SkySpeak

Python Project - Cohort02

Table Of Contents

1. Objective of the App 3
2. App Functions 4
User interface 4
Testing functions 4
Administrative functions 4
Caching functions 4
Analytics functions 4
3. APIs Used 5
3.1 Ipify 5
3.1.1 End Points 5
3.2 IpStack 5
3.2.1 End Points 5
3.3 Google Translate 5
3.3.1 End Points 5
3.4 TextToSpeech 6
3.4.1 End Points 6
4. Data Model 7
4.1 ERD Diagrams 7
4.1.1 Conceptual Design 7
4.1.2 Logical Design 7
4.1.3 Physical Design 8
4.2 Tables Definition 9
4.2.1 IP_ADDRESSES 9
4.2.2 COUNTRIES 9
4.2.3 LANGUAGES 9
4.2.4 LANGUAGE_COUNTRY 9
4.2.5 TEXT_TYPES 10
4.2.6 TRANSLATIONS 10
4.2.7 REQUEST_LOG 10
5. Repository structure 11

1. Objective of the App

The objective of the app is to provide a personalised and convenient way of accessing weather information based on your location.
The app would use your IP address to determine your approximate geographical location. Then, the app would use a weather API to fetch the current weather conditions.
The app would then display and speak a friendly greeting message that includes your location and some relevant weather information, such as “Good Morning! It’s a sunny day with a temperature of 25°C.”
The text to speech functionality is a feature that can convert text into audio and play it through the device’s speaker or headphones. The text to speech functionality can make the app more accessible, interactive, and engaging for the users.
The app would store all translated and spoken text in normalised form so as to not have to make repeated requests to the several APIs involved in delivering the user their message.

2. App Functions

User interface

The user has the option to auto-resolve their IP-address and the associated location data. If more than one language is spoken in the user's country, they will be able to select a preference. Once a language is resolved or confirmed, the user will be greeted with a spoken personalised message in their language. The greeting will depend on their time of day and the weather in their current location. If the user is from a multilingual country, they can change their preferred language at any time.

Testing functions

Instead of identifying the user's IP-address, a custom IP address can be submitted to review the application's response. A list of known existing IP addresses from a variety of countries has been provided for convenience.

Administrative functions

The application provides an administrator page to review the sentences stored for translation and to reset the tables used by the application. While clearing the tables would not be part of normal application use, it may prove useful after testing. Similarly, manually inserting acceptable strings to be translated is not part of the normal application loop as this should be handled by the wrapping functions, but the function is included for testing purposes while integrating new APIs.

Caching functions

All translated text and generated sound chunks (for example, "Good morning" and "The weather is sunny" are both separate chunks) are cached to the database before being concatenated into a greeting. When a greeting is requested for a user, every part is retrieved from the cache if available. If not, the part is generated and cached. A greeting can be made up from both cached and newly generated chunks.

Analytics functions

An analytics page is available to access graphs based on aggregated metadata, like the locations of IP addresses that have used the apps or the languages most used in a region. Additionally, all requests for translation and speaking are logged including whether they could be answered from cache or had to be generated. This gives insight into the application's backend's primary function, that is to be increasingly less dependent on external APIs as time goes on.

3. APIs Used

3.1 Ipify

This API retrieves information about your ip address.

3.1.1 End Points

End Point	Method	Descr.
https://api.ipify.org/	GET	Retrieves the ip address of the machine

3.2 IpStack

This API retrieves geographical information about the given ip address.

3.2.1 End Points

End Point	Method	Descr.
http://api.ipstack.com/{ip}	GET	Geolocalises the given ip and retrieves informations about the country and spoken languages

3.3 Google Translate

This API translates a string from one language to another.

3.3.1 End Points

End Point	Method	Descr.
https://google-translate1.p.rapidapi.com/language/translate/v2/languages	GET	Returns a list of supported languages for translation.
https://google-translate1.p.rapidapi.com/language/translate/v2	POST	Translates input text, returning translated text

3.4 TextToSpeech

This API gets a sentence and returns the audio of the text in binary format.

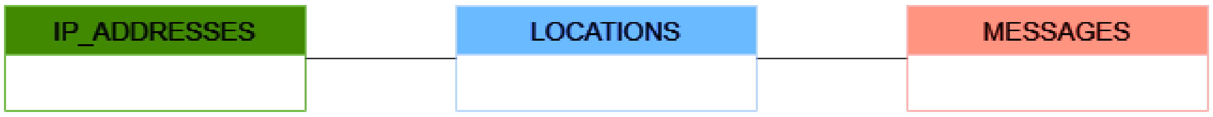
3.4.1 End Points

End Point	Method	Descr.
https://text-to-speech27.p.rapidapi.com/speech/lang	GET	Returns the list of supported languages on which TTS can be performed
https://text-to-speech27.p.rapidapi.com/speech	GET	Performs TTS on the given text in the params

