Assignment 2: Building a chatbot app

COS60016: Programming for Development

Student ID: 103952780

Name: Timothy Bailey

Creating and building a chatbot – justifying the approach

Amazon Lex is a web service that allows for the construction of conversation interfaces (Mishra, 2019). The concept is to create a 'friendly' user experience whilst fielding a wide range of questions from users and processing the information to fulfil particular goals, known as **intents**. Within each intent, several **slots** provide a framework of information in a JSON element. A user's query directs the Lex engine to select a particular intent, and a range of queries are answered by the user, adjusting the JSON file each time to contain the completed information for that intent. It also carries this prefilled information into the next query.

Numerous challenges were encountered and overcome during the development of the Lex bot for this assignment. Despite the recent launch of Version 2 for Amazon Lex, Version 1 was used for this assignment. This was due to its more streamlined implementation with Amazon Lambda. Also, of the few bot examples given in the Lex documentation (Amazon Web Services, 2022), the *Book Trip* blueprint provided a similar structure to that required of in this assignment. From this starting point, significant changes were made to create a unique bot that met the specific requirements of this assignment:

- 3 intents – BookFlight, BookCar and BookHotel
- 5 slots per intent
- at least 4 utterances for each intent

At first, I created detailed **Slot Types** to validate inputs, only to realise later that this was the Lambda function's role – generic alpha-numeric slot types were mostly used instead. Also, it was important to ensure that all mentions of any slots were accurately and consistently typed, and that small changes made in one area were reflected throughout. I also created a new slot (BookFlight). I nicknamed my bot "Jarvis" to give it a name and therefore a 'friendly' face for user engagement (Kull, et al., 2021). After some initial testing, I reconstructed the bot a second time to ensure correct configuration. Continuous testing was carried out during this development phase as detailed below.

Amazon Lambda is another aspect of the Amazon Web Service family, a **Function as a Service** platform that deploys functions within a lightweight virtual machine, or container and can be used for the improvement of functionality both within the AWS suite and beyond (Chapin & Roberts, 2020). Nicknamed "JarvisValidation", the Lambda function was the most intensively developed aspect of this assignment. Starting with the BookTrip blueprint, significant changes were made to achieve the specific goals of the assignment. Firstly, non-essential elements were removed from the code, such as "GeneratePrice" and its related elements. Also, a new method and validation function for "BookFlight" were added – the significant testing process to ensure correct integration of all intents is detailed in the next section, with examples of code provided in Appendix 1. After much work on the Lambda function, the role of different aspects became clear through much hard work and research.

The final aspect of this project was the Django site for deploying the chatbot. Although hosting was outside the scope of this assignment, I included a video of the chatbot functioning in place of an actual bot.

Testing the chatbot – test-driven development and reflective practice

Test-driven development is a double-edged sword (Janzen & Saiedian, 2008). Whilst consistent testing can lead to polished code with minimal errors, the quality of the end product depends on the quality of your testing tools. A prime example of this is the JSON file used for testing in Amazon Lambda, which represents the input data for a test user (Amazon Web Services, 2022). Initially, I was unsure of this system, and initially received a Key Error when testing, since the dictionary did not match my Lambda function at all (Hansen, 2022). This simple error taught me the importance of quality testing tools, and not simply relying on inbuilt systems or templates. Understanding is key to effective test-driven development.

In "Programming AWS Lambda", Chapin and Roberts had this to say regarding testing:

> *"A good test suite, like the solid foundation of a house, provides a known baseline of system behavior on top of which we can build confidently… When integrated into a development workflow, that same test suite also encourages good practices by making it easier to maintain existing tests and add new ones."*
>
> (Chapin & Roberts, 2020)

Throughout the development process, it was essential to not just solve a problem, but to understand it, and fix errors at their source. KeyErrors and syntax errors were resolved simply, as the error messages were specific. As my familiarity with the code grew, so too did my understanding of the need for specific JSON input and the role of key/value pairs in handling incoming data from the Lex bot and producing a response (Moisi, 2018). As it represented a single test case, it was important not to rely solely on the JSON test input, although it did provide initial screening before testing conversational flow.

As well as solving specific errors in the Lambda function, it also became clear that the Lex bot sometimes contained inconsistencies with the Lambda function. This required a detailed analysis of the bot for errors. One method I discovered for troubleshooting the Lex bot occurred somewhat by accident. Whilst analysing the differences between Version 1 and Version 2 of the Lex console, I discovered that error checking in Version 2 is far superior. If I had an error that I could not find a solution to, I was able to migrate it to the Version 2 console and use the error checking there to find the errors. Whilst this was not an ideal fix, it did help me on at least two occasions. The error messages in the test chatbot showed the location of the error in Lambda or Lex, but it was my understanding of these two tools that helped me find and resolve the problem.

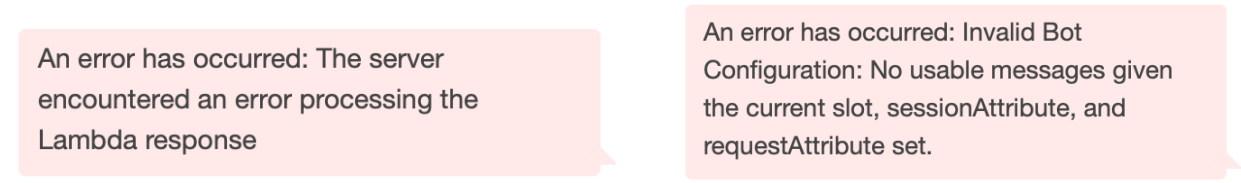


Figure 1: Two common error messages produced during testing of conversational flow in the Lex bot.

In considering the improvements I would make to this assignment, I would first suggest optimisation in the Lex bot's responses with further Lambda improvements (Amazon Web Services, 2022), and a handoff point for human assistance with difficult queries (Mamgain, 2022). Also, I discovered CloudWatch logs as a useful tool in the final stages of development, as well as in reviewing the development process. Fully searchable logs of events provided an abundant source of data for evaluating the development process. The table below shows the range of errors encountered in development of this project based on this data:

Table 1: Errors encountered during Lambda testing; Type and amount based on CloudWatch Log

Key Errors	Syntax Errors	Value Errors	Name Errors	Exceptions
5	14	1	12	4

Despite the challenges encountered during this assignment, especially in grasping a holistic system and the specificities of its data handling, it became clear that test-driven development was essential for this task. The immediate feedback provided by the aforementioned testing methods shaped any further development. As such, it was important to understand and modify testing procedures to steer the development process towards a successful product.

The GitHub repository for this assignment can be found here.

- Included are the assignment report, Lambda and Lex downloads, the CloudWatch log of all errors during testing and production, and a video showing my final successful test of the chatbot.

Bibliography

Amazon Web Services - Documentation, 2022. *aws-lambda-developer-guide.* [Online]
Available at: https://github.com/awsdocs/aws-lambda-developer-guide/blob/main/sample-apps/blank-python/function/lambda_function.py
[Accessed 15 06 2022].

Amazon Web Services, 2022. *Amazon Lex Documentation.* [Online]
Available at: https://docs.aws.amazon.com/lex/index.html
[Accessed 5 June 2022].

Amazon Web Services, 2022. *AWS Lambda Documentation.* [Online]
Available at: https://docs.aws.amazon.com/lambda/index.html
[Accessed 05 06 2022].

Amazon Web Services, 2022. *Boto3 documentation.* [Online]
Available at: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
[Accessed 25 06 2022].

Amazon Web Services, 2022. *How can I change the flow of my Amazon Lex bot using an initialization/validation or fulfillment AWS Lambda function?.* [Online]
Available at: https://aws.amazon.com/premiumsupport/knowledge-center/lex-dialogflow-fulfillment-lambda/
[Accessed 15 06 2022].

Chapin, J. & Roberts, M., 2020. *Programming AWS Lambda.* 1st Edition ed. Sebastopol: O'Reilly Media, Inc..

Hansen, C., 2022. *Python KeyError Exceptions and How to Handle Them.* [Online]
Available at: https://realpython.com/python-keyerror/
[Accessed 15 06 2022].

Janzen, D. S. & Saiedian, H., 2008. Does Test-Driven Development Really Improve Software Design Quality?. *IEEE Software,* 25(2), pp. 77-84.

Kull, A. J., Romero, M. & Monahan, L., 2021. How may I help you? Driving brand engagement through the warmth of an initial chatbot message. *Journal of business research,* 135(10), pp. 840-850.

Kumar, A. e. a., 2017. Just ASK: Building an Architecture for Extensible Self-Service Spoken Language Understanding. *arXiv preprint Arxiv,* Issue 1711.00549.

Mamgain, D., 2022. *Chatbot Human Handoff: Seamless Human takeover.* [Online]
Available at: https://www.kommunicate.io/blog/chatbot-human-handoff/
[Accessed 25 06 2022].

Mishra, A., 2019. *Machine Learning in the AWS Cloud.* 1st Edition ed. Indianapolis, Indiana: John Wiley & Sons Inc..

Moisi, A., 2018. *Building More Intelligent Lex Bots with Lambda Integration.* [Online]
Available at: https://blogs.perficient.com/2018/10/08/lexbots/
[Accessed 15 06 2022].

Velotio Technologies, 2019. *Amazon Lex + AWS Lambda: Beyond Hello World.* [Online]
Available at: https://medium.com/velotio-perspectives/amazon-lex-aws-lambda-beyond-hello-world-1403c1825e72
[Accessed 15 06 2022].

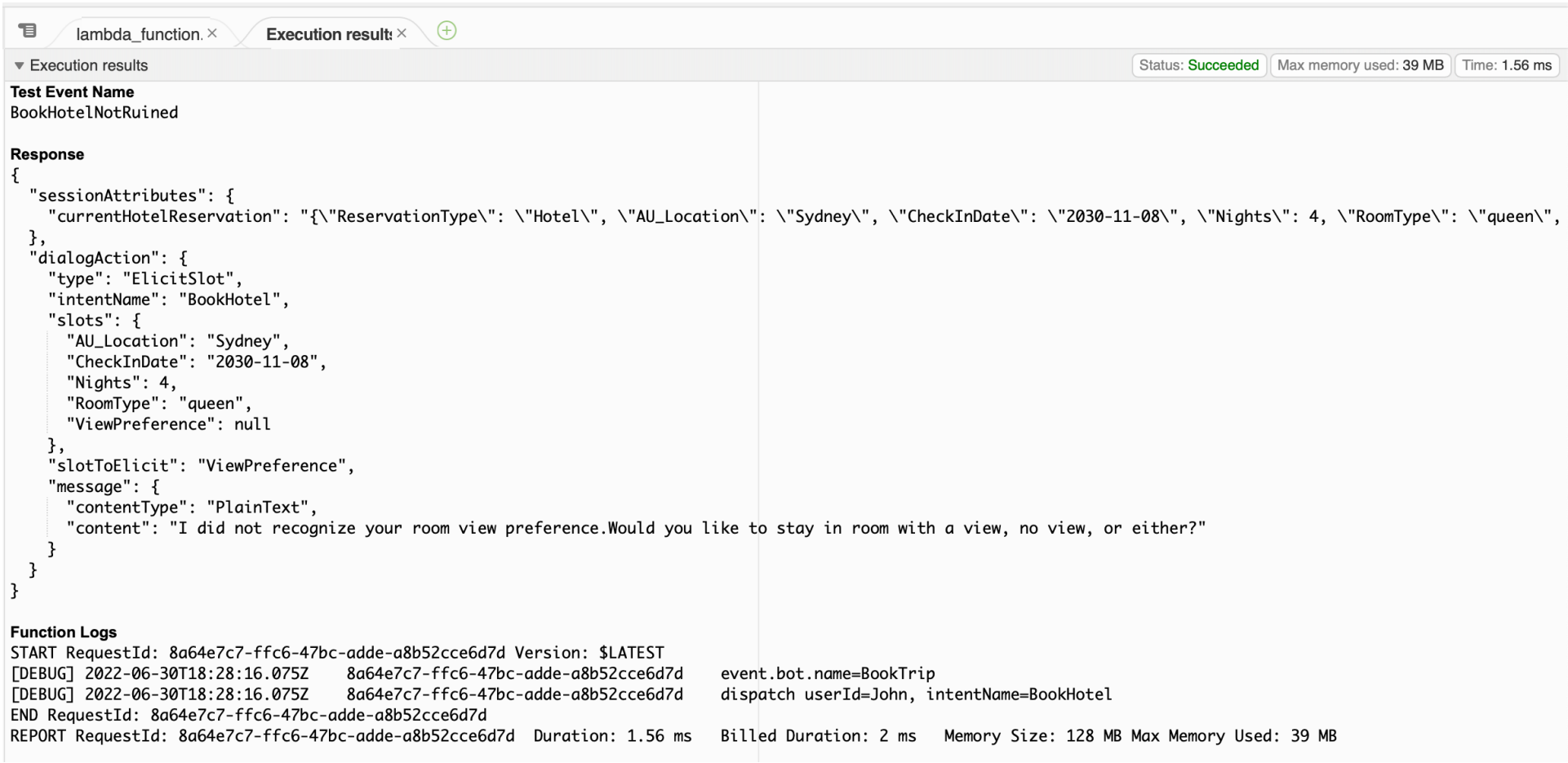
Appendix 1

Event JSON – BookCar

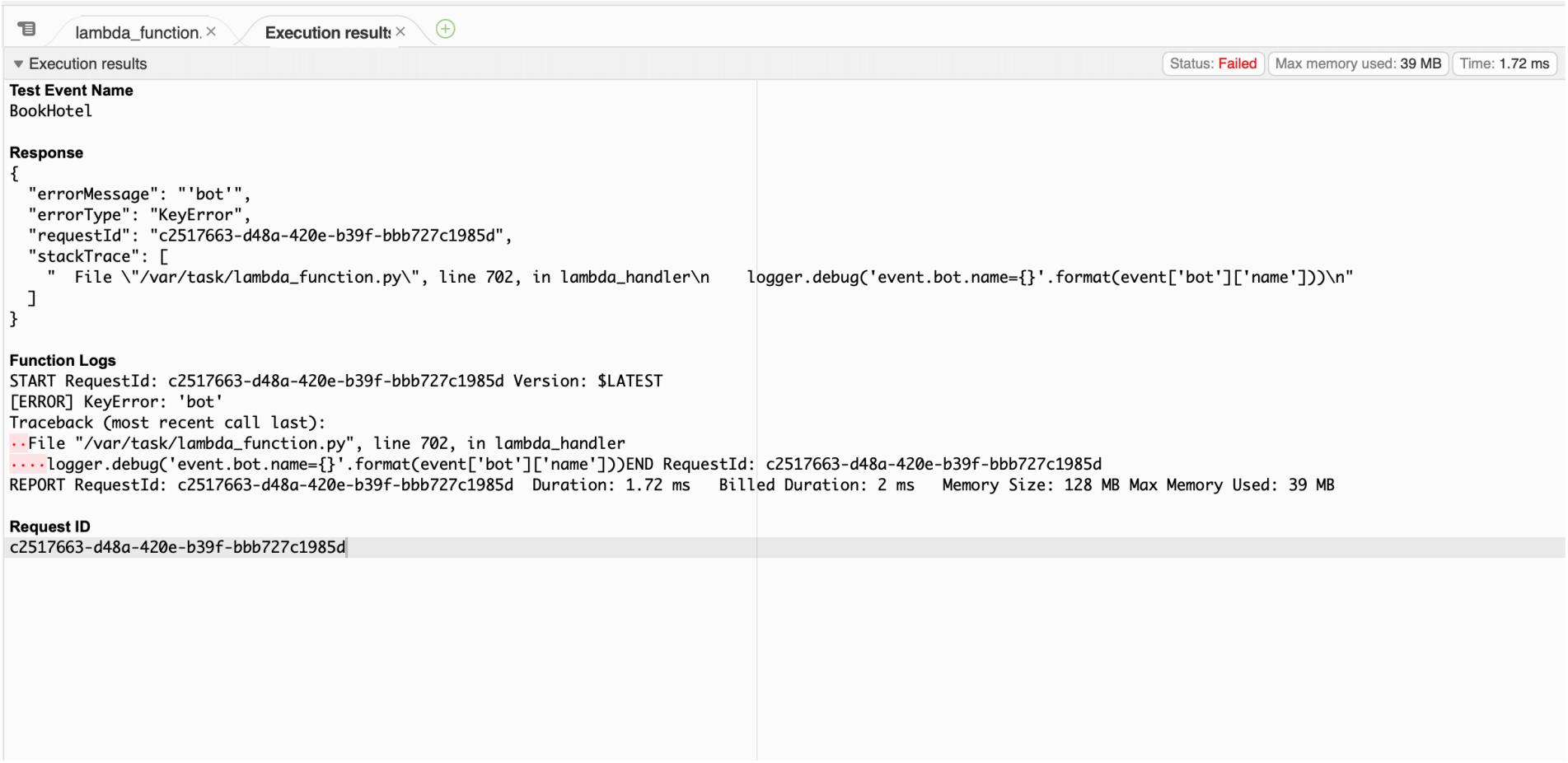
All information has been inputted as it would be expected to come from the user.

Event JSON

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "John",
  "sessionAttributes": {},
  "bot": {
    "name": "BookTrip",
    "alias": "$LATEST",
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookCar",
    "slots": {
      "PickUpCity": "Sydney",
      "PickUpDate": "2030-11-08",
      "ReturnDate": "2030-11-08",
      "CarType": "economy",
      "DriverAge": 21
    },
    "confirmationStatus": "None"
  }
}
```



A successful test using the above JSON input:



An unsuccessful test message due to incorrect test JSON configuration: