AKADEMIA GÓRNICZO-HUTNICZA

Audio Recorder
with
Google Drive Integration
„WaveReco”

Design Laboratory 2020/2021

Magdalena Nowosiad,

Mikołaj Telec,

Krzysztof Jędrejasz,

Sławomir Tenerowicz

1. Project description

WaveReco

Voice recording application integrated with Google Drive. Application will be designed on Kivy framework in Python.

IMPORTANT INFO

At the beginning, our aim was to create app for mobile devices. Unfortunately, due to technological issues, we had to limit our app only for desktop devices.

Team members:

- Krzysztof Jędrejasz - Scrum Master / Developer
- Mikołaj Telec - Product Owner / Developer
- Magdalena Nowosiad - Developer
- Sławomir Tenerowicz - Developer

Purpose:

The purpose of this project is to make uploading recorded audio to Google Drive easier and more comfortable, for those who use this facility a lot.

Success criteria:

- Writting an app which records audio data in uncompressed format (.wav),
- Accessing Google API -- ability of uploading data to user Google Drive,

Desired Result:

Desired outcome of this project would be an app with functionality of sending recorded audio in .wav format to user Google Drive and sharing it with other people.

Resources:

- Programming environment that enables us easy creation of apps,
- Easy way of testing our programm -- no need for outside testers,

Constraints:

- Little knowledge of programming mobile apps. Unfortunately, they came out during our work,
- Fully remote work.

Risks:

- Space management for .wav files (they are extremely spatious),
- Accessibility of Google API facilities,
- Problems with packaging finished app into one file.

Groups of target users:

- acoustic or DSP Engineers for recording sounds in high quality,
- people working remotely,
- seniors, who want to easily share recorded audio,
- students learning about digital signals,
- people who work at acoustic studios.

2. SWOT analysis

Strengths:

- Programming experience in using multiple languages (Python, C, C++),
- Flexibility and willingness to self-study in terms of the project,
- One of team members took part in Professional Scrum Course.

Weaknesses:

- Lack of experience in creating mobile applications,
- Lack of experience in cooperation with the use of SCRUM
- No budget for project development,
- Fully remote group work,

Opportunities:

- Due to the global pandemic, the recording application has a greater chance of appearing on the global market.

Threats:

- High competition on the mobile application market,
- Other student obligations (project deadlines, end-of-term examinations etc.) may cause difficulties in moving forward with the project,

3. Feasibility study (technical aspect)

This project aims to create a application that allows to record sound in uncompressed form and upload it to Google Drive with few simple clicks.

Potential solutions resulting from this project:

- great simplification for people who can't handle Google Drive uploads,
- time-saving solution for engineers,
- providing the tool that simplifies disc space management (uncompressed audio weighs a lot).

Criteria for evaluating above-metioned solutions:

- a number of new users who claim they cannot easily use Google Drive facilities,
- time difference in conduncting the same audio recording and uploading operations between two cases - with our application and without it.

The most feasible solution:

In this project we are going to focus on the group represented by scientist and try to deliver the most suitable solution for them. We have a guarantee that in this group there is a need for product we are going to create. Further use of the application by this target group will enable us to assess the level of our success.

Materials required:

- application will be used on desktop systems (Windows). Unfortunately, we didn't managed to create mobile version.

- programming environment that is both indented to program the mobile applications and enable use of Python langauge.

- Python language libraries which simplifies audio recording process, upload process and access to disc management.

- additional libraries that enables use of Google API faciliies.

Labor requirements:

This project is fully focued on software - there is a need only for software developers. Issuses like marketing and fundraising are not taken into account within this project.

4. Technology

a) Google Drive API

This **REST API** allows us to integrate our app with Google Drive cloud storage. Thanks to it every user will be able to upload audio file to his **My Drive**. API uses OAuth 2.0 authorization protocol which authenticates user before every data flow.

OAuth 2.0 authorization protocol:

To get access to the API, every app must be registered on **Google APIs** site. Then Google provides developer with **credentials.json,** which contains **Client ID** and a **client secret**. Those strings are unique for every application. Thanks to those, app user will **acquire access** and **refresh tokens**. For safety reasons credentials should be kept in safe place, away from unauthorized users.

When app needs access to user private data, first it asks Google for specific **scope of access** using credentials gained before. Scopes define app access level to the user cloud storage (set of allowable operations). After that, user is asked to authorize app action on his Google account. Then, Google grants application tokens mentioned earlier. Access token have short live time, but refresh tokens do not, so app will be able to acquire new access tokens, when previous one will expire.

After completing those steps, application can start data flow.

For more details, see API reference [link below] or check out our Tests scripts in the repository.

Modules:

Below is list of modules used for handling Google Drive API:

- **googleapiclien**t - core library for every Google API,
- **google.auth** - Google authentication library for Python,

- **google_auth_oauthlib** - library which contains oauthlib integration with google-auth. Makes authorization process simpler.

Tests:

For tests purpose two scripts were written:

a) **OAuth2_test.py**

Contains functions:

- **auth2_test(),**

This Python script tests OAuth2 authorization protocol. It includes generation, validation and refreshing of access tokens. Uses scope, which allows only read of file metadata. Script returns 10 last used files names and id numbers on Google Drive by user.

b) **Upload_test.py**

Contains functions:

- **authorization**(),
- **create_folder(),**
- **create_permission(),**
- **upload_file(),**

This Python script tests data upload to My Drive folder. For tests we have used 10 MB audio file in .wav format (nearly 1 minute long). Uploading took 13 seconds of execution time, which was nearly 16 seconds. Creating permissions for other users is also tested.

Reference documentation:

- **Introduction to Google Drive API** - https://developers.google.com/drive/api/v3/about-sdk
- **API reference** - https://developers.google.com/drive/api/v3/reference
- **Drive API instance methods** - https://googleapis.github.io/google-api-python-client/docs/dyn/drive_v3.html
- **googleapiclient core library docs** - https://googleapis.github.io/google-api-python-client/docs/epy/index.html
- **googleapiclient github site** - https://github.com/googleapis/google-api-python-client
- **google.auth** - https://google-auth.readthedocs.io/en/latest/index.html
- **google_auth_oauthlib** - https://google-auth-oauthlib.readthedocs.io/en/latest/#

b) Kivy framework

Kivy is a cross-platform Python framework for creating user interfaces. It will give us a possibility to create simple and nice looking UI.

File in .kv format is used for defining look of app. In this file:

- widgets/buttons and their layout on the screen is declared,
- graphic side of our app is defined.

Created .kv file is imported to the Python scripts. With created .kv file, Python classes defining role of user interface more precisely can be created.

Reference documentation: https://kivy.org/doc/stable/

5. User manual

Before recording you should choose your settings. In **Settings** menu you can modify:

- **Recording**:
  - Numbers of channels - 1 or 2,
  - Duration - audio recording time [s],
  - Sampling - 1kHz -> 384kHz,
  - Extension - .wav, .mp3 or .flac,
  - File name - recorded audio name,
  - A path of audio files - selecting audio to play.
- **Cloud**:
  - Access - specifies other users level of access to a file (reader, writer or None),
  - Folder name - name of folder in which file will be stored.
  - User email - email of user with whom you want to share. If blank space is left, file won't be shared with anyone.

Step by step guide:

1. Press **Record** to start saving audio from microphone.
2. After recoding ends you can **Play** captured audio. You can also **Delete recording** it if you don't like it.
3. Press **Send** to upload recorded audio to your Google Drive account. Before that check if you are logged to your Google account in your browser. If you are uploading something for the first time, authorization screen will show up. _App isn't verified by Google, so warnings will show up. You need to accept, to complete upload._
4. If all went well, your audio should show up in your Google Drive storage.
5. Press **Exit** to exit our app :).

Extra:

If you want to use another Google account, first you need to delete access token. You can do that by pressing **Del A.Token** button.

6. Software documentation

Source files:

- Wave_Reco_final.py:
  - main script with all classes,
- style.kv:
  - UI settings,
- settingsjson.py:
  - script used for managing app settings,
- wavereco_test_pc.json:
  - users credentials used for getting access to Google API,
- ai-eeg-data-processing.jpg:
  - app theme**,**

Used modules list:

- **kivy:**
  - necessary imports for creating UI,
- **googleapiclient:**
  - core library for every Google API,
- **google.auth:**
  - Google authentication library for Python
- **google_auth_oauthlib:**
  - library which contains oauthlib integration with google-auth. Makes authorization process simpler,
- **sounddevice:**
  - used for recording microphone audio**,**
- **scipy.io:**
  - used for saving audio in .wav format,
- **pickle:**
  - used for pickling/unpickling Google authorization tokens
- **os:**
  - used for pathname manipulations,

Class descriptions:

WaveReco(App)

Main class, which starts our app. Contains methods:

- `build()` - builds app UI,

- `build_config()` - sets app default settings,

- `build_settings()` - builds settings menu,

- `on_config_change()` - prints settings change in terminal,

- `display_settings()` - displays settings window,

- `close_settings()` - closes settings window,

MainWindow(Screen)

Class for handling buttons functionalities. Contains five methods:

