FACE PRACTICA

CONSULTAS Y DEVERES

MIGUEL ALEJANDRO OCHOA YEPEZ

PREIMER SEMESTRE SOFTWARE

AZURE LUIS

Agregue la comprensión lingüística natural a sus aplicaciones

Diseñado para identificar información importante en las conversaciones, Lenguaje Understanding interpreta los objetivos del usuario (intenciones) y sintetiza información de valor de las oraciones (entidades), a fin de ofrecer un modelo de lenguaje de alta calidad y matizado. Lenguaje Understanding se integra perfectamente con el servicio de voz para procesar de forma instantánea la conversión de voz en intención. También se integra sin problemas con Azure Bot Service, por lo que facilita la creación de un Bot sofisticado.

Cree una solución de lenguaje personalizada con rapidez

Se combinan eficaces herramientas para desarrolladores con diccionarios de entidades y aplicaciones precompiladas personalizables, como calendario, música y dispositivos, para que pueda compilar e implementar una solución con más rapidez. Los diccionarios se extraen de conocimiento colectivo de la Web y se suministran miles de millones de entradas.

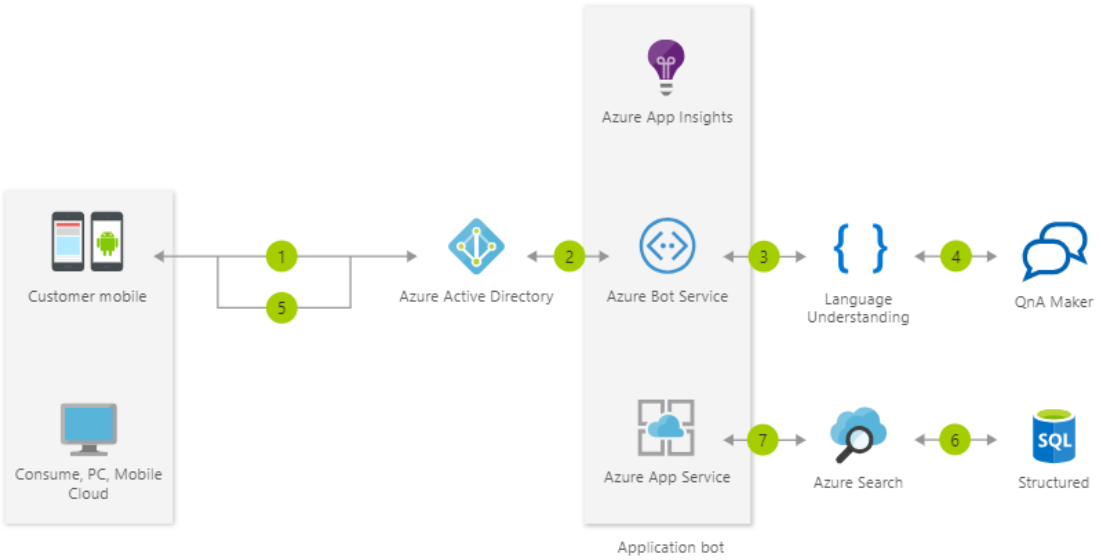
Aprendizaje y mejora constantes

El aprendizaje reforzado se usa para mejorar constantemente la calidad de los modelos de procesamiento del lenguaje natural. Una vez que el modelo empieza a procesar las entradas, Language Understanding inicia el aprendizaje activo, que permite actualizar y mejorar constantemente el modelo.

Preparado para la empresa y disponible en todo el mundo

El servicio está preparado para implementarse en aplicaciones comerciales y puede escalarse con una calidad y un rendimiento de nivel empresarial.

Bot de chat de información

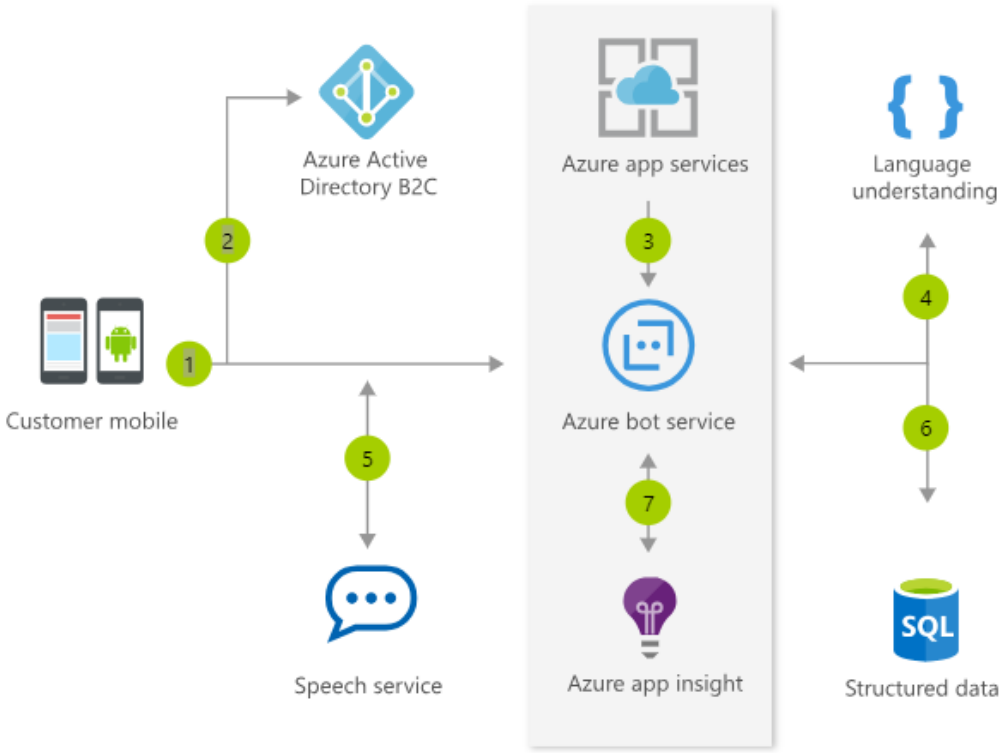


1. El empleado inicia el bot de aplicación.
2. Azure Active Directory valida la identidad del empleado.
3. El empleado puede preguntar al bot qué tipos de consultas se admiten.
4. Cognitive Services devuelve una pregunta frecuente creada con QnA Maker.
5. El empleado define una consulta válida.
6. El bot envía la consulta a Azure Search, que devuelve información sobre los datos de la aplicación.
7. Application Insights recopila telemetría de tiempo de ejecución para facilitar el desarrollo con el uso y el rendimiento del bot.

Bot de chat para operaciones comerciales

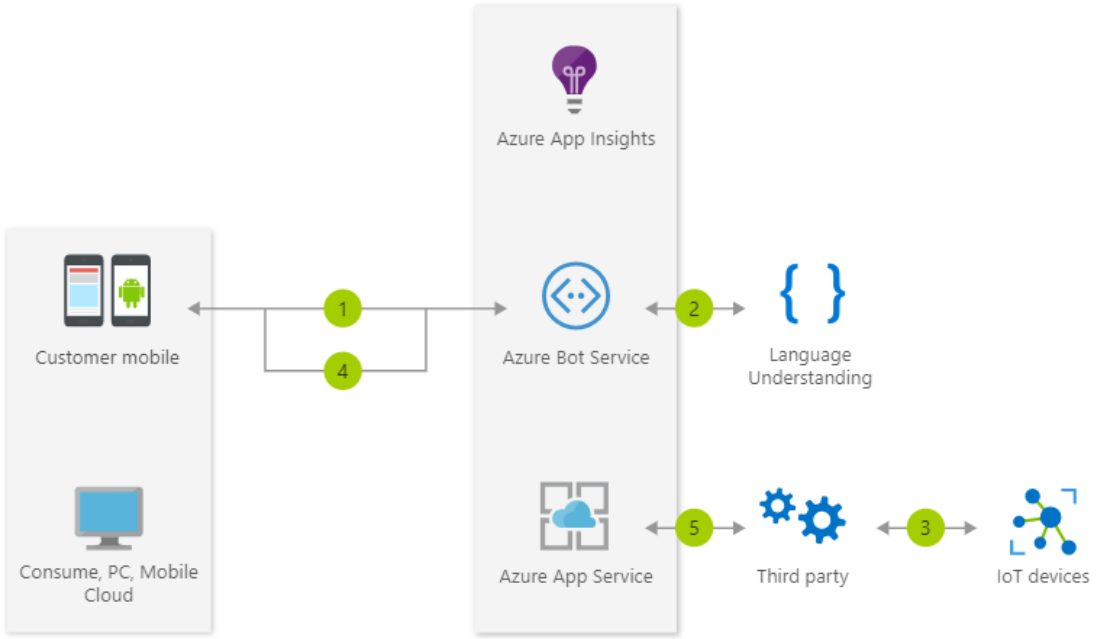
INSTITUTO SUPERIOR TECNOLOGICO DE TURISMO Y PATRIMONIO YAVIRAC

FACE PRACTICA



1. El cliente usa su aplicación móvil.
2. El usuario se autentica con Azure AD B2C.
3. El usuario solicita información con un bot de aplicación personalizado.
4. Cognitive Services ayuda a procesar las solicitudes de lenguaje natural.
5. El cliente revisa la respuesta, quien además puede matizar la pregunta mediante una conversación natural.
6. Una vez que el usuario está satisfecho con el resultado, el bot de aplicación actualiza la reserva del cliente.
7. Application Insights recopila telemetría de tiempo de ejecución para facilitar el desarrollo con el uso y el rendimiento del bot.

Dispositivos IoT



1. El usuario se registra en Skype y accede al bot de IoT.
2. El usuario pide por voz al bot que encienda las luces a través del dispositivo IoT.
3. La solicitud se retransmite a un servicio de terceros que tiene acceso a la red del dispositivo IoT.
4. Los resultados del comando se devuelven al usuario.
5. Application Insights recopila telemetría de tiempo de ejecución para facilitar el desarrollo con el uso y el rendimiento del bot.

C SHARP

Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

El nombre C Sharp fue inspirado por el signo '#' que se compone de cuatro signos '+' pegados.

C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows Microsoft, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

Literales:

Enteros	
decimal	`245` , `[0..9]+`
hexadecimal	`0xF5` , `0x[0..9, A..F, a..f]+`
entero largo	`12L`
entero largo sin signo	`654UL`
Coma flotante	
float	`23.5F` , `23.5f` ; `1.72E3F` , `1.72E3f` , `1.72e3F` , `1.72e3f`
double	`23.5` , `23.5D` , `23.5d` , `1.72E3` , `1.72E3D`
decimal	`9.95M`
Caracteres	
char	`'a'` , `'Z'` , `'\u0231'`
Cadenas	
String	`"Hello, world"` ; `"C:\\Windows\\"` , `@"C:\Windows\"`

Secuencias de escape	
Alerta (timbre)	`\a`
Retroceso	`\b`
Avance de página	`\f`
Nueva línea	`\n`
Retorno de carro	`\r`
Tabulador horizontal	`\t`
Tabulador vertical	`\v`
Nulo	`\0`
Comilla simple	`\'`
Comilla doble	`\"`
Barra inversa	`\\`

Variables:

Declarar una variable:

```
int miNumero; // Declaramos la variable, pero no la inicializamos con ningún valor.
```

Para asignar un valor a una variable, se indica el identificador de la misma, seguido del símbolo igual (`=`) y el valor que queremos que almacene:

```
miNumero = 5; // Asignamos el valor '5' a la variable creada.
```

Se puede declarar y asignar un valor al mismo tiempo:

```
int miNumero = 5; // Declaramos la variable, y asignamos el valor '5'.
```

También puedes declarar una variable sin especificar el tipo de dato, utilizando el mecanismo de inferencia mediante la palabra clave var donde el compilador determina el tipo de dato que se le asignará a la variable y sólo es permitida para variables locales, no para parámetros o datos miembro.

```
var cadena = "Esto es un string";

var numero1 = 5;

var numero2 = 4.5;

var numero3 = 4.5D;

var objeto = new Object();

var resultado = Math.Pow(5, 2);
long valor = 123; // Conversión implícita
long valor = (long)123; // Conversión explícita
```

Constantes:

Las constantes son valores inmutables, y por tanto no se pueden cambiar.

`const`

Cuando se declara una constante con la palabra clave `const`, también se debe asignar el valor. Tras esto, la constante queda bloqueada y no se puede cambiar. Son implícitamente estáticas (`static`).

```
const double PI = 3.1415;
```

`readonly`

A diferencia de `const`, no requiere que se asigne el valor al mismo tiempo que se declara. Pueden ser miembros de la instancia o miembros estáticos de la clase (`static`).

```
readonly double E;
E = 2.71828;
```

Operadores:

Categoría	Operadores
Aritméticos	`+` `-` `*` `/` `%`
Lógicos	`!` `&&` `||`
A nivel de bits	`&` `|` `^` `~`
Concatenación	`+`
Incremento, decremento	`++` `--`
Desplazamiento	`<<` `>>`
Relacional	`==` `!=` `<` `>` `<=` `>=`
Asignación	`=` `^=` `<<=` `>>=`
Acceso a miembro	`.`
Indexación	`[ ]`
Conversión	`( )`
Condicional	`? :` `??`
Creación de objeto	`new`
Información de tipo	`as` `is` `sizeof` `typeof`

Instrucciones de control:

`if-else`

```
if (i == 2)
{
    // ...
}
else if (i == 3)
{
    // ...
}
else
{
    // ...
}
```

`switch`

```
switch (i)
{
    case 1:
        ...
        break;
    case 2:
    case 3:
        ...
        break;
    default:
        ...
        break;
}
```

for

```
for (int i = 0; i < 10; i++)
{
    // ...
}
```

while

```
while (i < 10)
{
    // ...
}
```

do-while

```
do
{
    // ...
} while (true);
```

foreach

```
foreach (char c in charList)
{
    // ...
}
```

- Las instrucciones `if-else`, `for`, `while`, `do-while`, `switch`, `return`, `break`, `continue` son, básicamente, iguales que en C, C++ y Java.
- La instrucción `foreach`, al igual que en Java, realiza un ciclo a través de los elementos de una matriz o colección. En este ciclo se recorre la colección y la variable recibe un elemento de dicha colección en cada iteración.
- La instrucción `goto` se sigue utilizando en C# a pesar de la polémica sobre su uso.

Métodos:

- Todo método debe ser parte de una clase, no existen métodos globales (funciones).
- Por defecto, los parámetros se pasan por valor. (Nótese que las listas y otras colecciones son variables *por referencia* (referencias al espacio reservado para esa lista en la pila) y que se pasa por valor al método la referencia, pero el espacio reservado para la lista es común, por lo que si elimina un elemento lo hace también de la original).
- El modificador `ref` fuerza a pasar los parámetros por referencia en vez de pasarlos por valor y obliga a inicializar la variable antes de pasar el parámetro.
- El modificador `out` es similar al modificador `ref`, con la diferencia de que no se obliga a inicializar la variable antes de pasar el parámetro.
- Cuando `ref` y `out` modifican un parámetro de referencia, la propia referencia se pasa por referencia.
- El modificador `params` sirve para definir un número variable de argumentos los cuales se implementan como una matriz.
- Un método debe tener como máximo un único parámetro `params` y éste debe ser el último.

