COMP 10222 - Emerging Web Technologies

Lab #4 - SMS with FaaS

Due date: Sunday November 15th at 11:59pm

Worth: 14% of total grade

Marks: 100 marks total

Learning objectives

- Create a serverless (FaaS) SMS bot using Twilio
- Use a microservice (DBaas) to store application data

Requirements

We will create an SMS bot that when texted will modify a Redis database and send responses, and will also access at least one 3rd party API. You will also create an accompanying NodeJS application that continuously checks for changes to the Redis database, and will update a front-end website and SQLite database in response.

Your bot must include at least one command that accesses at least one 3rd party API. The precise syntax for the command is for you to choose, but you must document it. The API can be your choice... weather data, video game data, social media data, anything really, except the Yelp API, that isn't allowed for this one because we've already used it. A list of possible APIs is available here, though you're free to use APIs that aren't on this list. The command that accesses the API has to allow the user to specify an argument or arguments, for example latitidue/longitude, or a search string, etc.

Your bot must include one status update command **Update Status Message** where **Update** is the name of the command, **Status** is a one-word string argument (e.g. "Warning", "Error, etc.), and **Message** is a string (e.g. some sentence describing something). This command should insert this data into a Redis database somehow (exactly how, is up to you).

Your bot most include one help command that is invoked when the user sends a message containing exactly the text "helpme". This command should reply back to the user with as brief as possible a list of the available commands. If the user enters the text "help command", where "command" is the name of the command (e.g. "help search"), the bot should respond with instructions on how that specific command works (including one example of how it works).

Note that your bot does not need to account for different users using the bot from different phone numbers. It is possible to check the number of the person attempting to use the bot, and alter the behavior accordingly, and if you wish to do that, you're free to do so. But you can assume that the bot has only one user, and that any number accessing the bot is that one user.

All of the above "bot code" can be put directly into your Twilio FaaS using their web interface. You must also create a file called **server.js** which runs an Express web server that makes a webpage accessible at **http://localhost:3000/dashboard**. Your server.js file should regularly check the Redis database to see if any new status update commands have been received. If a new status update command has been received, a record should be inserted into an SQLite database table identical to that in Lab #1 with the status and message given by the user of the SMS bot, as well as a timestamp for the current date/time. When a user first accesses **http://localhost:3000/dashboard** they should initially be presented with a list of all status update records in the SQLite database (ordered most recent to least recent). When additional status update commands are found by the server via its regular checking of the Redis database, websockets should be used to communicate this new status update information to the front-end dashboard which should present an updated list containing the new status update.

Bot Documentation

Important: For this lab, given its open nature, this part is as important as the code you write. I won't award full marks if I can't use the bot successfully by using your documentation.

Create a file called **command.txt**, you can document your third-party API SMS bot command in this file. When documenting your third-party API command, give the following:

- A general definition of the command, for example, **Weather Address**
- A description of what the command does, for example, "Reports the weather for the given address"

- At least two example usages of the command, for example, **Address Hamilton, Ontario**, **Address Toronto, Ontario**

At the very top of the file, list the phone number where your SMS bot is hosted. I will test it using this phone number.

Hints

You can find an entire list of publicly available APIs here: https://github.com/toddmotto/public-apis.

Submission instructions

When you have completed the lab, put your source files and commands.txt file in a folder called **lab4.zip** and submit it in the Dropbox. You can copy your Twilio FaaS source into a file called source.js.

Important: I will need your account username/password so that I can add my phone number to your service (it requires verification). So in addition to these files, submit a file called account.txt with your Twilio username/password (and of course, pick a password you don't mind me seeing).

Ensure that your service has been deployed and is available by texting the phone number provided to you by Twilio. I will test it directly using this phone number.

Statement of Authorship

Copying others people work and claiming it as your own is a serious breach of ethics. If copied work is found, all parties involved will receive a failing grade as per departmental policy. Group work is encouraged but it is expected that you do your own work. If you do, you'll learn the material and feel better for it. Since all work submitted is assumed to be your own original work, you must include the following "Statement of Authorship" in every program file you submit for grading:

"StAuth10065: I John Doe, 123456 certify that this material is my original work. No other person's work has been used without due acknowledgement. I have not made my work available to anyone else."

- Use the exact text above. No substitutions.
- Replace John Doe with your name and the number 123456 with your student ID.
- Place this text as one line as close to the top of each document as possible.
- Include StAuth10065: at the beginning of each statement.
- Failure to include this statement will cause your work to be ineligible for grading.

Marking scheme

Server	20
Dashboard	20

API command	20
Status update command	20
Help command	10
Accessible at number	10
Total	/100

Labs submitted late will receive a 15% penalty per day. Labs more than 2 days late will receive a grade of zero. For example, a lab submitted at 11:01pm on the due date that was marked 80/100 would receive a mark of 80 - 15 = 65/100. A lab submitted at 11:01pm the day after the due date that was marked 65/100 would receive a mark of 65 - 30 = 35/100. A lab submitted at 11:01pm two days after the due date would receive a mark of zero.