Mahad Abdi, Jerry Hou
CSCI3359 Fall 2022
Professor Lewis Tseng
12/16/2022

Laundry Bot: A Text Bot That Helps With Laundry

We have decided to develop a text bot that can assist users with keeping track of laundry availability in their residential buildings. By asking the user to input their dorm information, the bot can provide updated laundry status in close to real-time. In addition, the bot can serve as a reminder, sending a notification to the user when a laundry machine becomes available or when their laundry cycle is complete. We chose this project because we experienced times when we wanted to do laundry only to find out that all the machines were in use or that people had forgotten to take out their clothes when they finished. This project aims to make the laundry process more convenient and efficient for students living in residential buildings.

The project has two responsibilities: communication with clients and access to laundry machine information of different residential buildings. To effectively inform users about the availability of laundry machines and send reminders, the text bot must be able to send and receive SMS messages and have access to current laundry machine availability. By fulfilling these responsibilities, the text bot is able to make the laundry process more convenient and efficient for the users.

We chose Twilio as the means to communicate with the clients over other services for a variety of reasons. Twilio provides an easy-to-use API that allows us to easily add message-sending and receiving functionality to the application. The API has all of the necessary functions that our application requires. For example, our application sends a reminder in the future, which is easily done with Twilio's API because it offers message scheduling functionality. So in order to send a reminder in the future using Twilio all you have to do is schedule a message. This functionality isn't offered in similar APIs, so if we had used another API we probably would have had to come up with our own solution. We chose Java as our framework for consistency since the process of how we extract the laundry information of the residential halls, which will be explained later in the report, is done using Java, and the Twilio API supports using Java to build the customized sending and retrieving information from the client. By using a single language, it will be easier to integrate these different components into our project. Overall, Twilio's easy-to-use API that offers all of our requirements and compatibility with Java made it a strong choice for our project's communication needs.

To understand how our text bot is able to access laundry machine information, it's important to understand the data collection process. Initially, we planned to use web scraping on the website laundryview.com, which provides laundry machine information, to parse the relevant HTML code containing dorm information and store it. For each residential hall, we would have stored four fields of data: the time of data collection, the residential hall name, the number of available washers, and the shortest time remaining for a washer to finish if there are none currently available. However, upon examining the HTML code, we discovered that

laundryview.com did not provide access to this information. This surprising finding forced us to change our approach to using web scraping.

Upon closer inspection of the source and network traffic, it was discovered that laundryview.com makes an HTTP GET request to an API every 30 seconds, which has all the washing and drying machine information for each residential hall, which it uses to display the washing machine availability on its website. This process was implemented using a JavaScript function which explains why laundry machine information was not available in the HTML code. As a result, web scraping was not a viable solution for accessing this data and using it in the text bot. The API provided information about washing machines in each laundry room, for example, the type of the machine(whether it is a dryer or washer), the orientation of the machine(which way it is faced in the dorm), the average time it takes for the machine to complete one cycle, and most importantly, the availability or how much time was left for a cycle to be completed. Those four fields are sufficient to collect for our server.

Luckily, we were also able to make a GET request to the API and receive a response. The API response contained data in JSON formation, which was structured in a way that made it easy to extract the information about each machine. The data was organized with each machine's information contained within curly brackets, and the fields within the machine were separated by commas. The name of the field was separated by its value with a colon, allowing for the use of split methods to access and manipulate the data as needed. If we did not have permission to request information from the API, then we would have had to consider alternative methods such as using Selenium to scrape the website data. But fortunately, we could make an HTTP GET request directly to access it to save all the trouble.

In order to provide the user with updated information about machine availability, our application focuses on the specific fields from the API response "appliance_type", which determines whether a machine is a washer or dry, "time_remaining", which gives the time remaining the current cycle in minutes; and "time_left_lite", which tells whether the machine is available. The "time_left_lite" field, discussed briefly above, is the field that is the main source that we are analyzing. If the "time_remaining" attribute that is with the same machine of the "time_left" field is non-zero, be a number k, indicating that the machine is experiencing a current ongoing cycle, then the "time_left_lite" will have associated value "k minutes remaining". If k is 0, then the associated "time_remaining" will be "Available". Occasionally, the "time_left_lite" field can be unavailable, due to maintenance and other issues.