- Un método puede devolver cualquier tipo de dato, incluyendo tipos de clase.
- Ya que en C# las matrices se implementan como objetos, un método también puede devolver una matriz (algo que se diferencia de C++ en que las matrices no son válidas como tipos de valores devueltos).
- C# implementa *sobrecarga de métodos*, dos o más métodos pueden tener el mismo nombre siempre y cuando se diferencien por sus parámetros.
- El método `Main` es un método especial al cual se refiere el punto de partida del programa.

`ref`

```
void PassRef(ref int x)
{
    if (x == 2)
    {
        x = 10;
    }
}

int z = 0;
PassRef(ref z);
```

out

```
void PassOut(out int x)
{
    x = 2;
}

int z;
PassOut(out z);
```

params

```
int MaxVal(char c, params int[] nums)
{
    // ...
}

int a = 1;
MaxVal('a', 23, 3, a, -12); // El primer parámetro es obligatorio,
seguidamente se pueden poner tantos números enteros como se quiera
```

Sobrecarga de métodos

```
int Suma(int x, int y)
{
    return x + y;
}
int Suma(int x, int y, int z)
{
    return x + y + z;
}
```

```
int Suma(params int[] numeros)
{
    int Sumatoria = 0;
    foreach(int c in numeros)
        Sumatoria += c;
    return Sumatoria;
}

Suma(1, 2); // Llamará al primer método.
Suma(1, 2, 3); // Llamará al segundo método.
Suma(1, 2, 3, 4, 5, 6) // Llamará al tercer método.
```

Main

```
public static void Main(string[] args)
{
    // ...
}
```

Matrices:

Declarar una matriz:

```
int[] intArray = new int[5];
```

Declarar e inicializar una matriz (el tamaño de la matriz se puede omitir):

```
int[] intArray = new int[] {1, 2, 3, 4, 5};
```

Acceder a un elemento:

```
intArray[2]; // Retornará el valor '3'
```

Declarar una matriz multidimensional:

```
int[,] intMultiArray = new int[3, 2]; // 3 filas y 2 columnas
```

Declarar e inicializar una matriz multidimensional (el tamaño de la matriz se puede omitir):

```
int[,] intMultiArray = new int[,] { {1, 2}, {3, 4}, {5, 6} };
```

Acceder a un elemento de una matriz multidimensional:

```
intMultiArray[2, 0];
```

Clases y objetos:

Declarar una clase:

```
class Clase
{
}
```

Iniciar una clase (también llamado *crear un objeto de la clase* o *instanciar una clase*):

```
Clase c = new Clase();
```

Constructor (como si fuera un método, pero con el nombre de su clase):

```
class Clase
{
    Clase()
    {
        // ...
    }
}
```

Destructor (como si fuera un método, precedido del símbolo '`~`'):

```
class Clase
{
    ~Clase()
    {
        // ...
    }
}
```

`this`:

```
class Clase
{
    int i = 1;
```

```
    Clase()
    {
        this.Arrancar(); // Llamará al método 'Arrancar' del objeto
    }
    void Arrancar()
    {
        // ...
    }
}
```

`static`:

```
class Clase
{
    static int i = 1;
}
Clase.i; // Retornará el valor '1'. No hace falta crear un objeto, ya que al ser 'static', pertenece a la clase.
```

`operator`:

```
class Clase
{
    static int operator +(int x, int y)
    {
        // Sobrecarga el operador '+'
        // ...
    }
    static int operator -(int x, int y)
    {
        // Sobrecarga el operador '-'
        // ...
    }
    static int operator int(byte x)
    {
        // Sobrecarga la conversión de tipo 'byte' a 'int'
        // ...
    }
}
```

Comparación de objetos:

```
class Clase
```

```
{
}
Clase c1 = new Clase();
Clase c2 = new Clase();
bool b = c1 == c2;
```

Cadenas de caracteres:

Declarar una cadena de caracteres (como si fuera una variable de un tipo de dato como `int` o `double`):

```
string texto = "Cadena de caracteres";
string texto = new System.String("Cadena de caracteres"); //
Equivalente al anterior
```

Longitud de una cadena:

```
string texto = "Cadena de caracteres";
int i = texto.Length; // Retornará '20'
```

Comparar dos cadenas:

```
bool b = "texto" == "texto"; // Retornará 'true', ya que ambas cadenas
contienen "texto"
```

Concatenar cadenas:

```
string texto = "Cadena de" + " caracteres"; // Nótese el espacio antes
de "caracteres", si no se pusiera, aparecería junto: "decaracteres"
```

La clase `System.String`, y una instancia de la misma, `string`, poseen algunos métodos para trabajar con cadenas, como:

`static string Copy(string str)`: devuelve una copia de `str`.

```
string texto1 = "abc";
string texto2 = "xyz";
string texto2 = System.String.Copy(texto1); // 'texto2' ahora contiene
"abc"
```

`int CompareTo(string str)`: devuelve menor que cero si la cadena que llama es menor que str, mayor que cero si la cadena que llama es mayor que str, y cero si las cadenas son iguales.

```
string texto = "abc";
int i = texto.CompareTo("abc"); // Retornará '0'
```

`int IndexOf(string str)`: devuelve el índice de la primera coincidencia de la subcadena especificada en `str`, o -1 en caso de error.

```
string texto = "abcdefabcdef";
int i = texto.IndexOf("e"); // Retornará '4'
int j = texto.IndexOf("def"); // Retornará '3', que es donde se
encuentra el carácter 'd', seguido de 'e' y 'f'
```

`int LastIndexOf(string str)`: devuelve el índice de la última coincidencia de la subcadena especificada en `str`, o -1 en caso de error.

```
string texto = "abcdefabcdef";
int i = texto.LastIndexOf("e"); // Retornará '10'
int j = texto.LastIndexOf("def"); // Retornará '9', que es donde se
encuentra el último carácter 'd', seguido de 'e' y 'f'
```

`string ToLower`: devuelve una copia de la cadena en minúsculas.

```
string texto = "ABC";
string texto = texto.ToLower(); // Retornará "abc"
```

`string ToUpper`: devuelve una copia de la cadena en mayúsculas.

```
string texto = "abc";
string texto = texto.ToUpper(); // Retornará "ABC"
```

`string Substring`: devuelve una subcadena, indicando la posición de inicio y la longitud que se desea.

```
string texto = "Cadena de caracteres";
string texto = texto.Substring(10, 8);
```

Ejemplos:

"Hola mundo":

```
using System;

public class Ejemplo
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hola mundo");
    }
}
```

Suma y concatenación:

```
using System;

public class Ejemplo
{
    public static void Main(string[] args)
    {
        int x = 10;
        int y = 20;
        Console.WriteLine("El resultado es: " + (x + y)); // Imprimirá en pantalla: "El resultado es: 30"
    }
}
```

Uso de clases, métodos, propiedades y sobrecarga:

```
using System;

public class Coche
{
    private int numPuertas;
    public int NumPuertas
    {
        get
        {
            return this.numPuertas;
        }
        set
        {
            this.numPuertas = value;

        }
```

```
    }

    public Coche(int numPuertas)
    {
        this.NumPuertas = numPuertas;
    }

    // Sobrecarga: si se instancia la clase sin indicar ningún
parámetro, se inicializa 'numPuertas' con el valor '2'
    public Coche() : this(2)
    {
    }
}

public class Ejemplo
{
    public static void Main(string[] args)
    {
        Coche coche = new Coche(); // Se usa el segundo constructor
        coche.NumPuertas = 4;
        Console.WriteLine("El número de puertas es: " +
coche.NumPuertas);// Imprimirá en pantalla: "El número de puertas es:
4"
    }
}
```