- `record()` - records microphone audio,

- `play()` - plays last recorded audio,

- `delete()` - deletes last recorded audio,

- `exit()` - exits application,

- `send()` - starts authorization and uploads audio to user drive storage,

Upload()

Class for handling OAuth2 app authorization protocol and Google Drive upload. Contains two methods:

- **authorization(client_secret_file, api_name, api_version, scope)**
  - runs OAuth2 authorization protocol and creates Google Drive resource for interaction with API,
- **upload(folder_name, file_name, user_email, access):**
  - uploads recorded audio file to specified folder in user Google Drive and shares it with other user (if `user_email` and `access` provided),

7. Functional and non functional tests

Functional tests:

1. Checking output of **record** facility - record an audio, then play it using play facility or regular audio player.

2. Checking **delete** facility, whether the file is removed right after clicking **delete** button.

3. Checking whether the **play** facility works correctly.

4. Testing **sending** facility, process of authorisation and uploading audio data.

5. **Settings** menu testing, whether given data is correctly set.

Non-functional test:

Performance testing

1. To test our app performance we are going to conduct **Endurance testing**. We are plannning uploading "heavy" .wav for significant period of time. Simultaneously we will measure time of each upload. This test will show us how our app runs under large workload.

2. Testing **repeatability** - whether the app works appropriate no matter how many times was used (in a row).

8. Prototype testing outcome

Recording facility:

- The duration of recording was preset in code for 10 seconds.
- Pressing button **record** after above-mentioned 10 seconds results in correcly captured audio file.
- Capturing next audio file is possible either when **delete** facilty has been performed over the file or after re-runing application (the previous audio file will be overwritten).
- **Record** function is blocking.

Delete facility:

- After every recording session the deletion is possible, no matter how many times is performed.
- Pressing **Delete** button even if there's **no file** to delete has no unwanted impact on the application

Play facility:

- After every recording session playing audio file is **possbile**.
- When there's no audio to play, the **play** button changes its name to No file!. It remains so even if there's new audio to play, nevertheless works still correctly.
- Play facility is a blocking function.

Sending facility:

- Pressing send button begins google-drive authorisation process (only once). Completing all steps correctly allows to send audio file to given google drive otherwise upload is impossible.
- Captured audio saved on the local hard disc and the one sent to google drive is the same audio file.
- Using send facility when there's no file to send on the local disc crashes the application.
- Send facility is blocking.

Settings menu testing:

Clicking on settings bar opens new window with some options useful for recording process, i.e setting duration of recording process, setting sampling rate, choosing output audio format. Given values immediately update old ones.

Endurance testing

Uploading huge data files:

Reliability of sending facility highly depends on time needed to upload audio file. If needed time is longer than 2 minutes, application is not responding and then crashes.

Time difference between uploading audio using the application and on regular basis (via browser) is inconsiderable.

Testing repeatability:

Multiple use of recording - delete sequence, play faciliy or changing settings in one-run of application is possible. Send facility, if takes no logner than 2 minutes, can be used many times in a row without any fatal errors too.

9. Things to fix

What things we need to fix before Demo Day?

User interface:

- added security in the settings when entering extreme numbers.
- option to remove access token.
- modify option to delete recorded files by selecting / selecting all,

Google Drive upload:

- eliminate app crashing when authorization fails,
- prevent creating multiple folders in gdrive storage with the same name,

9.. Final version tests

Testing -- final version (.exe)

1. Recording facility

Preserved its primary functionality - no new bugs.

Added facilities:

- every use of record facility creates new audio file (no overwriting)
- name for captured audio can be set in **settings**

2. Delete facility

Preserved its primary functionality - no new bugs.

Changed functionality - file to delete must be chosen in **settings**

3. Play facility

Preserved its primary functionality - no new bugs.

Changed functionality - file to play must be chosen in **settings.**

4. Sending facility

Preserved its primary functionality - no new bugs.

Changed functionality - file to send must be chosen in **settings.**

Added widget to delete authorisation token - new cloud can be specified, works correctly.

5. Settings menu testing

Preserved its primary functionality - no new bugs.

Path file to save audio recordings is located automatically.

Endurance testing

1. Uploading huge data files

Upload that takes more than 2 minutes still is not successful but app does not crash.

2. Testing repeatability

No new bugs.

Fixed bugs from previous version

Added security in settings options (if inproriate values set returns to default ones).

Added option to remove authorisation token.

Modified delete option - enables to choose which file should be erased.

Modified recording facility - no overwritings, possibility to name a file.

Prevented creation of multiple folders on google drive (no new folder for each recording).

Unsuccessful upload does not crash the app (app hanging is terminated).