The key challenge was to extract the relevant data fields from the API and store them in the format described above. We wrote a few methods to successfully extract the relevant information. The first method consists of connecting to the API by submitting an HTTP GET request by establishing a connection. Then we used a Bufferreader as a scanner to store the attributes or fields into a string. Using the Java String Library function ReadLine(), we can convert all the information on the API into a string format: a single string that contains all the lines in the API.

The next step involves converting the JSON string data into a list of hashmaps. After examining the API, we determined that it would be beneficial to first transform the entire string into a list of hashmaps, with the final element in the list including a key of objects and a value consisting of all the information combined into a single string. This is because the JSON string contains that information that is not about the laundry machines and is instead used in the website formatting for laundreview.com. Our server will not require the formatting attributes as no visualization is required in the project.

After converting the JSON string into a hashmap, we can extract all the relevant fields that have information about every laundry machine in the residential hall. The next step is to create a list of strings, with each element in the list containing all the information about a single laundry machine. The method to obtain each string in the list is not too complicated: as mentioned before, information for each laundry machine is contained within a pair of curly brackets, so a simple two-pointer technique can be used to identify when a closing curly brace follows an opening curly brace, which means that the information between the two braces is the laundry machine information. Each string of laundry machine information is added to the list.

The fourth step is to format the output of each element in the list, obtained by the third step. Since the original API response included field names and values separated by colons, we chose to use a hashmap to implement this function, as colons are used to distinguish the key-value pair in a HashMap. In each element of the list, every field is separated by a comma, so we split the string by comma, then we used a Java hashmap built-in function withKeyValueSeperator(), to distinguish the key and the value to convert the string formatting into a hashmap with a key-value pair for each field of the machine. This step will result in a list of hashmaps, with each hashmap containing the fields and values from the API response.

The last step is to extract relevant fields of the data and put them into a specific format that was mentioned previously: the current time, the residential hall, the number of washers available, and when the next washing machine will be available if there are currently no washers available. Using the corresponding key that represents the field, we extract all the information stated above, and put it in the following format: [current time, a residential hall, the number of washers available, and when washer will be available (minutes) if there are currently no washers available]. The format will be per residential hall per line. At this stage, the process of extracting the information is completed, next part is to have the properly formatted data into a CSV file.

One of the main reasons we choose to use a CSV file to store the data rather than using a database, as in the original proposal, is because our data is simple and structured, consisting of only four columns, and does not require any modification or querying, If our application had more complex, interrelated data and needed to handle frequent updates, then a database would be a more suitable choice. Additionally, scalability is not a concern in this part of our application, as there are a limited number of laundry rooms at Boston College.

The final two steps involve extracting and storing the information in the CSV file. It was decided that instead of adding new entries or updating the CSV file when we send a new GET request to the API that has data, we will just create a new CSV file whenever a new connection is

created, as modifying the CSV requires making a copy of the CSV file, which we believe it is more complicated than simply creating a new CSV file, repeating the steps listed above. Currently, we have designed to refresh the information every 10 minutes. During our presentation, some of our peers pointed out that a 10-minute refreshing period may cause inaccurate information provided to the user. There are a few reasons behind our choice: the laundry cycle takes on average of 30 minutes to wash and 60 minutes for the drying cycle. It is also very unlikely that there are more than 2 people planning on doing the laundry at the same time, suppose both use our service. The biggest question is that we are concerned if we make too many requests to the API at a high frequency we may get denied access to the API data. A denial of access to the API will hinder our project and this is not a risk that we are willing to test.

The other responsibility of the application is to communicate with the user in order to inform them of the availability of the washing machines as well as to send them reminders. As mentioned previously the Twilio API was used to receive and send SMS messages. Once the application has received a message from a user the application must determine what the appropriate response should be. If the user is a new user the application sends them a welcome message and asks the user for their residence hall. In each message object, there is a field that contains the phone number of the sender. The program uses a hashmap, which keeps track of users by storing their phone number as the key and their residential hall as the value, this is vital for the rest of the application. If the program receives a message from a sender whose phone number is not in the hashmap it will ask them to send their residential hall. Currently, only the following residential halls are supported - Keyes South Hall on the Newton Campus; FitzPatrick Hall on upper campus; Williams Hall on upper campus; GreyCliff, Voute, Walsh, Mods, 2150, Stayer, and reservoir apartments (2nd floor only).

Once the program receives a residential hall in a message it will store the residential hall and phone number in the hashmap, and then send them a new message listing the functions of the text bot. Which is that the bot can inform them of the availability of the laundry machines and send them reminders. If the program receives a message that asks them about the availability of the washing machine then it will use the information from the CSV to send a message to the user, that either the washing machine is available or that a washing machine will be available in a certain number of minutes. The bot detects that a message is asking about availability by searching for keywords in the message. Some keywords that were used that would imply the user is asking about the availability are “when”, “washing machine” and “available”. If these words are detected in the message, then the program assumes the user is asking if a washing machine is available.

The program follows similar steps to get the data from the CSV as it did to create the CSV but in reverse. It parses the CSV into a list of lists and then converts the list into a list of hashmaps. The program uses the list of hashmaps to determine the availability of the laundry machine and if the washing machine is not available, then the shortest wait time in minutes for when a washing machine will be available. Once it has gotten the relevant data from the

hashmap the program uses the Twilio API create message function to create and send a message with the availability.

If a washing machine is not available then the program tells the user that a washing machine is not available but will be available in a certain number of minutes, and then asks them if they would like a reminder for when a machine becomes available. If the user sends a message containing the word yes, then the program uses the Twilio API message schedule function to schedule a message in the future to the user by using the “when next washing machine will be available" field from the CSV, which was stored in the hashmap, to determine the time that the message should be scheduled. One limitation is that Twilio can only schedule messages fifteen minutes into the future, which means that if a washing machine will be available in less than fifteen minutes, the text bot cannot schedule a reminder to be sent when the washing machine will be available.

If the washing machine detects a message with the keywords “started” or “timer” and “washing” or “washer”, then it will assume that the user wants to be reminded when their washing machine is done and will schedule a message to be sent after 33 minutes, which is how long a wash cycle usually takes.

One of the questions we have received regarding the justification of our service is that students can go to laundryview.com to view the availability directly instead of using the service. This is true, however, the text bot is more convenient since it remembers the user's residential hall, thus the user does not need to select it each time they visit the website. In addition, the text bot can send a reminder about when the next laundry machine will be available, which the website does not do. Although this is not expected to be a major problem, our service can be accessed without an internet connection, which is useful in the event that WiFi is down. Overall, our service is more accessible and convenient than directly going to the website.

We have successfully achieved most of what we set out to do in our initial proposal. In the initial proposal our milestone was as follows - “ to create a text bot that can tell users whether a washing or drying machine is available in their hall, as well as the closest available washing or drying machine … give a reminder of when the user should pick up their laundry”. Although the text bot does not inform the user of the dryer availability we realized this does not make sense, if a washing machine is available, then a drying machine will almost always be available, therefore it would be redundant, it is able to tell a user if/when a washing machine is available, and send them a reminder when a machine will be available or when their laundry is done. Although the number of residential halls was not specified in the proposal, our text bot supports ten residential halls currently across the campus. In the future, more residential halls could be added by following the framework that was used to add the initial ten, and if it was determined that there was a demand, then functionality could be added to tell the user of the nearest available washing machine. From the limited inquiry that was done, it seems that most students would prefer to wait for an available washing machine at their dorm than to travel to another dorm’s washing machine which they might not even have access to. On the day of the presentation, we recorded a

demo where the user was able to receive the availability of the washing machine in their dorm, as well as receive a reminder when a washing machine would become available.

We have learned various skills in this project. We have learned how to be skilled in using Java Twilio API to send and receive text messages. We have also learned other skills such as how to employ a divide-and-conquer approach to cast the data into proper fields which were explained in the paper earlier. We have also enhanced our problem-solving skills. For example, we changed our approach from using web scraping to obtain the relevant information, but we realized that would not work and found a new method by observing the network traffic to discover the fact that our server can make calls to API directly to access the data. Although web scraping was not the ultimate solution we used to retrieve the data from the website, it was our initial solution and so we also learned how to scrape a website.

We agree to share our reports/code/video with other students. We are hoping to create an application that can benefit all BC students so we would like as many people to be aware of our projects and our efforts to create a better environment for BC students. The best way to promote this idea is to share. Our video so more students are aware of our project and can use such a service. Our report can serve as detailed documentation of our choice of design, along with the comments we left in our coding project, in case future BC students are interested in optimizing our design.

Here's the link to our code: https://github.com/Mahad-Abdi/LaundryBot