4. Data Model

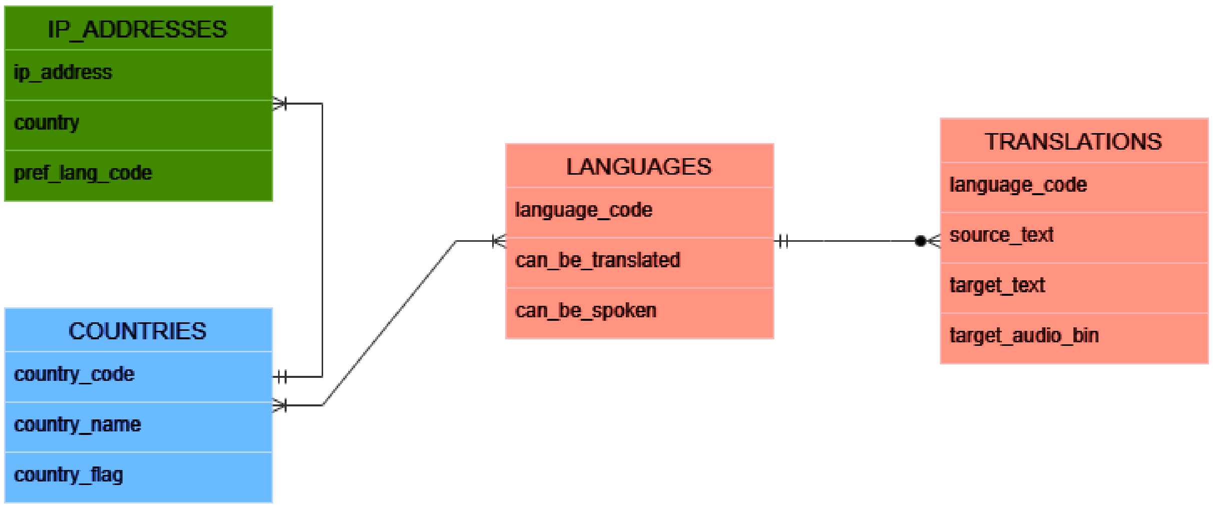
4.1 ERD Diagrams

4.1.1 Conceptual Design



The conceptual design of the app resembles the main functionalities: the app retrieves an ip address, localises it and then displays messages in the form of text and TTS.

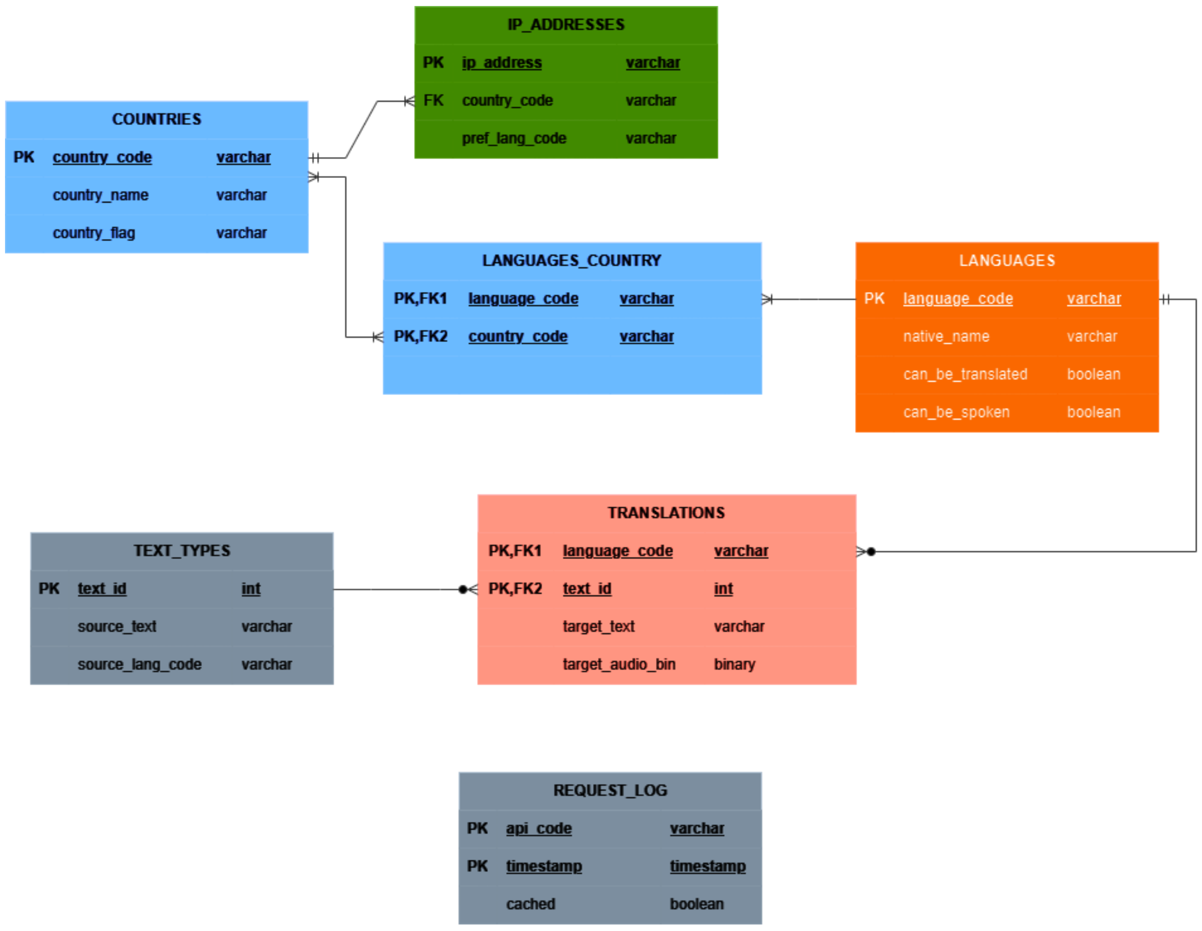
4.1.2 Logical Design



The logical design consists of four entities:

- Ip addresses are stored with the information about the country they belong to and the preferred language of the interface selected by the user.
- Countries stores the information about countries, such as the code, the name and the url of the flag
- Languages stores all the possible languages supported by the app, and the information about whether they can be translated or spoken by the TTS functions
- Translations is a caching table that stores translations of sentences and their audio generated by TTS

4.1.3 Physical Design



The physical design is composed by seven tables:

- IP_ADDRESSES stores informations about the ip address and the preferred language selected for that specific address
- COUNTRIES stores information about the countries that have been added after an ip geolocalization
- LANGUAGES stores the information about the language and the caching informations about whether it can be translated and/or TTS can be performed
- LANGUAGES_COUNTRY stores the list of languages of each country, to have the possibility for the user to change their preferred interface language
- TEXT_TYPES is a table that stores unique sentences acting as indexes for the caching mechanism
- TRANSLATIONS is a table storing translated text and audio generated by the TTS engine to act as the cache of the system
- REQUEST_LOG stores calls made by the application in order to see if the translation resulted in a miss or in a match in the cache

4.2 Tables Definition

4.2.1 IP_ADDRESSES

IP_ADDRESSES			
	Column Name	**Null**	**Data Type**
🔑	ip_address		varchar
🌐	country_code		varchar
	pref_lang_code	Y	varchar

4.2.2 COUNTRIES

COUNTRIES			
	Column Name	**Null**	**Data Type**
🔑	country_code		varchar
	country_name		varchar
	country_flag		varchar

4.2.3 LANGUAGES

LANGUAGES			
	Column Name	**Null**	**Data Type**
🔑	language_code		varchar
	native_name		varchar
	can_be_translated		bool
	can_be_spoken		bool

4.2.4 LANGUAGE_COUNTRY

LANGUAGE_COUNTRY			
	Column Name	**Null**	**Data Type**
🔑	language_code		varchar

🔑	country_code		varchar

4.2.5 TEXT_TYPES

TEXT_TYPES			
	Column Name	**Null**	**Data Type**
🔑	text_id		int
	source_text		varchar
	source_lang_code		varchar

4.2.6 TRANSLATIONS

TRANSLATIONS			
	Column Name	**Null**	**Data Type**
🔑	language_code		varchar
🔑	text_id		int
	target_text		varchar
	target_audio_bin	Y	binary

4.2.7 REQUEST_LOG

LANGUAGE_COUNTRY			
	Column Name	**Null**	**Data Type**
🔑	api_code		varchar
🔑	timestamp		timestamp
	cached		bool

5. Repository structure

Files included in this repository:

Folder/File	**Description**
/Documentation	
– Application Core Loop.pdf	Primary design document of the application as submitted
– ERD.drawio	Conceptual, logical and physical Entity Relation Diagrams
– table_definitions.sql	All physical tables in SQL
/gui	
– session.py	Handles the session variables for the benefit of Streamlit
/modules	
– connector.py	Handles connection with Snowflake, the cursor, and the execution of queries
– errors.py	Definitions of possible errors thrown throughout the application
– ipify_api.py	Connects through the API for retrieving a user's IP
– ipstack_api.py	Connects through the API for resolving a user's location data from their IP
– language_functions.py	Wraps the translate_api and tts_api in a code deciding to pull from cache or contact the API
– translate_api.py	Connects through the API for translating text, also connects to the endpoint to retrieve available languages
– tts_api.py	Connects through the API for speaking text, also connects to the endpoint to retrieve available languages
– weather_api.py	Connects through the API for weather data based on the user's location

/pages	
– 1_Manual_Ip.py	Streamlit interface page for changing the user's IP for testing purposes
– 2_Settings.py	Streamlit interface page for selecting a prefered language when resolved to a location with more than one
– 3_Admin.py	Streamlit interface page for managing the cached sentences and resetting all tables
– 4_Analytics.py	Streamlit interface page for viewing analytics on application metadata
/	
core_analytics.py	All functions for aggregating application metadata
core_info.py	All functions for concatenating the user's spoken informative greeting
core_init.py	All functions for initializing the user's location and language environment
core_reset.py	All functions for clearing the application tables for testing purposes
core_set_pref_language.py	All functions for changing the user's language environment
core_text_types.py	All functions for handling text types (cached sentences that can be translated)
Home.py	Main Streamlit interface page for the application