

Services and Applications for a Healthy Ageing

Servicios y Aplicaciones para Envejecimiento Saludable



Este proyecto ha sido concedido por la Agencia Estatal de Investigación del Ministerio de Ciencia e Innovación dentro del procedimiento de concesión de ayudas a 'Proyectos de I+D+i' en el marco de los programas estatales de generación de conocimiento y fortalecimiento científico y tecnológico del sistema de I+D+i y de I+D+i orientada a los retos de la sociedad, convocatoria 2020.



Proyecto de investigación financiado por
MCIN/AEI/10.13039/501100011033

D.3.3: First prototype

WP3: Design and development of an intelligent conversational agent.



atlanTTic

research center
for Telecommunication Technologies



1- Introducción	3
2 - Estructura de la Skill.....	4
2.1. Importaciones y configuraciones iniciales	5
2.2. Manejadores de solicitudes.....	6
2.3. Manejadores de intenciones.....	7
2.4. Manejadores de errores	9
3 - Resumen.....	10
4 - Repositorio.....	11
5 - Personal participante.....	12
6 - Lista de acrónimos	13

1- Introducción

Este entregable presenta el código fuente de una skill de asistente virtual diseñada específicamente para replicar la Entrevista T-Moca (Test de Movilidad Cognitiva) en adultos mayores de 60 años. Esta prueba de evaluación genera una puntuación total con un máximo de 22 puntos, eliminando de la prueba original los ítems que requieran estímulos visuales o el uso de papel y lápiz.

La prueba MoCA original tiene un poder de discriminación similar al test Mini-Mental State Examination, actualmente el test de referencia para detectar el deterioro cognitivo. Por lo tanto, T-MoCA es un enfoque prometedor para diseñar una solución de detección para detectar deterioro cognitivo leve basada en asistentes de voz inteligentes.

La skill propuesta busca facilitar la administración y evaluación de la T-Moca de manera autónoma a través de dispositivos compatibles con Alexa de Amazon. En este entregable, se describe en detalle el código fuente de la skill, destacando los algoritmos utilizados para la presentación de preguntas, el registro de respuestas y la generación de puntuaciones según los criterios establecidos por la T-Moca.

2 - Estructura de la Skill

En esta sección del entregable se muestra en detalle el código de la skill que se organiza en varias secciones, cada una responsable de manejar diferentes tipos de solicitudes y lógica de negocio. A continuación, se describen las secciones principales del código:

- ★ Importaciones y configuraciones iniciales
- ★ Manejadores de solicitudes
- ★ Manejadores de intenciones
- ★ Manejadores de errores

2.1. Importaciones y configuraciones iniciales

Se ilustra en la Fig. 1 las librerías y dependencias necesarias de módulos y configuraciones iniciales, como la configuración del registro de eventos y la creación de instancias de objetos de servicio AWS.

```
000-importaciones.py x
1 import logging
2 import time
3 import os
4 import boto3
5 from datetime import datetime
6 import ask_sdk_core.utils as ask_utils
7
8 from ask_sdk_core.skill_builder import CustomSkillBuilder
9 from ask_sdk_core.dispatch_components import AbstractRequestHandler
10 from ask_sdk_core.dispatch_components import AbstractExceptionHandler
11 from ask_sdk_core.handler_input import HandlerInput
12
13 from ask_sdk_dynamodb.adapter import DynamoDbAdapter
14 # initialize persistence adapter
15 ddb_region = os.environ.get('DYNAMODB_PERSISTENCE_REGION')
16 ddb_table_name = os.environ.get('DYNAMODB_PERSISTENCE_TABLE_NAME')
17
18 ddb_resource = boto3.resource('dynamodb', region_name=ddb_region)
19 dynamodb_adapter = DynamoDbAdapter(table_name=ddb_table_name, create_table=False, dynamodb_resource=ddb_resource)
20
21 from ask_sdk_model import Response
22
23 from state import *
24 from f_words import *
```

Figura 1. Cabecera con las importaciones necesarias.

2.2. Manejadores de solicitudes

En esta sección, se muestran los manejadores de solicitudes para manejar las interacciones del usuario con la skill. Cada manejador se encarga de un tipo específico de solicitud, como el inicio de la skill, la solicitud de ayuda o la finalización de la interacción. Se ilustra a continuación Fig. 2, el manejador correspondiente al inicio de la skill, Digi-Moca.

```

1 class LaunchRequestHandler(AbstractRequestHandler):
2     """Handler for Skill Launch."""
3     def can_handle(self, handler_input):
4         # type: (HandlerInput) -> bool
5
6         return ask_utils.is_request_type("LaunchRequest")(handler_input)
7
8     def handle(self, handler_input):
9         # type: (HandlerInput) -> Response
10        logger.info("In LaunchRequestHandler")
11        attr = handler_input.attributes_manager.persistent_attributes
12        if not attr: # Init attr
13            attr['test_state'] = vars(State())
14            attr['test_state']['item'] = MEMORY
15            attr['test_state']['step'] = 1
16            attr['test_state']['date'] = datetime.today().strftime('%Y-%m-%d-%H:%M:%S')
17        else:
18            item = int(attr['test_state']['item'])
19            step = int(attr['test_state']['step'])
20            attr['test_state']['item'] = item
21            attr['test_state']['step'] = step
22            attr['test_state']['forward_numbers'] = int(attr['test_state']['forward_numbers'])
23            attr['test_state']['backward_numbers'] = int(attr['test_state']['backward_numbers'])
24            attr['test_state']['score'] = int(attr['test_state']['score'])
25            attr['test_state']['letter_mistakes'] = int(attr['test_state']['letter_mistakes'])
26            attr['test_state']['first_sentence'] = int(attr['test_state']['first_sentence'])
27            attr['test_state']['second_sentence'] = int(attr['test_state']['second_sentence'])
28            attr['test_state']['transport'] = int(attr['test_state']['transport'])
29            attr['test_state']['measure'] = int(attr['test_state']['measure'])
30            attr['test_state']['day'] = int(attr['test_state']['day'])
31            attr['test_state']['year'] = int(attr['test_state']['year'])
32            attr['test_state']['afaga'] = int(attr['test_state']['afaga'])
33            attr['test_state']['vigo'] = int(attr['test_state']['vigo'])
34            if item == ATTENTION and step == 36: attr['test_state']['calculations'] = [] #step == 16
35            #if item == LANGUAGE and step == 6: attr['test_state']['f_words'] = []
36
37        handler_input.attributes_manager.session_attributes = attr
38        handler_input.attributes_manager.save_persistent_attributes()
39
40        item = list(TEST)[attr['test_state']['item']]
41        step = attr['test_state']['step']
42        attr['test_state']['letters_last_time'] = str(time.time())
43        #attr['test_state']['f_start_time'] = str(time.time())
44        return handler_input.response_builder.speak(TEST[item][step-1]["speak"]).ask(TEST[item][step-1]["reprompt"]).response
45

```

Figura 2. Manejador que lanza la skill.

2.3. Manejadores de intenciones

En esta sección, se muestran los manejadores de intenciones que son los encargados de procesar las solicitudes de los usuarios basadas en las intenciones detectadas por Alexa. Cada intención puede ir asociada con una acción específica que la skill debe realizar.

Una intención representa un comportamiento que satisface la solicitud expresada por un usuario. Los comportamientos pueden incluir, por ejemplo, interactuar con otros módulos de software, actualizar elementos de datos locales o remotos o sintetizar una respuesta expresada. Por lo tanto, en esta fase, se especifica un conjunto de expresiones de muestra que se asignan a intenciones específicas.

Las intenciones definidas para esta skill son:

- *RememberIntentHandler*
- *YesIntentHandler*
- *NoIntentHandler*
- *DigitListIntentHandler*
- *NumbersIntentHandler*
- *FirstSentenceIntentHandler*
- *SecondSentenceIntentHandler*
- *TransporteIntentHandler*
- *MedirIntentHandler*
- *SemanaIntentHandler*
- *MesIntentHandler*
- *VigoIntentHandler*
- *AfagaIntentHandler*
- *FWordIntentHandler*
- *NextIntentHandler*
- *RepeatIntentHandler*
- *HelpIntentHandler*
- *CancelOrStopIntentHandler*
- *FallbackIntentHandler*
- *SessionEndedRequestHandler*
- *IntentReflectorHandler*

Se puede ver a continuación en la Fig. 3 dos de las intenciones definidas. El resto de las intenciones se pueden consultar en el github compartido en la sección, repositorio.

```

215
216
217 class YesIntentHandler(AbstractRequestHandler):
218
219
220
221
222 class NoIntentHandler(AbstractRequestHandler):
223     """Handler for No Intent"""
224     def can_handle(self, handler_input):
225         return ask_utils.is_intent_name("AMAZON.NoIntent")(handler_input)
226
227     def handle(self, handler_input):
228         logger.info("In NoIntentHandler")
229         logger.info(str(handler_input))
230         state = handler_input.attributes_manager.session_attributes['test_state']
231         if state['item'] == MEMORY:
232             if state['step'] == 1:
233                 return RepeatIntentHandler().handle(handler_input)
234
235         elif state['item'] == ATTENTION:
236             if state['step'] in [1, 3, 5, 35]:#15
237                 return RepeatIntentHandler().handle(handler_input)
238             elif state['step'] >= 6 and state['step'] <= 34:#14
239                 elapsed = time.time() - float(state['letters_last_time'])
240                 if elapsed <= TIME_THRESHOLD and LETTERS[state['step']-6] != 'A':
241                     state['letter_mistakes'] += 1
242                 state['step'] += 1
243                 state['letters_last_time'] = str(time.time())
244                 return handler_input.response_builder.speak(TEST['attention'][state['step']-1]['speak']).ask(TEST['attention'][state['step']-1]['reprompt'])
245
246         elif state['item'] == LANGUAGE:
247             if state['step'] in [1, 3, 5]:
248                 return RepeatIntentHandler().handle(handler_input)
249
250         return NextIntentHandler().handle(handler_input) # By default, if users says "no", we go to the next step
251
252
253 class DigitListIntentHandler(AbstractRequestHandler):
254     """Handler for Digit List Intent"""
255     def can_handle(self, handler_input):
256         state = handler_input.attributes_manager.session_attributes['test_state']
257         full_state = str(state['item']) + str(state['step'])
258         return (ask_utils.is_intent_name("DigitListIntent")(handler_input) and full_state in ['24', '22'])
259
260     def handle(self, handler_input):
261         logger.info("In DigitListIntentHandler")
262         state = handler_input.attributes_manager.session_attributes['test_state']
263         if state['item'] == ATTENTION:
264             if state['step'] == 2:
265                 state['forward_numbers'] = 1
266             elif state['step'] == 4:
267                 state['backward_numbers'] = 1
268
269         return NextIntentHandler().handle(handler_input)

```

Figura 3. Manejador que lanza la skill.

2.4. Manejadores de errores

En esta sección se muestran los los manejadores de errores implementados (Ver Fig. 4). Estos manejadores manejan cualquier error que pueda ocurrir durante la ejecución de la skill, proporcionando respuestas adecuadas al usuario y registrando información de errores para su posterior análisis.

```
001-exception.py x
1  class CatchAllExceptionHandler(AbstractExceptionHandler):
2      """Generic error handling to capture any syntax or routing errors. If you receive an error
3      stating the request handler chain is not found, you have not implemented a handler for
4      the intent being invoked or included it in the skill builder below.
5      """
6      def can_handle(self, handler_input, exception):
7          # type: (HandlerInput, Exception) -> bool
8          return True
9
10     def handle(self, handler_input, exception):
11         # type: (HandlerInput, Exception) -> Response
12         logger.error(exception, exc_info=True)
13
14         speak_output = "Perdone, he tenido un problema para hacer esto."
15
16         return (
17             handler_input.response_builder
18                 .speak(speak_output)
19                 .ask(speak_output)
20                 .response
21         )
```

Figura 4. Cabecera con las importaciones necesarias.

3 - Resumen

Este entregable ilustra el código en python para la implementación de una skill de asistente virtual para la plataforma de Amazon Alexa, diseñada específicamente para replicar la Entrevista T-Moca en adultos mayores. La skill presenta preguntas de la T-Moca, registra las respuestas del usuario y calcula la puntuación final según los criterios establecidos por la evaluación. La skill es fácilmente accesible a través de dispositivos compatibles con Alexa, permitiendo la administración autónoma y la evaluación de la T-Moca sin la necesidad de la presencia de un profesional de la salud.

Esencialmente, la implementación de la SKILL Digi-Moca ilustrada en este entregable se explica en los siguientes tres pasos:

- ★ **Archivo `lambda_function.py`** para recopilar todos los controladores que finalmente ejecute el servicio AWS Lambda para cada intención. Este código tiene siempre la misma estructura (Ver Fig. 3).
- ★ **La SKILL es una aplicación impulsada por eventos.** En pocas palabras, las SKILLS implementan métodos obligatorios. Estos métodos reciben como datos de entrada un diccionario Python codificado en JSON que contiene toda la información de la solicitud del usuario (Ver Fig. 4). Finalmente, la conversación avanza a la siguiente etapa al invocar a métodos que permiten actualizar los valores del elemento y avanzar al siguiente paso y enviar una nueva señal (es decir, la siguiente línea de la conversación escrita) al usuario. Estos elementos se almacenan en el objeto que se instancia de la clase State (Ver Fig. 5).
- ★ **El objeto Custom SkillBuilder** que recopila todas las clases de controlador de excepciones e intenciones creadas. La SKILL se ejecuta como un programa al invocar por el nombre definido.

4 - Repositorio

En esta sección se indica el enlace al repositorio donde se deposita todo el contenido de la skill. Se muestra a continuación el enlace a los distintos ficheros que forman parte de la skill.

[Github](#) - DigiMoca

5 - Personal participante

➤ Investigadores Principales (IPs)

- ❖ Luis Eulogio Anido Rifón (Uvigo)
- ❖ Luis Modesto Álvarez Sabucedo (Uvigo)

➤ Equipo de Investigación (EI)

- ❖ Manuel Fernández Iglesias (Uvigo)
- ❖ Martín Llamas Nistal (Uvigo)
- ❖ Manuel Gandoy Crego (USC)
- ❖ Marcos Pazos Couselo (USC)
- ❖ Manuel Caeiro Rodríguez (Uvigo)
- ❖ Juan Manuel Santos Gago (Uvigo)
- ❖ Fernando Ariel Mikic Fonte (Uvigo)

➤ Equipo de Trabajo (ET)

- ❖ Alba Elena Martínez Santos (USC)
- ❖ César Bugallo Carrera (USC)
- ❖ Sonia María Valladares Rodríguez (USC)
- ❖ Martín Liz Domínguez (Uvigo)
- ❖ Moises Ruben Pacheco Lorenzo (Uvigo)

➤ Personal Contratado (PC)

- ❖ Iván Otero González

6 - Lista de acrónimos

AWS	Amazon Web Services
JSON	JavaScript Object Notation
MoCA	Montreal cognitive assessment
PLN	Procesamiento del Lenguaje Natural
SaaS	Software as a Service
T-MoCA	Montreal Cognitive Assessment, Telephone version
Wi-Fi	Wireless Fidelity