IRIS: Asistente virtual para la redacción personalizada de correos electrónicos
IRIS: Virtual Assistant for Personalized Email Writing

Trabajo de Fin de Grado
Curso 2018–2019

Autor
Carlos Moreno Morera

Director
Raquel Hervás Ballesteros
Gonzalo Méndez Pozo

Doble Grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

IRIS: Asistente virtual para la redacción personalizada de correos electrónicos
IRIS: Virtual Assistant for Personalized Email Writing

Trabajo de Fin de Grado en Ingeniería Informática
Departamento de Ingeniería del Software e Inteligencia Artificial

Autor
Carlos Moreno Morera

Director
Raquel Hervás Ballesteros
Gonzalo Méndez Pozo

Convocatoria: Junio 2020
Calificación: *Nota*

Doble Grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

4 de octubre de 2019

A Pedro Pablo y Marco Antonio, por crear TeXiS
e iluminar nuestro camino

Acknowledgments

A Guillermo, por el tiempo empleado en hacer estas plantillas. A Adrián, Enrique y Nacho, por sus comentarios para mejorar lo que hicimos. Y a Narciso, a quien no le ha hecho falta el Anillo Único para coordinarnos a todos.

Resumen

IRIS: Asistente virtual para la redacción personalizada de correos electrónicos

Un resumen en castellano de media página, incluyendo el título en castellano. A continuación, se escribirá una lista de no más de 10 palabras clave.

Palabras clave

Máximo 10 palabras clave separadas por comas

Abstract

IRIS: Virtual Assistant for Personalized Email Writing

An abstract in English, half a page long, including the title in English. Below, a list with no more than 10 keywords.

Keywords

10 keywords max., separated by commas.

Contents

List of figures

List of tables

Chapter 1

Introduction

"Have you ever retired a human by mistake?"
— Rachael - Blade Runner (1982)

Smartphone development meant not only a technological advance but a social revolution too. They have brought with them countless paradigm shift in terms of social sphere. Since then, we are able to speak of a new model of human relationship both between people and with our technology. This current relation standard is due to the easily and quickly way to access to the different information that our mobile devices provide us. Long waits (nowadays the meaning of "long" waits has changed too, people consider more than two or three second too many time) for obtaining anything such as accessing to a website or showing any operation result, are excessively tedious and could be even frustrating for some smartphone users. When we are using our mobile, we want, as fast as possible, the information we are looking for. Precisely because of this, human-computer interaction (HCI) becomes a very important part in the process of development of the most of apps, not only in terms of speed of response and efficiency of our algorithms, but also in how we show the different informations and the easiness for obtaining them.

As for the relationships between people, as we have said, they have dramatically changed. There is no doubt that the main driving technologies behind this transformation of our relational paradigm are the social networks and the instant messaging. Focusing on the last type of mobile application we have mentioned, it is necessary to make a break-down of what consequences to our interpersonal interaction the instant communication have brought with itself. Just as it happens with the HCI, easiness and speed are probably the first features we look for when we are going to send any information to anybody. If we also expect a reply, the ideal would be to obtain it as quickly as possible. Therefore, in most of occasions, in practice we are looking for an automatic response from a human, what practically implies that everyone is "obligated" to be connected at any time with the answer we are asking for prepared.

1.1. Incentive

Introducción al tema del TFM.

1.2. Objectives

Descripción de los objetivos del trabajo.

1.3. Working plan

Aquí se describe el plan de trabajo a seguir para la consecución de los objetivos descritos en el apartado anterior.

1.4. Explicaciones adicionales sobre el uso de esta plantilla

Si quieres cambiar el **estilo del título** de los capítulos, edita `TeXiS\TeXiS_pream.tex` y comenta la línea `\usepackage[Lenny]{fncychap}` para dejar el estilo básico de LaTeX.

Si no te gusta que no haya **espacios entre párrafos** y quieres dejar un pequeño espacio en blanco, no metas saltos de línea (\\) al final de los párrafos. En su lugar, busca el comando `\setlength{\parskip}{0.2ex}` en `TeXiS\TeXiS_pream.tex` y aumenta el valor de $0.2ex$ a, por ejemplo, $1ex$.

TFMTeXiS se ha elaborado a partir de la plantilla de TeXiS[1], creada por Marco Antonio y Pedro Pablo Gómez Martín para escribir su tesis doctoral. Para explicaciones más extensas y detalladas sobre cómo usar esta plantilla, recomendamos la lectura del documento `TeXiS-Manual-1.0.pdf` que acompaña a esta plantilla.

El siguiente texto se genera con el comando `\lipsum[2-20]` que viene a continuación en el fichero .tex. El único propósito es mostrar el aspecto de las páginas usando esta plantilla. Quita este comando y, si quieres, comenta o elimina el paquete *lipsum* al final de `TeXiS\TeXiS_pream.tex`

1.4.1. Texto de prueba

[1] `http://gaia.fdi.ucm.es/research/texis/`

Chapter 2

State of the Art

2.1. Gmail API

In order to be able to read and send emails, it is necessary to access to the user's email data. For this reason, the different ways to obtain this information were studied. One of them is the Gmail API, which allows developers to perform all the actions we need in an easy way.

Gmail API can be used in several programming languages such as Python, PHP, Go, Java, .NET, ... Due to the greater number of examples in the starting guides of the Gmail API (Google, 2019a) and the previous knowledge that was already had of it, Python was chosen for the first contact with this implement.

The following is a step-by-step explanation of what is necessary to do to access the user's Gmail account, create a message, send an email previously created, create and update a draft, reply a received message (for this it is necessary to know how to create an email) and read important information of message threads and individual emails (such as who is the sender, who will received the message, the subject, the date, the email's body, the attached files, ...).

2.1.1. OAuth 2.0 Protocol

Gmail API, as it also happens in the case of other Google APIs, uses OAuth 2.0 protocol (Google, 2019f) to handle authentication and authorization. As it will be seen later in this section, it is needed to be in possession of OAuth 2.0 client credentials from the Google API Console for having the appropriate permissions to use the Gmail API.

The Google API Console, also known as Google Console Developer[1], built into Google Cloud Platform, makes possible an authorized access to a user's Gmail data. In order to achieve it, having a Google account is a prerequisite because it will be necessary to access to this platform. Once this web has been accessed, at first we have to create a new development project by clicking in "New Project" in the control panel (which is the main tab of the Google Console Developer and the one that opens by default when you access it). When we have already created a project, we will enable the API we are going to work with, in this case the Gmail API. To do this we will look for it in the search engine that we can find in the library of APIs of this platform. Now we can apply for the credentials we need. Accessing to the "Credentials" tab and clicking on "Create Credentials" will lead us

[1] `https://console.developers.google.com/`

to an easy questionnaire about what type of credentials we prefer that we have to answer by basing on what type of application we are building. Then we must download the .json file and save it in the folder we are going to work in.

Before starting the development of the implementation of the OAuth 2.0 protocol which will provides us a secure and trusted login system to access to the user's Gmail data, we must install the Google Client Library[2] of our choice of language (as we have said we will use Python for this, so we have to install the libraries *google-api-python-client*, *google-auth-httplib2* and *google-auth-oauthlib*).

There are many ways to obtain the necessary permissions for accessing to the user's emails data following the OAuth 2.0 protocol. As this is a first contact with the Gmail API only with the intention of knowing the possibilities it offers to us and its advantages and disadvantages of using it in our future implementation of our virtual assistant, we are going to develop a simple script which is using a class very useful for local development and applications that are installed on a desktop operating system. The class *InstalledAppFlow*, in *google_auth_oauthlib.flow* (Google, 2019b), is a *Flow* subclass (which belongs to the same library). Thanks to this last class we have mentioned, *InstalledAppFlow* uses a *requests_oauthlib.OAuth2Session* instance at *oauth2session* to perform all of the OAuth 2.0 logic. Besides it also inherits from *Flow* the class method *from_client_secrets_file* which creates a *Flow* instance from a Google client secrets file (this file will be the .json file that we obtained through the Google API Console) and a list of OAuth 2.0 Scopes (Google, 2019e), which are a mechanism in OAuth 2.0 to limit an application's access to a user's account. An application can request one or more scopes, this information is then presented to the user in the consent screen, and the access token issued to the application will be limited to the scopes granted (we will use the Gmail API OAuth 2.0 Scope which allows us to read, compose, send, and permanently delete all your email).

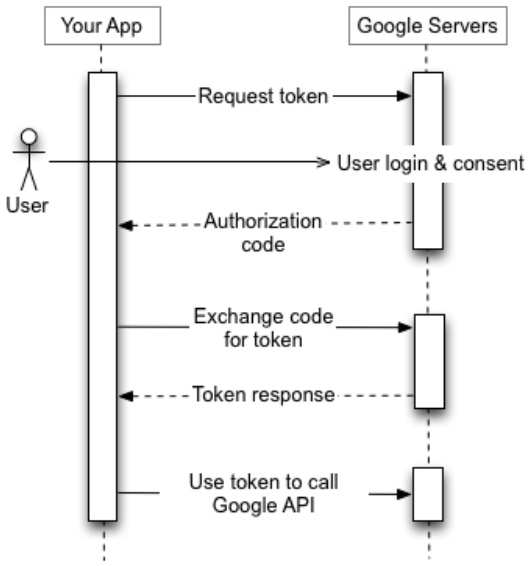

Figure 2.1: OAuth 2.0 for Web Server Applications and Installed Applications. Image extracted from Google (2019f)

After constructing an *InstalledAppFlow* by calling *from_client_secrets_file* as we have explained, we can invoke the class method *run_local_server* which instructs the user to

[2] `https://developers.google.com/gmail/api/downloads`

open the authorization URL in the browser and will attempt to automatically open it. This function will start a local web server to listen for the authorization response. Once there is a reply, the authorization server will redirect the user's browser to the local web server. As we can see in 2.1, the web server will get the authorization code from the response and shutdown, that code is then exchanged for a token.

Then, we will be in possession of the OAuth 2.0 credentials for the user (Google, 2019d) which we are going to use for accessing the user's Gmail account. In summary, it is possible to obtain the necessary permissions from the user and to follow the OAuth 2.0 protocol, by executing these instructions (written in Python):

```
from google_auth_oauthlib.flow import InstalledAppFlow

# Create a flow instance
flow = InstalledAppFlow.from_client_secrets_file('credentials.json',
    ['https://mail.google.com/'])
# Obtain OAuth 2.0 credentials for the user
creds = flow.run_local_server(port = 0)
```

Now, we are able to call Gmail API by using the token (which is stored in the variable *creds*). However, before starting working on the email data, we should save the OAuth 2.0 credentials since otherwise the user would need to go through the consent screen every time the application is opened. To prevent the latter from happening, to differentiate access from mail management and consequently to reuse as much code as possible; we have implemented the following class *auth*, in *auth.py*, with a main method *get_credentials*:

```
import pickle
import os.path
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request

class auth:
    def __init__(self, SCOPES, CLIENT_SECRET_FILE):
        self.SCOPES = SCOPES
        self.CLIENT_SECRET_FILE = CLIENT_SECRET_FILE

    def get_credentials(self):
        """
        Obtains valid credentials for accessing Gmail API
        """
        creds = None
        # The file token.pickle stores the user's access and refresh tokens
        if os.path.exists('token.pickle'):
            with open('token.pickle', 'rb') as token:
                creds = pickle.load(token)
        # If there are no (valid) credentials available, let the user log in
        if not creds or not creds.valid:
            if creds and creds.expired and creds.refresh_token:
                creds.refresh(Request())
            else:
                flow = InstalledAppFlow.from_client_secrets_file(
                    self.CLIENT_SECRET_FILE, self.SCOPES)
                creds = flow.run_local_server(port=0)
            # Create token.pickle and save the credentials for the next run
            with open('token.pickle', 'wb') as token:
                pickle.dump(creds, token)
        return creds
```

As we can observe in line 17 within *get_credentials* method, at first we check if the file called *token.pickle* exists, and in that case, it is opened and its information is stored in the variable *creds*. Thus, we avoid to force the user to open the authorization screen. By contrast, as we have seen before, if it does not exists, we obtain the credentials by calling the class methods *from_client_secrets_file* and *run_local_server* (it is written between lines 25 and 30).

There is another case that is also reflected in the code above (in lines 23 and 24): the credentials are expired (it is possible to check it by executing *creds.expired*) and they can be refreshed (the OAuth 2.0 refresh token is *creds.refresh_token*) (Google, 2019d). In this situation, we will refresh the access token by invoking the method known as *refresh* and by giving it a *Request* object (Google, 2019c) from *google.auth.transport.requests* as the function parameter which used to make HTTP requests.

2.1.2. Building a Gmail Resource

At this point, with the OAuth 2.0 credentials, we are able to call the Gmail API. For this purpose, it is necessary to construct a resource (Google, 2019a, /v1/reference) for interacting with the API. The *build* method, from *googleapiclient.discovery* library (Gregorio, 2019), create that object. As we will see later, this resource will lead us to manage emails, drafts, threads and everything we will like to do with the user's Gmail data. This is why, using the *auth.py* file explained in 2.1.1, we are going to start every user session with the instructions below (or their equivalents in the language we are using):

```
from googleapiclient.discovery import build
import auth

SCOPES = ['https://mail.google.com/']
CLIENT_SECRET_FILE = 'credentials.json'

# Creation of an auth instance
authInst = auth.auth(SCOPES, CLIENT_SECRET_FILE)
# Constructing the resource API object
service = build('gmail', 'v1', credentials = authInst.get_credentials())
```

Henceforth, we will use the *service* variable to relate it with the resource object created by the *build* method.

2.1.3. Messages resource

In most of the operations we are going to execute, it will be essential the correct management of messages. Therefore, knowing how the emails are represented in Gmail API and how to use them is imperative to understand how to work with this API. For this reason, in this section we are going to delve into the *Message* resource of the Gmail API, its structure and its methods.

Regardless of which programming language is used, *Message* resource (Google, 2019a, /v1/reference/users/messages) internally has a dictionary structure. The more important (for us) keys of this data structure are (at least the most useful keys for our purpose):

- *id*: an immutable string which identifies the message.
- *threadId*: we will explain the thread resource in 2.1.4 and we will see that a thread is composed of different messages that share common characteristics. The value of this field is a string which represent the id of the thread the message belongs to.

2.1.4. Threads resource

Chapter 3

Descripción del Trabajo

Aquí comienza la descripción del trabajo realizado. Se deben incluir tantos capítulos como sea necesario para describir de la manera más completa posible el trabajo que se ha llevado a cabo. Como muestra la figura 3.1, está todo por hacer.

Figure 3.1: Ejemplo de imagen

Si te sirve de utilidad, puedes incluir tablas para mostrar resultados, tal como se ve en la tabla 3.1.

Col 1	**Col 2**	**Col 3**
3	3.01	3.50
6	2.12	4.40
1	3.79	5.00
2	4.88	5.30
4	3.50	2.90
5	7.40	4.70

Table 3.1: Tabla de ejemplo

Chapter 4

Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

Antes de la entrega de actas de cada convocatoria, en el plazo que se indica en el calendario de los trabajos de fin de máster, el estudiante entregará en el Campus Virtual la versión final de la memoria en PDF. En la portada de la misma deberán figurar, como se ha señalado anteriormente, la convocatoria y la calificación obtenida. Asimismo, el estudiante también entregará todo el material que tenga concedido en préstamo a lo largo del curso.

Bibliography

Y así, del mucho leer y del poco dormir, se le secó el celebro de manera que vino a perder el juicio.
(modificar en Cascaras\bibliografia.tex)

Miguel de Cervantes Saavedra

Google. Gmail api | google developers. 2019a. [Online; accessed 23-September-2019].

Google. google_auth_oauthlib.flow module. 2019b. [Online; accessed 23-September-2019].

Google. google.auth.transport package. 2019c. [Online; accessed 23-September-2019].

Google. google.oauth2.credentials module. 2019d. [Online; accessed 23-September-2019].

Google. Oauth 2.0 scopes for google apis. 2019e. [Online; accessed 23-September-2019].

Google. Using oauth 2.0 to access google apis. 2019f. [Online; accessed 23-September-2019].

Gregorio, J. googleapiclient.discovery. 2019. [Online; accessed 23-September-2019].

Appendix A

Título del Apéndice A

Contenido del apéndice

Appendix B

Título del Apéndice B

Este texto se puede encontrar en el fichero Cascaras/fin.tex. Si deseas eliminarlo, basta con comentar la línea correspondiente al final del fichero TFMTeXiS.tex.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes