as some buttons needed to be clicked in order to access the information.
The tool we used really helped ease this process, as Selenium offers the functionality of dynamically loading pages.
 - Data format even within pages (ie, the timetable) wasn't consistent, which complicated pre-processing
 - Automated scraping and database population
 - Wasn't considered a priority, because we assumed the data didn't change frequently enough to justify running a cron to scrape daily.
- Brock University Data
 - Data was often inconsistent and complex, leading us to spend a lot of time preprocessing and cleaning it.
 - One source of this was the impact of COVID on lecture information, which saw lectures split into multiple formats like blended, synchronous, and asynchronous often missing locations.
- Back-end Infrastructure
 - Creating a functional back-end system required knowledge of each of the subsystems and how they communicate with one another.
 - Originally we were using the Node.js server packaged with React but this became difficult to integrate with our other components, mainly our NLP using Python and the database it would need to query. This created a new set of tasks to solve the networking.
 - After research into Flask, we pivoted to Python to consolidate our back-end architecture. Additionally, Flask allows us to use a SQL framework called SQLAlchemy to make simpler queries to an embedded database.
 - This also gave us the ability to serve the React app through Flask as opposed to routing our networking through a proxy.

Front end- related Issues

- React
 - React is a powerful tool for creating websites that have a clean and professional feel. That being said, its component-based nature does significantly increase the size and complexity of the project. While we have made a concerted effort to structure our project in the most intuitive way possible, keeping track of file

structures, component states, component interactions, and so forth can make it difficult for new people to read, modify, or update the frontend's code.

- Modularization
 - We wanted to modularize the front end as much as possible, so that in the future it can support various languages and more improvements. This presented us with the issue of making sure we make reusable components. As a result, a lot of time was spent considering how to separate the code into different files.

NLP-related Issues


- Match patterns
 - Due to the rule-based approach we chose for our NLP strategy, we were faced with the task of generating sets of keyword patterns that were likely to cover a broad spectrum of possible user queries. To try to tackle this issue, we tried to consider a wide variety of phrasings as well as asking for suggestions/questions from outside the team to get ideas for other patterns to support.
- Database integration
 - Getting the initial system for formatting keyword information so that it could be used to query the database took some time to set up. Additionally, there were some challenges in coordinating what information was held in the database to respond to queries to ensure that we were adding support for appropriate questions/question types. In solving this problem, we iterated over ideas and collaborated closely with the team members focussing on building the database to find a good solution for communicating between the NLP and the database.
- Spell correction
 - The frequency dictionary that shipped with the spelling correction package we used did not reflect the term frequencies that our chatbot was likely to experience in practical use. As a result, predicting the likelihood/frequency of various terms being used in the context of our chatbot and adjusting frequencies accordingly became somewhat of a challenge. To address this, we made adjustments to frequencies to correct issues found through testing the system with misspellings of words frequently used in chatbot queries (where, when, course codes etc)
 - The spell correction dictionary would not recognize certain terms unique to Brock or the Niagara Region and as a result would replace them with terms it was familiar with. To address this, we added these unique terms to the frequency dictionary to ensure that they would be supported by the spell checker.
- Modularization
 - Ensuring that the information that was unique to either Brock or Canada Games could be separated out and that when users switched between chatbots the matcher maintained only the appropriate match case information was more challenging to do than expected.

Testing-related Issues

- Pytest
 - The primary issue we faced while testing is figuring out what each of the functions does. The documentation provided by each of the functions helped, however, it was hard to keep up with the different changes that happened as the functions developed.
The next issue was that the file structure for our backend wasn't properly laid out, so there were some relative path issues and in order to fix that a lot of time was spent organizing the files. In the end, this helped us make a file structure that ensured that the testing files were easily found and the deployment of this was also easy to configure.
 - Decomposing some functions was difficult. We unit-tested many of the component functions, but larger functions were tested only for integration.
 - Would have liked to do scripted/automated load testing, but the logistics of high-volume testing against a server we didn't control weren't ideal.
- Github Actions
 - We initially had on-deployment automated tests, but these were failing on known non-critical test cases and preventing deployment, so they were eliminated.
 - The initial setup for GitHub actions brought some issues with the file path for our system. This had to be resolved as when testing, the file paths were messed up which caused false fails.

Deployment Issues

- Docker
 - The primary issues faced when configuring Docker for our software were that this was a new tool and understanding how to use this was difficult. A lot of time was spent performing spikes to understand how to use it, and configuring ports. In the end, this was accomplished with the help of the documents provided by Docker, and other sorts of resources.
- Heroku
 - While using this platform, configuring Docker and managing ports was tricky, as not only did we have to manage the port through which the application communicates with the end-user, but also had to manage the port through which the containers communicate with the platform.
In addition to that, we had an issue with the security of GitHub and Heroku, so we had to pivot into using git provided by Heroku. This is just a temporary issue, as when they are able to fix the security issues, we will be able to continuously deploy our code.
 - HTTPS was suggested as a user story but never implemented. After research, self-signed certificates were possible but not a route we wanted to take, and



working with real issued certificates would take more time to implement than we had at that point.

4. Manual

Web Scraping and Data Preprocessing

In this section, we describe how to update the database to the most accurate and updated version of the data.

The following .txt files must be present and up-to-date to build the database:

- buildingCodesClean.txt
- Program.txt
- Course.txt
- Exams.txt
- offering.txt

The following scripts are used to scrape the necessary data:

- offeringScrape.py
 - scrapes the offering table for Brock
- courseScrape.py
 - scrapes course info from the course calendar
- examsScrape.py
 - scrapes the exam info from the Brock timetable
- buildingCodesScrape.py
 - scrapes the building codes for Brock
- programsScrape.py
 - scrapes the program info from the Brock websites

The following scripts are used to ensure the data collected using the aforementioned scraping scripts is in the correct format:

- buildingCodecleanup.py
 - properly formats the building data we get.
- Cleaningcourse.py
 - fixes the issue where course scrape gets cosc4p61)
 - note: this is just to delete the) at the end.
- Cleaningtimetable.py
 - takes the timetable data scraped and normalizes it
- utfClean.py
 - converts a utf-16 encoding into utf-8 encoding

In order to update the data in the database, you will have to run the appropriate scraping and preprocessing scripts as listed below.

Web scraping	Data preprocessing	Text file	Data
offeringScrape.py	cleaningtimetable.py	offering.txt	The offering for a given year.
courseScrape.py	cleaningcourse.py	course.txt	The course information includes prereqs format and so on.
examsScrape.py	utfClean.py	exams.txt	The exams attached to courses
buildingCodesScrape.py	buildingCodecleanup.py	buildingCodeClean.txt	These are the data for building information. Notes: More match patterns are added right into the data that is cleaned, this is done so that keywords extracted can be identified while querying the database.
programsScrape.py		program.txt	These scrape the information for programs offered by brock. Notes: More match patterns are added right into the data that is cleaned, this is done so that keywords extracted can be identified while querying the database.

Initialize the database first using the command:

- Brock University
 - `sqlite3 buchotbot.db < bu_init.sql`
- Canada Games
 - `sqlite3 cgchatbot.db < cg_init.sql`

Now populate the database with the previously scraped data using the python input scripts:

- `python3 dbinput.py`
- `python3 cginput.py`

Frontend

Chatbox	tables are added and the splash page is structured proper	5 days ago
Language	made changes to testingfiles, added help menu,	3 days ago
Navbar	css seperated the cross corrected and added more menu options	5 days ago
popup	css seperated the cross corrected and added more menu options	5 days ago
brockcolor.css	some minor fixes	6 days ago
canadacolor.css	some minor fixes	6 days ago
index.css	tables are added and the splash page is structured proper	5 days ago
index.js	some minor fixes	6 days ago
logo.svg	Add files via upload	3 months ago

To help with organization, our components are split into different folders. The overarching file structure can be seen in the above picture. The main sub-folders are as follows:

1. Chatbox

ClipButton	css seperated the cross corrected and added more menu options	5 days ago
Feed	tables are added and the splash page is structured proper	5 days ago
Thinking	started splash screen	2 months ago
Zoomin	css seperated the cross corrected and added more menu options	5 days ago
Zoomout	css seperated the cross corrected and added more menu options	5 days ago
chatbox.js	tables are added and the splash page is structured proper	5 days ago
chatscreen.css	tables are added and the splash page is structured proper	5 days ago

The chatbox component is what deals with the core functions of our chatbot; accepting user input and displaying responses. The file `chatbox.js` is the primary component in this folder, and it utilizes several smaller supporting components, all of which are stored in their own folders.

2. Language

BrockInfo.js	made changes to testingfiles, added help menu,	3 days ago
CanadaInfo.js	added shadow to help	9 days ago
Lanprocess.js	cleaned the code a bit, added comments, and changed it so now sends t...	last month

This folder exists to support text modularization within the chatbot. Any static words used within the Brock chatbot or Canada Games chatbot (e.g. button names, descriptions, etc.) are put in a dictionary corresponding to the appropriate language. Then using the function offered by Lanprocess.js, we can retrieve text to display on the website with the keywords in the dictionary.

```
const values = {
  "en": {
    {
      "name": "BrockChatBOT",
      "clear": "Clear",
      "enter": "Enter",
      "message": "Hello! Welcome to the Brock chat bot! What can I help you with today?",
      "inputmessage": "Type a query here...",
      "disclaimer": "All information taken from Brock University",
      "helptext": "You can ask questions on topics like course scheduling, course prerequisites, exam schedules, transit o",
      "disclaimermenu": "Disclaimer",
      "disclaimertext": "This is scraped from brock websites",
      "aboutusmenu": "About",
      "aboutustext": "We are a group of people",
      "helpmenu": "Help",
      "mainhelpmenu": "Help Menu",
      "switchbutton": "Switch"
    }
  }
},
```

In the above picture we show the main dictionary for the english version of the website. Any additional static text should be added under the “en” identifier. Should the chatbot be extended to support any additional languages, a dictionary for that language would have to be created using an appropriate identifier (e.g. “fr” for french).


3. Navbar

Burgeritems	css seperated the cross corrected and added more menu options	5 days ago
backarrow	cleaned the code a bit, added comments, and changed it so now sends t...	last month
Navbardesign.css	css seperated the cross corrected and added more menu options	5 days ago
navbar.js	downloaded pop up library.	17 days ago

The Navbar component is what handles the secondary functions. It holds the back arrow component, which allows the user to return to the initial splash page. It also holds the burger menu, which gives the user the option to switch between the Brock and Canada Games chatbots, to read helpful tips on how to use the chatbots, and to see where we have retrieved our data.

4. Popup

popUp.css	added shadow to help	9 days ago
popUp.js	css seperated the cross corrected and added more menu options	5 days ago



This component is used to display a popup to the user, whether it be to provide information as to how to use the chatbot, or to give a disclaimer as to where the chatbot's data is coming from.

The commands to run are:

- npm install
 - This is to install the necessary package in the package.json file. This ensures that we can build the file into the backend folder.
- npm run build
 - This command runs and builds the files into the backend, however, we have to ensure we are in the chatbot folder.

NLP

SpaCy is a Python package for "industrial strength natural language processing". We use this package to handle all of the NLP tasks for our chatbot. We chose spaCy as it is specifically designed and optimized for use in production apps. You can read more about spaCy [here](#).

Our software primarily relies on pattern matching to extract keyword information from user queries. We do this by using spaCy's Matcher for token-based matching and PhraseMatcher for efficient case-insensitive matching for longer lists of phrases. For more information on how these matchers work refer to their documentation pages: [Matcher](#), [PhraseMatcher](#). For additional support, refer to the [rule-based matching guide](#).

For performing spell correction on user queries, we chose to use [SymSpellPy](#), as it provides easy access to the fast and robust SymSpell algorithm for spell checking. Documentation for SymSpellPy is available [here](#).

To handle the unique terminology required by the Brock chatbot (like course and building codes and names of locations unique to Brock or the Niagara Region) we load a custom dictionary for use with SymSpellPy. This dictionary, frequency_dictionary_en_82_765.txt, contains words and their corresponding frequency weights, and is based off of the dictionary that ships with SymSpellPy. To add the unique terms, we simply added entries to the dictionary's .txt file with weights chosen based on approximations of how frequently they may appear in the chatbot, relative to other terms. These frequencies can be adjusted based on usage and observation to improve accuracy.

The file structure for the NLP system is as follows:

- botNLP.py is the main point of entry into the NLP system, it acts as the link between the server and the database.

- brockMatcher.py and canadaMatcher.py define the match patterns and response generation logic for the Brock chatbot and the Canada Games chatbot. These files also include the link tables necessary to generate any link responses expected of the system.
- The nlp-resources folder contains any additional files needed by the NLP code. This includes
 - frequency_dictionary_en_82_765.txt, a text file containing the frequency dictionary used by SymSpellPy to perform spell correction. This file contains a line with each term and its corresponding frequency.
 - Files ending in "-list.txt" are used to define match patterns for the PhraseMatcher. These files contains lists of building codes, location and venue names etc

To add support for additional keyword patterns to the system, the steps are as follows:

1. Select which chatbot you wish to add the new patterns to (either Brock or Canada Games) and open the appropriate Matcher file (brockMatcher.py or canadaMatcher.py, respectively).
2. Determine if you wish to add a token-based match or a phrase-based match.
 - a. To **add a new token-based match pattern**, define the pattern by writing a list of dictionaries, where each dictionary defines what token attributes to be matched upon. An example of such a match pattern is seen below. This pattern is added to the matcher using "matcher.add("pattern_label", pattern)" with additional optional keyword arguments. In the example below, we use "greedy="LONGEST" to ensure that only the longest pattern that matches is stored (ex. only "what time is" and not both "what time" and "what time is").

```
# when, when is, when are, what time, what time is
when = [[{'LOWER': 'when'},
          {'LEMMA': 'be', "OP": "?"}],
         [{'LOWER': 'what'},
          {'LOWER': 'time'},
          {'LEMMA': 'be', "OP": "?"}]]
matcher.add("time", when, greedy="LONGEST")
```

An alternative option for creating token based match patterns is to use the [Matcher Explorer tool](#) provided by the maker's of spaCy. This tool helps guide you through the process of creating match patterns, as well as allowing you to see how they match on sample text of your choosing.

- b. To **extend an existing token-based match pattern**, add a new list of tokens to an existing match pattern list. For example, to add the pattern "instructor of" to

the "instructor" pattern you would add list of tokens you'd like to match to the list of patterns already defined, as shown in green below:

```
teaching = [[{'LOWER': 'who'},
              {'LEMMA': 'be', 'OP': '?'},
              {'LEMMA': 'teach'}],
            [{'LOWER': 'who'},
              {'OP': '*'},
              {'LOWER': 'instructor'}],
            [{'LOWER': 'who'},
              {'OP': '*'},
              {'LOWER': 'professor'}],
            [{'LOWER': 'who'},
              {'OP': '*'},
              {'LOWER': 'prof'}],
            [{'LOWER': 'instructor'},
              {'LOWER': 'of'}]]
matcher.add("instructor", teaching, on_match=assignPriority)
```

- c. To **add a phrase-based match to an existing category** of phrase-based matching patterns (for example a new building code), simply add the term to the appropriate .txt file.
- d. To **add a new category of phrase-based match**, use the following format, where “phrases” is the name of whatever type of pattern you wish to add and “pattern_label” is the label that you’d like to be assigned when the pattern is matched:

```
phrases = []
with open(filepath()+"phrases-list.txt", encoding="utf-8") as f:
    for line in f:
        phrases.append(line.strip())
patterns = list(nlp.pipe(phrases))
phrase_matcher.add("pattern_label", patterns)
```

To **update the spellcheck frequencies**, simply alter the frequency weights stored in frequency_dictionary_en_82_765.txt to be higher or lower depending on how likely you want the given term to be used in spell correction. For example, if “wjere” is being mistakenly corrected to “were” instead of “where”, you could address this by increasing the frequency value of “where” to be greater than that of “were”.

Server

For our server options, we used Flask, and using that we were able to easily host our website onto our local host for testing purposes. You can read more about Flask [here](#).

In order to run the development server, do the following:

- Change directory to backend
- Run `python3 devserver.py`
 - This is already configured to start running the application using the `server.py` file.

This will start running the server on localhost at port 5000.

So you can visit it at `localhost:5000`.

Deployment

For our hosting solution, we are using Heroku. The reason why we decided to use Heroku is because of the simplicity it offers with the power to be able to integrate with GitHub, and be able to engage in CI/CD model. The `heroku.yml` file is critical for this process, as it ensures that the container is fully built.

More documentation provided by Heroku can be found [here](#).


In order to be able to deploy our app, the commands that are used include:

- `heroku login`
 - required to log in to heroku by one of the owners of the pipeline
- `heroku git:remote -a name-of-app`
 - this is to add the git repo link into `.git` file so that we can track the file
- `git push heroku main`

This assumes that Heroku CLI is already installed and these commands are run from the command line.

Testing and Coverage

For testing our backend we used PyTest. Additional documentation can be found [here](#). PyTest allows easy integration with our existing Python code. It is also simple to automate as both a command-line script and GitHub action. Each file within our main folder that starts with `test_*.py`, contains the tests for one function. Multiple scenarios for that function are tested to ensure that we achieve at least 80% code coverage. We test the coverage percentage using the Python Coverage framework. More documentation can be found [here](#).



To add tests for future functions, files will be named `test_[function-name].py` and added to the backend folder containing the target function.

In order to run the tests, make sure you are in the backend folder and then run.

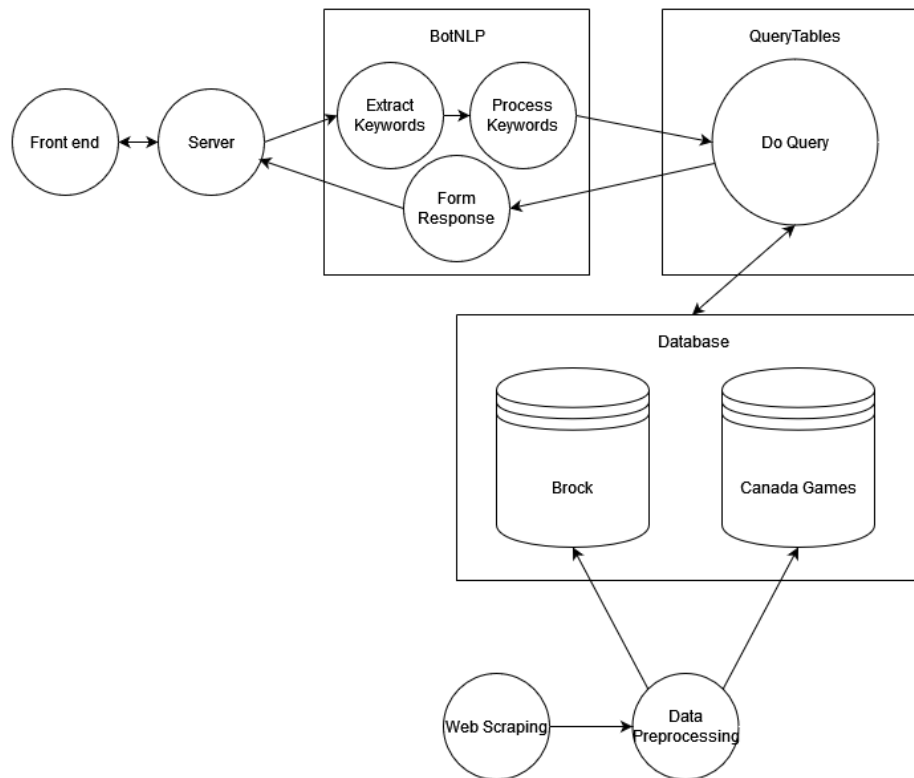
- `python3 -m pytest`
 - this will run all the files previously mentioned
- `python3 -m coverage run -m pytest .`
 - this will run and produce the coverage report for the testing files
- `python3 -m coverage report`
 - this command will produce the report on the command line

5. Future Works

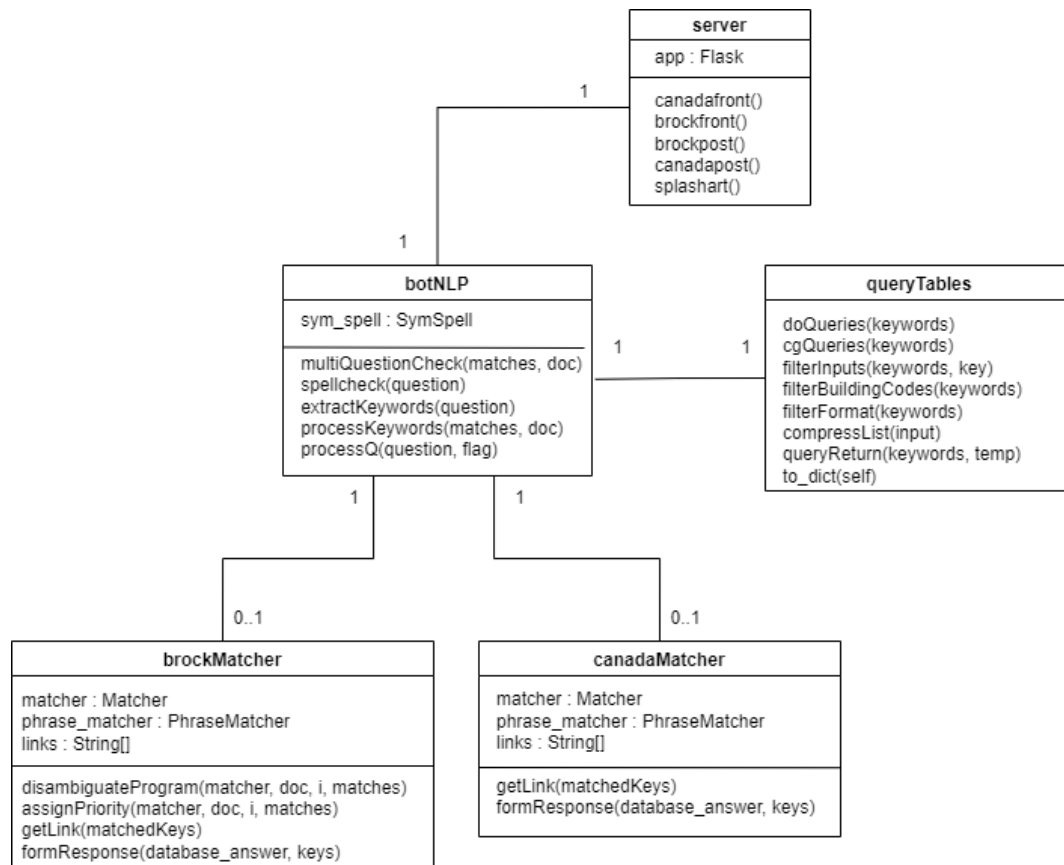
- Automate Web Scraping and database updating process.
 - We want to be able to run an automated script that runs every month or so, that scrapes the data from the appropriate websites and updates the database.
 - The proposed plan is using GitHub Actions, where we can set up a scheduler to be able to run the script at a particular time, and push the changes.
- Update front end UI.
 - We wanted to make the UI look better, and some suggestions were made, however, we did not have the time to be able to achieve this goal.
- Update UI for mobile devices.
 - Since we primarily focused on the design for the desktop version of the software, we didn't focus on the mobile version of the software heavily. However, the translation of the current UI to the mobile version of the UI is easy due to React.
- Update server to use HTTPS.
 - Flask offers the functionality of communication over HTTPS, however, we did not end up using this as Heroku and Docker's configuration files were not properly set up for this type of connection.
- Add Support for multiple questions in the same user input.
 - Our software in its current form, is able to detect that a user is asking multiple questions in a single message, however, we haven't configured it to be able to respond to multiple questions in a single reply yet.
 - Since we are able to identify multiple questions, providing a response to both questions should not be difficult to implement.
- Increase Canada Games chat bot functionality.
 - In the current version of our software, the Canada Games chatbot is able to answer only a selected number of questions. However, we want to support more questions, so that the end user can get more functionality out of the chatbot.

Appendix A: Diagrams

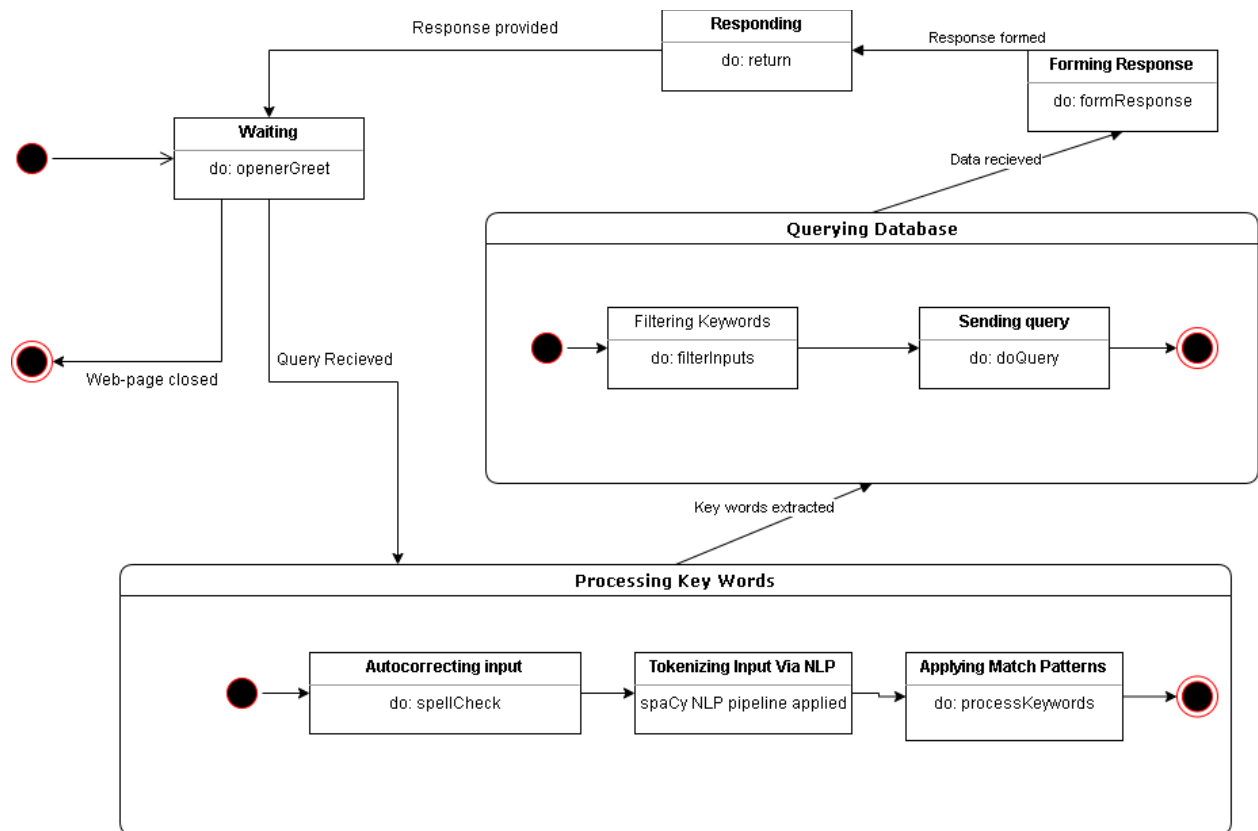
System Structure Diagram



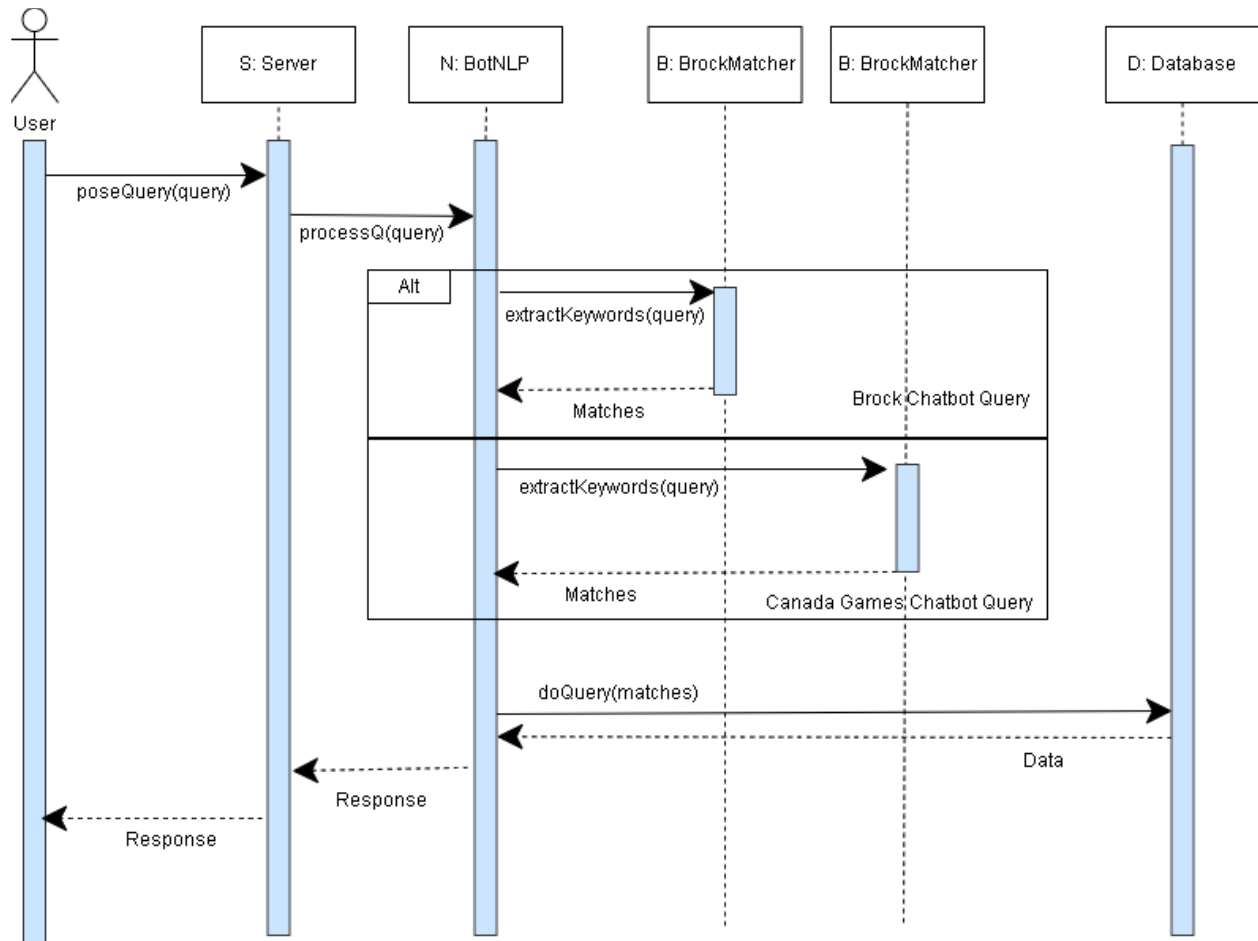
System Class Diagram



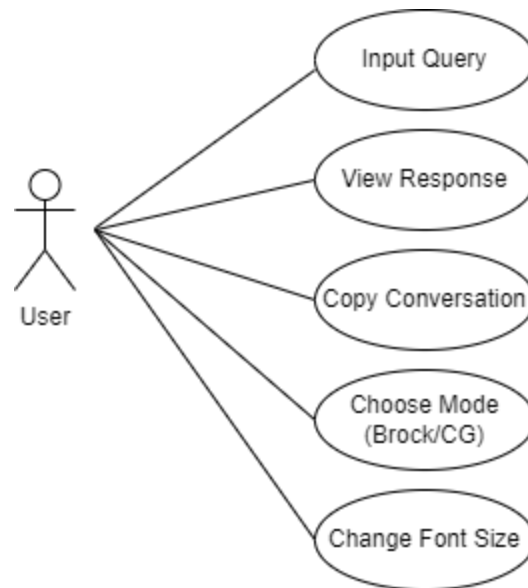
System State Diagram



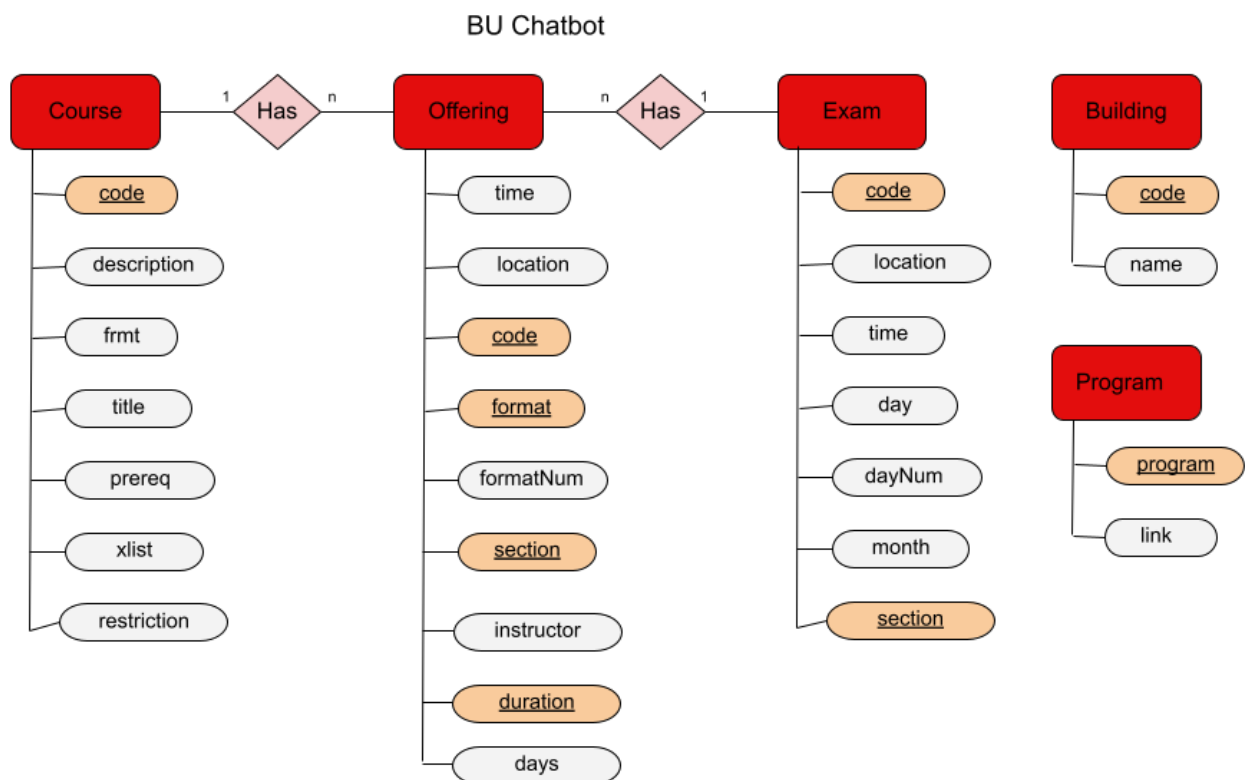
System Sequence Diagram



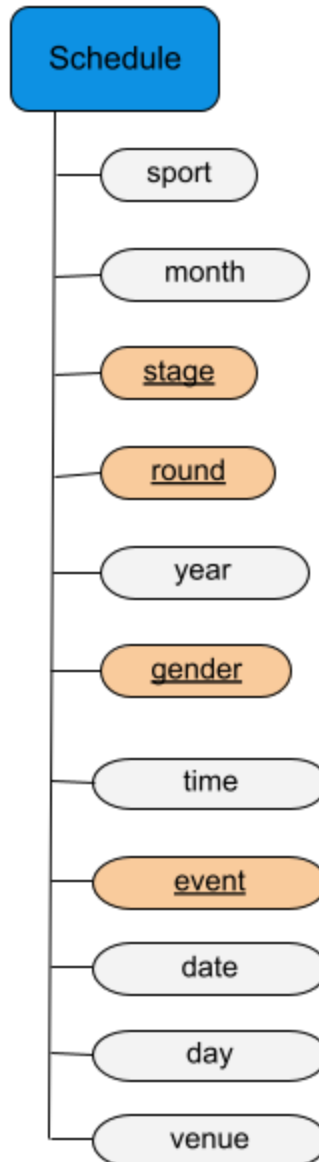
Use Case Diagram



Entity-Relationship Diagrams

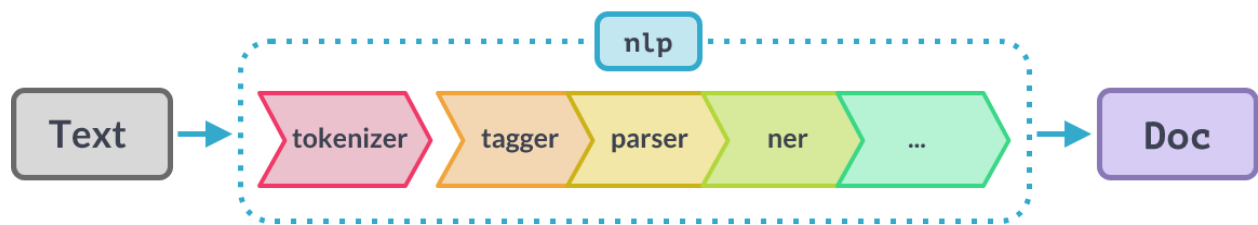


Canada Games Chatbot

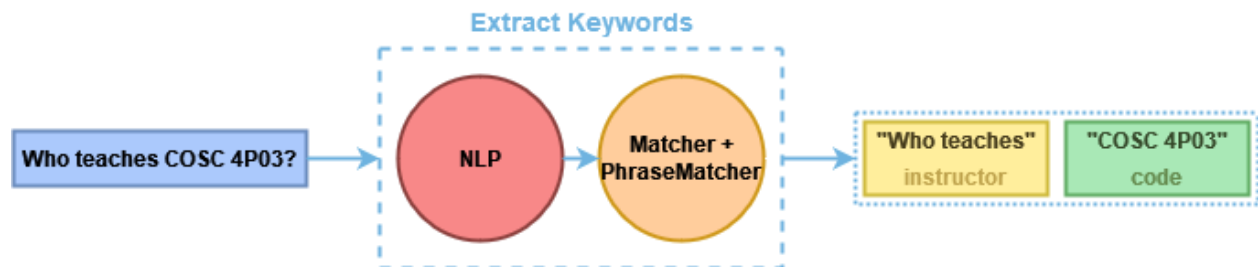


NLP Pipeline

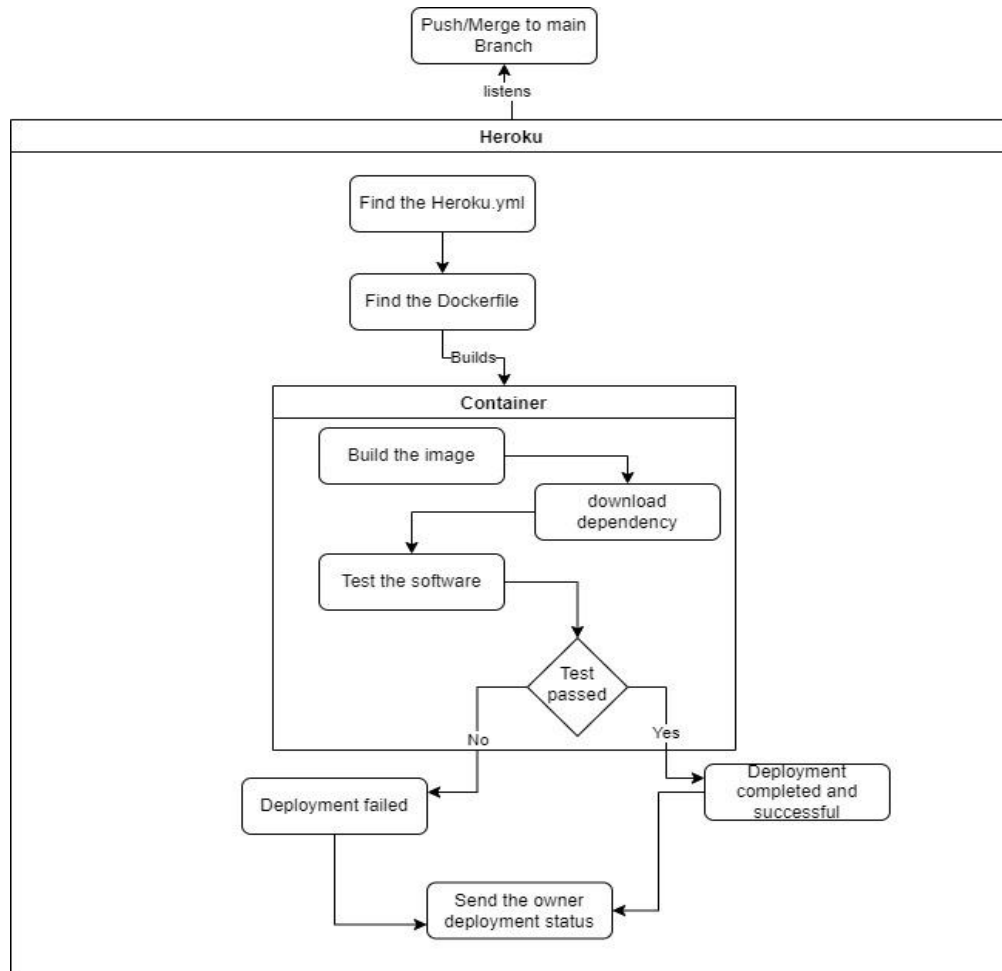
(ref: <https://spacy.io/usage/processing-pipelines>)



Processing a user query to extract keyword information for use by database/response generation



CI/CD model





Appendix B: Project Management

Product Backlog

See Product Backlog Final.pdf

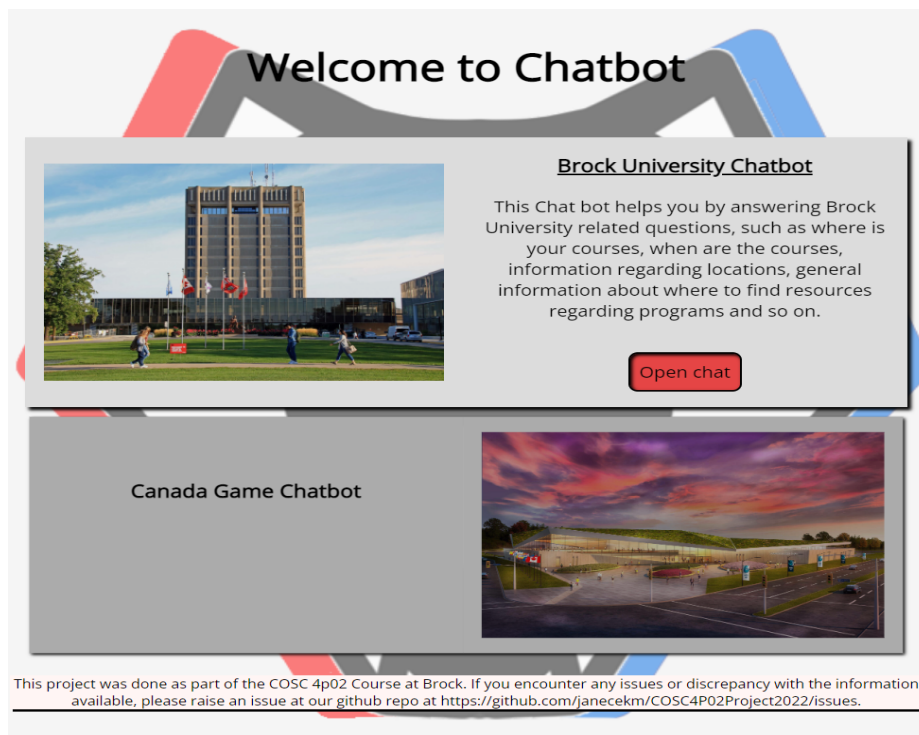
Sprint Backlog

See Sprint Backlog 1 - 6 pdfs

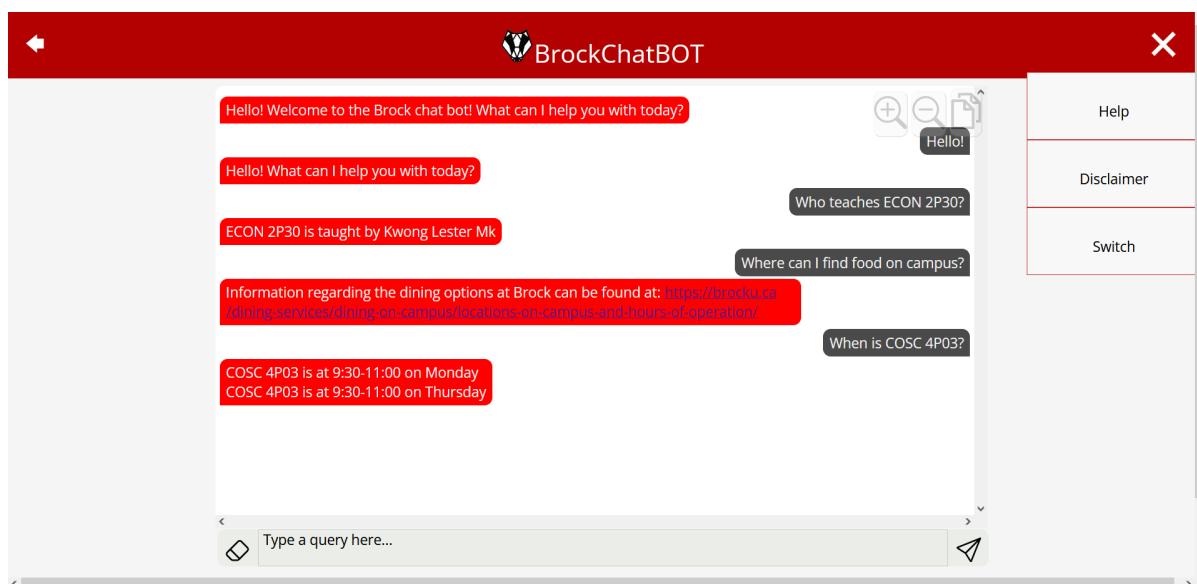
Appendix C: UI Design

Images

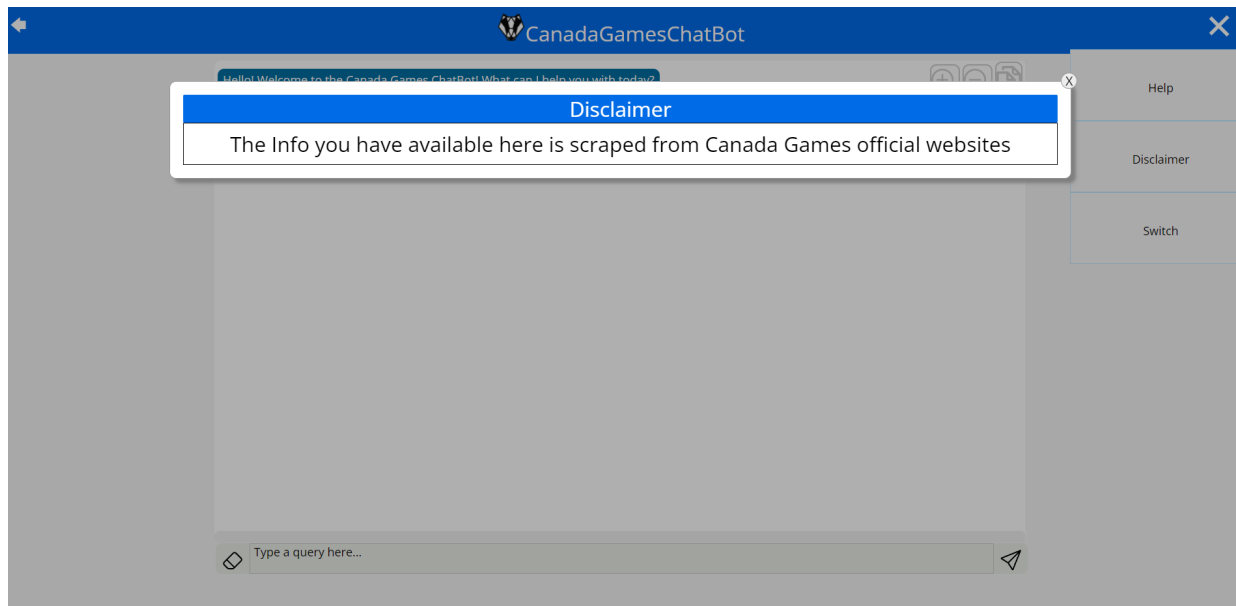
Splash Page:



BrockChatBot:




Disclaimer popup:



Components

The UI is made up of the following components:

- **Splash Page:** This initial page will allow the user to select the chatbot they wish to use (either Canada Games or Brock).
- **Back Button:** This button will return the user to the initial splash page.
- **Burger Menu:** Upon being clicked, a menu with different options will be displayed to the user. The menu can be closed by either clicking the button again or clicking outside of the menu.
 - **Help Button:** This button will display a popup informing the user of how to use the chatbot.
 - **Disclaimer Button:** This button will display a popup informing the user of where we've retrieved our data from.
 - **Switch Button:** This button allows the user to easily switch between the Brock chatbot and the Canada Games chatbot.
- **Font Increase/Decrease Buttons:** These buttons will allow the user to increase/decrease the font size of the text bubbles/user feed.
- **Copy Button:** This button will copy the conversation to the user's clipboard.
- **User Feed:** This is the area that shows the user queries and responses in the form of chat bubbles. These buttons will resize themselves depending on the length and font size of the texts. When the length of the conversation exceeds the user feed, a scroll bar will be shown and will automatically scroll with the addition of new queries/responses. When the chatbot is waiting for a response from the server, a thinking animation will be shown to the user.

- 
- **User Input Bar:** This is an area where the user can type their queries. When empty, a prompting message of 'Type a query here...' is displayed to the user. The input bar also imposes a character limit to ensure that any user queries are of a reasonable length. The input bar will resize itself to fit large input and font sizes.
 - **Clear Button:** Clicking this button will remove any text shown in the user input bar.
 - **Send Button:** Upon clicking this button, the input bar will be cleared and whatever the user put into the user input bar will be sent to the server. If the input bar is selected, pressing Enter will have the same effect. If there is no text in the input bar, no action will be taken.

Appendix D: Testing

Test Case Outlines

Testing Strategy & Cases

Testing Plan:

- Unit tests written for **botNLP.processKeywords()** and **botNLP.extractKeywords()**
- These functions used for integration testing on **botNLP.getLink()**, **botNLP.formResponse()**, **botNLP.processQ()** and **queryTables.doQueries()**. These six are the core chatbot functions.
- General_tests contains edge case tests, such as attempted SQL injection

Automation:

- Implemented via PyTest
- Tests for different functions modularized into separate files
- Python Coverage module used to get percentage coverage values automatically
- Used GitHub actions to run tests when code was merged with Main

Name	Coverage	Miss	Statements
BotNLP.py	82%	20	110
brockMatcher.py	94%	11	197
models.py	95%	3	64
queryTables.py	68%	67	210
	84.75%		

Testing files	Total	Pass	Fail
General Testing	11	11	0
queryTables.doQuery()	8	8	0
botNLP.extractKeywords()	25	25	0
botNLP.processKeywords()	13	13	0

botNLP.formResponse()	8	8	0
botNLP.processQ()	11	11	0
Miscellaneous tests	12	11	1
Total	88	87	1

Specific Function notes:

- botNLP.extractKeywords(): “failures” likely due to testing predicting match-spans incorrectly - match hashes are correct
- There are also other tests done to processes like autocorrection and other sorts of functions.

Function	Input	Expected Output	Output	Pass/Fail
queryTables.doQuery()	String : "What is COSC 1P03?"	Dict with {code : "COSC 1P03"} and {description : contains "Programming and problem solving in a high-level"}	Dict with {code : "COSC 1P03"} and {description : contains "Programming and problem solving in a high-level"}	PASS
queryTables.doQuery()	String : "What are prereqs for COSC 1P03?"	Dict with {code : "COSC 1p03"} and {prereq : "COSC 1P02"}	Dict with {code : "COSC 1p03"} and {prereq : "COSC 1P02"}	PASS
queryTables.doQuery()	String : "What is COSC 4P61 crosslisted as?"	Dict with {code : "COSC 4P61"} and {xlist : MATH 4P61}	{'xlist': 'MATH 4P61'}, 'code': 'COSC 4P61'}	PASS
queryTables.doQuery()	String : "When is COSC 1P02 exam?"	Dict with {code : "COSC 1p02"} and {time : "14:00-17:00"}	Dict with {code : "COSC 1p02"} and {time : "14:00-17:00"}	PASS
queryTables.doQuery()	String : "When is COSC exam"	String "more info required" or "placeholder return"	String : "more info required"	PASS
queryTables.doQuery()	String : "where is MCJ"	String containing "Mackenzie Chown"	String containing "Mackenzie Chown"	PASS
queryTables.doQuery()	String : "when is econ 2p30 lab"	Dict with {code : "ECON 2P03"} and {"course component" : lab} and	List of the labs for the course	PASS

		{time != none}		
queryTables.doQuery()	String : "who teaches math 1p66?"	Dict with {Instructor : != (NONE or " ")}	Dict with {Instructor : != (NONE or " ")}	PASS
botNLP.extractKeywords()	String : "What is COSC 1P03?"	response[0][0] == (15699362302781265145, 0,2) and response[0][1] == (3084953006211575075, 2, 4)	response[0][0] == (15699362302781265145, 0,2) and response[0][1] == (3084953006211575075, 2, 4)	PASS
botNLP.extractKeywords()	String : "COSC 1P03 what is"	response[0][1] == (15699362302781265145, 2,4) and response[0][0] == (3084953006211575075, 0, 2)	response[0][1] == (15699362302781265145, 2,4) and response[0][0] == (3084953006211575075, 0, 2)	PASS
botNLP.extractKeywords()	"What is prereqs for COSC 1P03?"	response[0][0] == (15699362302781265145, 0,2) and response[0][1] == (12246778916035409871, 0, 3) and response[0][2] == (3084953006211575075, 4, 6)	[(15699362302781265145, 0, 2), (12246778916035409871, 2, 3), (3084953006211575075, 4, 6)], what is prereqs for cosc 1p03)	PASS
botNLP.extractKeywords()	"Hello"	response[0][0] == (13357464451824626592, 0, 1)	response[0][0] == (13357464451824626592, 0, 1)	PASS
botNLP.extractKeywords()	"Who is teaching COSC 4P03?"	response[0][0] == (8794063152469658309, 0, 3) and response[0][1] == (3084953006211575075, 3, 5)	response[0][0] == (8794063152469658309, 0, 3) and response[0][1] == (3084953006211575075, 3, 5)	PASS
botNLP.extractKeywords()	"What time is MUSI 3P99?"	response[0][0] == (3084953006211575075, 3, 5) and response[0][1] == (8885804376230376864, 0, 3)	response[0][0] == (3084953006211575075, 3, 5) and response[0][1] == (8885804376230376864, 0, 3)	PASS

botNLP.extractKeywords()	"When is lab for COSC 1P02?"	(8383911667181403691, 2, 3) in response[0]	(8383911667181403691, 2, 3) in response[0]	PASS
botNLP.extractKeywords()	"When is tutorial for COSC 1P02?"	(8383911667181403691, 2, 3) in response[0]	(8383911667181403691, 2, 3) in response[0]	PASS
botNLP.extractKeywords()	"Where is CHEM 1P02 lecture?"	(4272944830542554761, 0, 1) in response[0]	(4272944830542554761, 0, 1) in response[0]	PASS
botNLP.extractKeywords()	"When is MATH 1P06 exam?"	(8885804376230376864, 0, 2) in response[0] and (9704506296879342145, 4, 5) in response[0]	(8885804376230376864, 0, 2) in response[0] and (9704506296879342145, 4, 5) in response[0]	PASS
botNLP.extractKeywords()	"Who is the advisor for math?"	(13790784326572561368, 3, 4) in response[0]	(13790784326572561368, 3, 4) in response[0]	PASS
botNLP.extractKeywords()	"Where is the bus?"	(18207339096255186229, 3, 4) in response[0]	(18207339096255186229, 3, 4) in response[0]	PASS
botNLP.extractKeywords()	"How do I get to Welland?"	(18207339096255186229, 0, 5) in response[0]	(18207339096255186229, 0, 5) in response[0]	PASS
botNLP.extractKeywords()	"How do I get to Narnia?"	(18207339096255186229, 0, 5) in response[0]	(18207339096255186229, 0, 5) in response[0]	PASS
botNLP.extractKeywords()	"Does the bus go to Narnia?"	(18207339096255186229, 2, 3) in response[0]	(18207339096255186229, 2, 3) in response[0]	PASS
botNLP.extractKeywords()	"How do I register for courses?"	(507029484556359861, 3, 4) in response[0]	(507029484556359861, 3, 4) in response[0]	PASS
botNLP.extractKeywords()	"Campus store?"	(7338335557000497525, 1, 2) in response[0]	(7338335557000497525, 1, 2) in response[0]	PASS
botNLP.extractKeywords()	"Where do I buy textbooks?"	(7338335557000497525, 4, 5) in response[0]	(7338335557000497525, 4, 5) in response[0]	PASS
botNLP.extractKeywords()	"Where can I eat?"	(18057327756930201825, 3, 4) in response[0]	(18057327756930201825, 3, 4) in response[0]	PASS
botNLP.extractKeywords()	"Where can I buy food?"	(18057327756930201825, 4, 5) in response[0]	(18057327756930201825, 4, 5) in response[0]	PASS
botNLP.extractKeywords()	"What are the covid rules at Brock?"	(2127825066894192516, 3, 4) in response[0]	(2127825066894192516, 3, 4) in response[0]	PASS

botNLP.extractKeywords()	"Do I need a covid vaccine?"	(2127825066894192516, 4,5) in response[0]	(2127825066894192516,4,5) in response[0]	PASS
botNLP.extractKeywords()	"How do I pay tuition?"	(1002886381125543945, 4,5) in response[0]	(1002886381125543945, 4,5) in response[0]	PASS
botNLP.extractKeywords()	"How much is tuition?"	(1002886381125543945, 3,4) in response[0]	(1002886381125543945, 3,4) in response[0]	PASS
botNLP.extractKeywords()	"What is crosslist for COSC 4P61?"	(12057252092477718455, 0,3) in response[0]	([(15699362302781265145, 0, 2), (12057252092477718455, 2, 3), (3084953006211575075, 4, 6)], what is crosslist for cosc 4p61)	PASS
botNLP.getLink()	Matches for "exam"	Response with "https://brocku.ca/guides-and-timetables/exams/#more-exam-info"	Response with "https://brocku.ca/guides-and-timetables/exams/#more-exam-info"	PASS
botNLP.getLink()	Matches for "what is prereq?"	Response with "https://brocku.ca/"	Response with "https://brocku.ca/"	PASS
botNLP.getLink()	Matches for "COSC 1P02"	Response with "https://brocku.ca/"	Response with "https://brocku.ca/"	PASS
botNLP.getLink()	Matches for "lab"	Response with "https://brocku.ca/"	Response with "https://brocku.ca/"	PASS
botNLP.getLink()	Matches for ""	Response with "https://brocku.ca/"	Responded with "https://brocku.ca"	PASS
botNLP.getLink()	Matches for "tauwehiwuaegawugu"	Response with "https://brocku.ca/"	Response with "https://brocku.ca/"	PASS
botNLP.getLink()	Matches for "advisor"	Response with "https://brocku.ca/academic-advising/find-your-advisor/"	Response with "https://brocku.ca/academic-advising/find-your-advisor/"	PASS
botNLP.getLink()	Matches for "tuition"	Response with https://brocku.ca/safa/tuition-and-fees/overview/"	Response with https://brocku.ca/safa/tuition-and-fees/overview/"	PASS
botNLP.getLink()	Matches for "cost"	Response with https://brocku.ca/safa/tuition-and-fees/overview/"	Response with https://brocku.ca/safa/tuition-and-fees/overview/"	PASS
botNLP.getLink()	Matches for	Response with	Response with	PASS

	"coronavirus"	"https://brocku.ca/coronavirus/"	"https://brocku.ca/coronavirus/"	
botNLP.getLink()	Matches for "covid19"	Response with "https://brocku.ca/coronavirus/"	Response with "https://brocku.ca/coronavirus/"	PASS
botNLP.getLink()	Matches for "quarantine"	Response with "https://brocku.ca/coronavirus/"	Response with "https://brocku.ca/coronavirus/"	PASS
botNLP.getLink()	Matches for "food"	Response with ""https://brocku.ca/dining-services/dining-on-campus/locations-on-campus-and-hours-of-operation/"	Response with "https://brocku.ca/dining-services/dining-on-campus/locations-on-campus-and-hours-of-operation/"	PASS
botNLP.getLink()	Matches for "eat"	Response with "https://brocku.ca/dining-services/dining-on-campus/locations-on-campus-and-hours-of-operation/"	Response with "https://brocku.ca/dining-services/dining-on-campus/locations-on-campus-and-hours-of-operation/"	PASS
botNLP.getLink()	Matches for "meal"	Response with "https://brocku.ca/dining-services/dining-on-campus/locations-on-campus-and-hours-of-operation/"	Response with "https://brocku.ca/dining-services/dining-on-campus/locations-on-campus-and-hours-of-operation/"	PASS
botNLP.getLink()	Matches for "register"	Response with "https://discover.brocku.ca/registration/"	Response with "https://discover.brocku.ca/registration/"	PASS
botNLP.getLink()	Matches for "registration"	Response with "https://discover.brocku.ca/registration/"	Response with "https://discover.brocku.ca/registration/"	PASS
botNLP.getLink()	Matches for "choose my courses"	Response with "https://discover.brocku.ca/registration/"	Response with "https://discover.brocku.ca/registration/"	PASS
botNLP.getLink()	Matches for "travel"	Response with "https://transitapp.com/region/niagara-region"	Response with "https://transitapp.com/region/niagara-region"	PASS

botNLP.getLink()	Matches for “transit”	Response with "https://transitapp.com/region/niagara-region"	Response with "https://transitapp.com/region/niagara-region"	PASS
botNLP.getLink()	Matches for “transportation”	Response with "https://transitapp.com/region/niagara-region"	Response with "https://transitapp.com/region/niagara-region"	PASS
botNLP.getLink()	Matches for “bus”	Response with "https://transitapp.com/region/niagara-region"	Response with "https://transitapp.com/region/niagara-region"	PASS
botNLP.getLink()	Matches for “contact”	Response with "https://brocku.ca/directory/"	Response with "https://brocku.ca/directory/"	PASS
botNLP.getLink()	Matches for “call”	Response with "https://brocku.ca/directory/"	Response with "https://brocku.ca/directory/"	PASS
botNLP.getLink()	Matches for “email”	Response with "https://brocku.ca/directory/"	Response with "https://brocku.ca/directory/"	PASS
botNLP.getLink()	Matches for “store”	Response with "https://campusstore.brocku.ca/"	Response with "https://campusstore.brocku.ca/"	PASS
botNLP.getLink()	Matches for “textbook”	Response with "https://campusstore.brocku.ca/"	Response with "https://campusstore.brocku.ca/"	PASS
botNLP.getLink()	Matches for “booklist”	Response with "https://campusstore.brocku.ca/"	Response with "https://campusstore.brocku.ca/"	PASS
botNLP.processKeywords()	Matches,doc for “what is cosc 1P03”	Dict with {"description" : what is} and {"code" : COSC 1P03}	Dict with {"description" : what is} and {"code" : COSC 1P03}	PASS
botNLP.processKeywords()	Matches,doc for "what are prereq for math1p67?"	Dict with {"prereq" : what are prereq} and {"code" : MATH 1P67}	Dict with {"prereq" : prereq} and {"code" : MATH 1P67}	PASS
botNLP.processKeywords()	Matches,doc for "where is my cosc 4p32 exam”	Dict with {"exam" : exam} and {"code" : COSC 4p32}	Dict with {"exam" : exam} and {"code" : COSC 4p32}	PASS

botNLP.processKeywords()	Matches,doc for "what is COSC 4p61 crosslisted as?"	Dict with {"xlist" : crosslisted} and {"code" : COSC 4P61}	Dict with {"xlist" : crosslisted} and {"code" : COSC 4P61}	PASS
botNLP.processKeywords()	Matches,doc for "what are the crosslisting of Cosc 4p61"	Dict with {"xlist" : crosslisted} and {"code" : COSC 4P61}	Dict with {"xlist" : crosslisting} and {"code" : COSC 4P61}	PASS
botNLP.processKeywords()	Matches,doc for "where is cosc 3p98 class"	Dict with {"location" : where} and {"code" : COSC 3p98}	Dict with {"location" : where} and {"code" : COSC 3p98}	PASS
botNLP.processKeywords()	Matches,doc for "who teaches math 1p66"	Dict with {"instructor" : who teaches} and {"code" : MATH 1P66}	Dict with {"instructor" : who teaches} and {"code" : MATH 1P66}	PASS
botNLP.processKeywords()	Matches,doc for "when is econ 2p30 class"	Dict with {"time" : when is} and {"code" : econ 2p30}	Dict with {"time" : when is} and {"code" : econ 2p30}	PASS
botNLP.processKeywords()	Matches,doc for "when is econ 2p30 lab"	Dict with {"time" : when is} and {"code" : econ 2p30} and {"course component" : lab}	Dict with {"time" : when is} and {"code" : econ 2p30} and {"course component" : lab}	PASS
botNLP.processKeywords()	Matches,doc for "where is MCJ"	Dict with {"location" : where} and {"buildingCode" : mcj}	Dict with {"location" : where} and {"buildingCode" : mcj}	PASS
botNLP.processKeywords()	Matches,doc for "How can I get to MCD"	Dict with {"buildingCode" : mcd}	Dict with {"buildingCode" : mcd}	PASS
botNLP.processKeywords()	Matches,doc for "Where is MCD 205"	Dict with {"buildingCode" : MCD}	Dict with {"buildingCode" : mcd}	PASS
botNLP.processKeywords()	Matches,doc for "Where is MCD205")	Dict with {"buildingCode" : MCD}	Dict with {"buildingCode" : mcd}	PASS
botNLP.formResponse()	String "What is VISA 1p95"	"Technical foundations of digital images, media methods and concepts" in response string	"Technical foundations of digital images, media methods and concepts" in response string	PASS
botNLP.formResponse()	String "when is econ 2p30 exam"	"April 18 at 14:00-17:00 in WCDVIS" in response string	"April 18 at 14:00-17:00 WCDVIS" in response string	PASS

botNLP.formResponse()	String "where is econ 2p30"	"Sync" in response or "STH 207" in response	"Sync" in response or "STH 207" in response	PASS
botNLP.formResponse()	"where is clas 1p97"	"EA 106" in response	"CLAS 1P97 is in room EA 106"	PASS
botNLP.formResponse()	String "what are the prereqs for entr 2p51"	"No prerequisites" in response	"No prerequisites" in response	PASS
botNLP.formResponse()	"what are the prereqs for btec 4p06"	"BTEC 3P50, BIOL 3P51" in response	"BTEC 3P50, BIOL 3P51" in response	PASS
botNLP.formResponse()	"Who teaches COSC 4P61"	"Ke" in response	"Ke" in response	PASS
botNLP.formResponse()	"Who teaches VISA 1P95"	"Cerquera Benjumea Gustavo" in response	"Cerquera Benjumea Gustavo" in response	PASS
botNLP.formResponse()	"Who teaches PHIL 2p17"	"Lightbody Brian" in response	"Lightbody Brian" in response	PASS
botNLP.formResponse()	"Who teaches BIOL 4p06?"	"Liang Ping" in response	"Liang Ping" in response	PASS
botNLP.formResponse()	"what crosslist COSC 4p61"	"MATH 4P61" in response	"MATH 4P61" in response	PASS
botNLP.formResponse()	"What crosslist BIOL 4P06"	"BTEC 4P06" in response	"BTEC 4P06" in response	PASS
botNLP.formResponse()	"This is nonsense"	" https://brocku.ca/ " in response	" https://brocku.ca/ " in response	PASS
botNLP.processQ()	"raietweiaweeiiwnaeiug we"	Response['message'] is "I'm sorry, I wasn't able to find what you were looking for. However, you might be able to find more information at: https://brocku.ca/ " or "I am not quite sure what you're asking. Could you rephrase that?"	Response['message'] is "I'm sorry, I wasn't able to find what you were looking for. However, you might be able to find more information at: https://brocku.ca/ " or "I am not quite sure what you're asking. Could you rephrase that?"	PASS
botNLP.processQ()	"What is the prereqs for COSC 1P03"	"COSC 1P02" in response['message']	"COSC 1P02" in response['message']	PASS
botNLP.processQ()	"What aer the Prereq for	"COSC 1P02" in	"COSC 1P02" in	PASS

	COSC 1p03'	response['message']	response['message']	
botNLP.processQ()	"Qjtat is jet prereq for COSC 1p03"	"COSC 1P02" in response['message']	"COSC 1P02" in response['message']	PASS
botNLP.processQ()	'Wat is prereqs for COSC 1P03'	"COSC 1P02" in response['message']	"COSC 1P02" in response['message']	PASS
botNLP.processQ()	'What are the prereqs for *- 1P02'	Response['message'] is "I'm sorry, I wasn't able to find what you were looking for. However, you might be able to find more information at: https://brocku.ca/ " or "I am not quite sure what you're asking. Could you rephrase that?"	Response['message'] is "I'm sorry, I wasn't able to find what you were looking for. However, you might be able to find more information at: https://brocku.ca/ " or "I am not quite sure what you're asking. Could you rephrase that?"	PASS
botNLP.processQ()	'What are the prereqs for /**/ *-'	Response['message'] is "I'm sorry, I wasn't able to find what you were looking for. However, you might be able to find more information at: https://brocku.ca/ " or "I am not quite sure what you're asking. Could you rephrase that?"	Response['message'] is "I'm sorry, I wasn't able to find what you were looking for. However, you might be able to find more information at: https://brocku.ca/ " or "I am not quite sure what you're asking. Could you rephrase that?"	PASS
botNLP.processQ()	'Hello Hello'	"Hello! What can I help you with today?" in response['message']	"Hello! What can I help you with today?" in response['message']	PASS
botNLP.processQ()	"what are prereq for math1p67"	"MATH 1P66" in response['message']	"MATH 1P66" in response['message']	PASS
BotNLP.processQ()	"hello"	"Hello in response"	"Hello! What can I help you with today?"	PASS
botNLP.processQ()	"Where is my exam"	"Exam" in response	Link to the exam timetable	PASS
botNLP.processQ()	"Where is math 1p66"	"STH 204" in response	The lecture times and component times returned	PASS
brockMatcher.formResponse()	"Where is econ 2p30 lec? "	"SYNC" in response	I don't know what you asking about	FAIL

—