13/12/2021

Assignment 2: Creating a chatbot

COS60016: Programming for development

Matthew Gennaccaro
103771529 SWINBURNE ONLINE

Matthew Gennaccaro 103771529

Contents

Approach taken in building integrated chatbot

When deciding what approach to take to create and build the integrated chatbot, there were considerations to make prior. The first, and perhaps most important of all, is the time constraint of the project. I had to make a chatbot, but the skills required to do so were not covered until the final week of the unit, which ends 3 days before the assignment is due, the same is true for the other unit I am completing alongside this one. As someone with no programming background prior to this course, I was very reliant on the skills taught in the modules to achieve learning outcomes and complete the assignment. Additionally, it meant that there are weaknesses that are clear within the programming of the chatbot that I would otherwise rectify prior to submission or release, but simply did not have enough time to do so. I also had to consider my current level of programming knowledge and be realistic about the approaches I could take, as stepping too far outside course content would require time to learn and practice.

There are many ways that chatbots can be made and while I did have issues with getting much of the course material to work as intended, it was still my most-used source of information. I also referred to my previous assignment as a refresher in using Django. With these considerations in mind, I begun my assignment by first creating the necessary templates and framework. I personally work best when I create some kind of outline first, and then complete the work within it.

In the context of this assignment, I first created a basic HTML page and integrated it into the web application I'd previously created, creating a space to house the chatbot. Once that was complete, I decided to work on setting up the necessary framework for the second application in the project using Django. Being more comfortable with the Django framework, this was a faster process overall this time around, however, I did encounter some issues as I hadn't created a second app within Django before. The biggest issue I encountered in setting up the second app was specifying the location of the JSON file to read the question and answers from. In the modules, the "BASE_DIR" from the Django settings was referenced, however, that same approach would not work for me, so I ended up referring to the file locally. During development I was using an absolute path to ensure the correct file was being used, and then later figured out how to reference it locally.

Once I begun the task of creating the view for the chatbot, I decided to reuse a lot of the knowledge and skills I gained from the first assignment. While we did cover some advanced techniques in python, I simply hadn't had enough time or practice with all of them to use them efficiently. As for the view itself, I once again used the requests module to first determine if the request method was POST, if so, proceeded with creating a form and if valid, the form would be used to capture the "question" of the user. I decided to use the ListTrainer to use a JSON file I created to train the chatbot. This method made the most sense to me to train the chatbot quickly and easily for some basic questions and answers. I originally had the API call in each nested "if "statement but noticed immediately that this resulted in far too much redundant code, so I moved the API call out to be executed prior to the keyword checks. If the user's question was not within the JSON file but did contain a keyword such as "temperature" or "humidity", then the appropriate "if" statement would be triggered, the relevant JSON data would be obtained from the API and then passed into a response from the chatbot. This approach works for the required task; however, through my testing I found a major issue with the chatbot, to be discussed in the following section.

Process of testing chatbot

Once again, testing was performed throughout the development of the chatbot. More testing was required with the API call, as there is more conditional programming involved. I did not write any test programs using "pytest", as I was very time-conscious when completing the assignment. Most of my testing occurred at the user level and lead to the discovery of a major issue with the chatbot. If the user inputs a keyword on its own, such as just "temperature", the keyword then gets passed to the API call as the city name, and it breaks the chatbot. I am of the opinion that no user input should ever be able to break a program and I definitely would try to rectify this if I had the time. To do so, there could be some form of input validation similar to how I handled user input in the weather app. In this case it may be more difficult as the input can be much more complex. Perhaps a separate input box could be used to allow the user to submit a question and the location separately. The city name variable could then take that separate input to pass into the API call, thus preventing the incorrect word being passed as the city name. This is a very rough workaround to the issue, however, and I believe it may defeat the purpose of a chatbot's purpose to flow like natural conversation as much as possible.

When creating and training a chatbot, the terminal can be used to test the chatbot functionality before implementing it into an app. This can be achieved by creating an infinite while loop and running the program to "talk" to the chatbot in the terminal. This is helpful to see what kind of responses you will get from the bot immediately. While this is helpful before the view is created and functional, I found this less useful after I had my chatbot integrated into my web app, as I could do the same testing at the user level. Aside from chatbot functionality, I used a bottom-up testing approach to ensure URL configurations were working correctly, especially since I was integrating a second app, as well as templates ensuring the overall framework was functional. After this I conducted the views testing and then high-level functionality testing of the entire project.

I've learned the importance of testing when developing any program, as even slight mistakes in code can have drastic effects on said program. I've also learned how taking extra time to test early, can save time when coding and testing later. Ensuring the basic functionality of your program and framework are working correctly first, makes the process of testing higher level functionality later on much smoother.

In the future, I will definitely put more time and effort into testing basic configurations and functionality at very early stages, as I have experienced firsthand how much time and frustration this can save in the future. If I had more time, I would have ensured that no user input could potentially break the chatbot, as I believe that is not indicative of a high quality and well-designed program. Additionally, more time would have allowed for writing tailored testing programs. While I did employ a test-driven approach, I believe there is room for improvement specifically in writing tests and employing testing techniques earlier in development.

Link to GitHub repository

https://github.com/Mattscodingacc/COS60016_GENNACCARO_Matthew_Assignment_2_Creating_a_chatbot.git