ECM3408 Enterprise Computing
Continuous Assessment 2021-22
Alexa, Go Shoot Yourself in the Head

David Wakeling

Handed Out	Handed In
Wednesday 9th February 2022 (T2:04)	Wednesday 23rd March 2022 (T2:10)

This Continuous Assessment is worth 40% of the module mark. It is an **individual assessment**, to be submitted using both E-Submit and BART. Your attention is drawn to the College and University guidelines on collaboration and plagiarism.

1 Background

Amazon engineers working on the third version of the Kindle e-book reader were puzzled by the insistence of then-CEO Jeff Bezos that it should continue to have a microphone, although there was no obvious use for it [1]. The only explanation that they could get from Bezos — a confirmed *Star Trek* enthusiast — was that in future he thought users would talk to their devices. Although the microphone was eventually dropped from the Kindle, Bezos remained convinced that Amazon should produce a cheap device that was controlled by voice with its brains in the cloud.

Eventually, the production of this device was designated Project Doppler. A tug-of-war soon developed between the project developers, who saw playing music as a practical and marketable feature, and Bezos, who was continuing to look beyond this to a *Star Trek*-like artificial intelligence that could answer questions and serve as a personal assistant. Unsurprisingly, Bezos prevailed.

Sadly, the early *Alexa* devices were considered to be slow and dumb by the Amazon employees who were given them for testing and had to record their conversations with their "Pringles can."

Even Bezos became frustrated. Dismayed by the lack of comprehension shown by a test unit in his Seattle home, he was recorded telling *Alexa* to "go shoot yourself in the head."

Bezos became impatient. Amazon rented homes and apartments in Boston and ten other cities, and installed muted Alexa prototypes disguised as "decoy" devices, such as Xbox consoles and TVs. A steady stream of contract workers then passed through the properties, reading canned scripts from an iPad. The resulting "mushroom cloud" of training data catapulted Amazon's *Alexa* ahead of Apple's *Siri* and Google's *Assistant*. Once again, the reward for Bezos — a stubborn, relentless man, prepared to burn billions of pounds on research if he believed an idea would work — was huge sales. Amazon has sold more than 100 million *Alexas* since its launch.

This Continuous Assessment is about producing an Alexa-like Minimum Viable Product (MVP) that is controlled by voice with its brains in the cloud, just as Bezos envisaged.

2 Assessment

The assessment is organised into four parts. These are described below, and also in a short video at `http://youtu.be/GS9RQcLVJ-Q`.

2.1 Part 1: An Alpha Microservice

Show some Go source code for an *Alpha* microservice that provides computational knowledge. This microservice takes a JSON object that has a "`text`" property whose value is a question string, and returns a JSON object that has an "`text`" property whose value is an answer string. The answer must be a knowledgeable one to the question. See Figure 1.

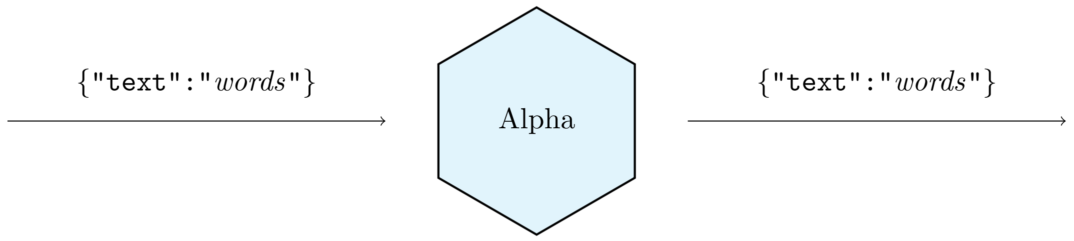

Figure 1: The Alpha microservice.

This *Alpha* microservice may be tested with the following "`alphatest.sh`" script.

```
#!/bin/sh
JSON="{\"text\":\"What is the melting point of silver?\"}"
JSON2=‘curl -s -X POST -d "$JSON" localhost:3001/alpha‘
echo $JSON2
```

Here, the expected output is `{"text":"961.78 degrees Celsius"}`.

To keep the implementation of this microservice manageable, it should be based on the Short Answers API of the Wolfram Alpha computational knowledge engine. You will need to signup to get an AppID to use this service, but this is free.

Here (and elsewhere), your microservice implementation will be judged on the clarity of its source code and the correctness of its operation (including error handling). By the clarity of source code, we mean the use of appropriate functions, statements and expressions, and the choice of meaningful identifiers. By the correctness of operation, we mean the processing of inputs to give outputs (including the signalling of invalid input, errors in processing or failures of communication).

	0	1	2	3	4	5	6	7	8	9	10
clarity											
correctness											

(20 marks)

2.2 Part 2: A STT Microservice

Show some Go source code for a *STT* microservice that provides speech-to-text conversion. This microservice takes a JSON object that has a "`speech`" property whose value is a WAV sound encoded as a string using the Base64 scheme, and returns a JSON object that has a "`text`" property whose value is a text string. The text must correspond to the speech. See Figure 2.

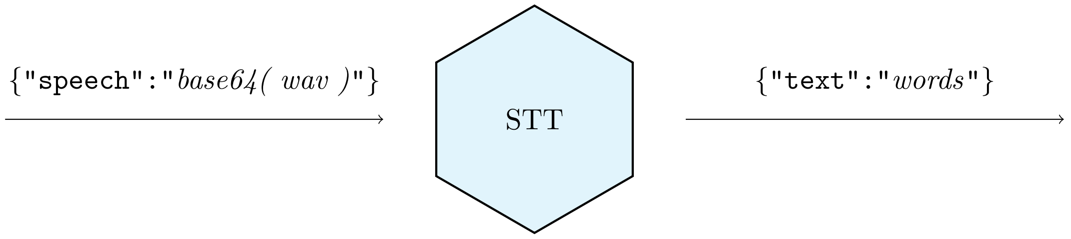

Figure 2: The STT microservice.

This *STT* microservice may be tested with the following "`stttest.sh`" script.

```
#!/bin/sh
JSON="{\"speech\":\"`base64 -i speech.wav`\"}"
JSON2=`curl -s -X POST -d "$JSON" localhost:3002/stt`
echo $JSON2
```

Here, the expected output is `{"text":"`*words*`"}`.

To keep the implementation of this microservice manageable, it should be based on the Microsoft Azure speech-to-text service.

	0	1	2	3	4	5	6	7	8	9	10
clarity											
correctness											

(20 marks)

2.3 Part 3: A TTS Microservice

Show some Go source code for a *TTS* microservice that provides text-to-speech conversion. This microservice takes a JSON object that has a "`text`" property whose value is a text string, and returns a JSON object that has a "`speech`" property whose value is a WAV sound encoded as a string using the Base64 scheme. The speech must correspond to the text. See Figure 3.

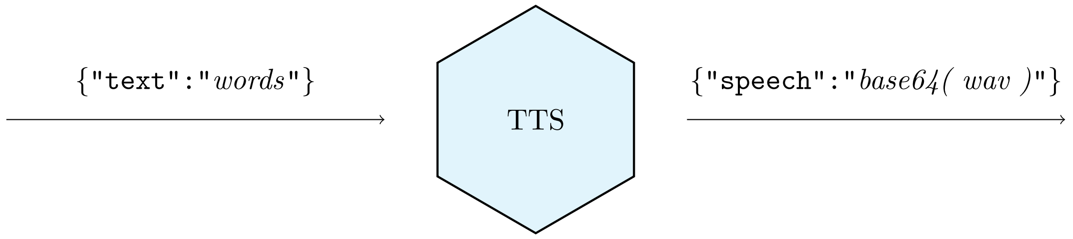

Figure 3: The TTS microservice.

This *TTS* microservice may be tested with the following "`ttstest.sh`" script.

```
#!/bin/sh
JSON="{\"text\":\"What is the melting point of silver?\"}"
JSON2=‘curl -s -X POST -d "$JSON" localhost:3003/tts‘
echo $JSON2
```

Here, the expected output is `{"speech":"`*Base64 encoding*`"}`.

To keep the implementation of this microservice manageable, it should be based on the Microsoft Azure text-to-speech service.

	0	1	2	3	4	5	6	7	8	9	10
clarity											
correctness											

(20 marks)

2.4 Part 4: An Alexa Microservice

Show some Go source code for an *Alexa* microservice that provides digital assistance. This microservice takes a JSON object that has a "`speech`" property whose value is a WAV sound encoded as a string using the Base64 scheme. This speech should be a question. The microservice returns a JSON object that has a "`speech`" property whose value is again a WAV sound encoded as a string using the Base64 scheme. This speech should be a knowledgeable answer to the question. See Figure 4.

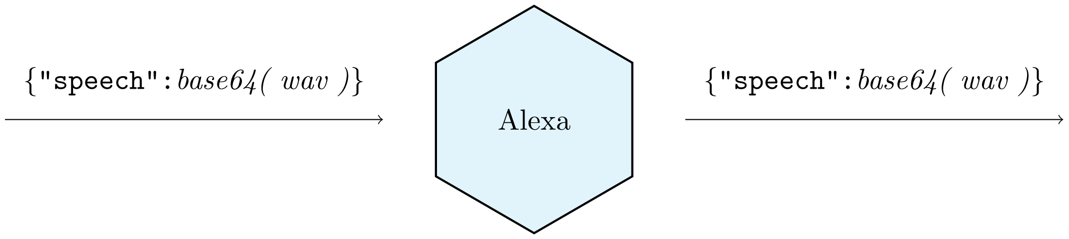

Figure 4: The Alexa microservice.

This *Alexa* microservice may be tested with the following "`alexatest.sh`" script.

```
#!/bin/sh
JSON="{\"speech\":\"`base64 -i question.wav`\"}"
JSON2=`curl -s -X POST -d "$JSON" localhost:3000/alexa`
echo $JSON2 | cut -d '"' -f4 | base64 -d > answer.wav
```

Here, it is expected that "`answer.wav`" will contain an answer to the question in "`question.wav`."

To keep the implementation of this *Alexa* microservice manageable, it should use the *Alpha*, *STT* and *TTS* microservices developed earlier.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20
clarity																				
correctness																				

(40 marks)

3 Submission

You should submit a single ".zip" file (note, *not* 7zip or WinRAR) containing the Go source code of your microservices by midday on *Wednesday 23rd March 2022.*

References

[1] B. Stone. *Amazon Unbound: Jeff Bezos and the Invention of a Global Empire.* Simon and Schuster, 2021.