Multimedia Services and Applications (SAM)
2021/2022

Social Compressor Application

João Carlos Machado Rocha Pires (UP201806079)
Nuno Rodrigues de Castro Santos Silva (UP201404676)

Contents

1. Motivation, application context and description

With the never ending betterment of technologies, smartphones and cameras are capturing better image and videos, leading to an increase in size of these media files.

At the same time, instant message platforms like Discord and Facebook Messenger, and social media platforms like Facebook, Instagram, Twitter and LinkedIn are getting a growing influx of users every day.

An issue with never ending growth is the increasing quantity of media circulating through these platforms, made worse with the increasing size (in bytes) of these pictures and video. To curb the exponential increase of data, these platforms implemented different limitations of the size for uploaded pictures and videos.

To put things into perspective, the following lists present the size limits for pictures and video in the 6 current most used platforms at the moment.

Maximum picture file size:

- Facebook: 8MB
- Instagram: 8MB
- Twitter: 5MB (JPG, PNG), 15MB (GIF)
- LinkedIn: 5MB
- Discord: 8MB
- Messenger: 25MB

Maximum video file size:

- Facebook: 10GB
- Instagram: 100MB
- Twitter: 512MB
- LinkedIn: 5GB
- Discord: 8MB
- Messenger: 25MB

With this problem in mind, the current solution for the users that want to share media whose size (in bytes) surpasses the defined limits, is to upload those files to WeTransfer or any other Cloud sharing provider, and then share the link with the receiver, who also has to open the link and download the file (sometimes it has also to create an account, which increases the necessary steps, making this process even more time-consuming and intrusive).

To solve this issue, we came up with the idea for a desktop application, Social Compressor, that enables the users to compress any image or video keeping in mind the size limit of the platform where the media will be shared, having the compressed file fit within the boundaries accepted by said platform.

By uploading the media and selecting the desired platform, our application compresses the file in a lossy but perceptually lossless compression. Then, it allows the user to compare the original with the compressed media and to download the compressed version.

Note 1: The download is automatic. This will be explained further below in the User Manual chapter.

Note 2: The comparison screen with the original and compressed media is only available for pictures. In the case of videos, there's a different procedure, which will be explained in the User Manual chapter.

With our app, users lose the need to utilize 3rd party services for media compressing, and receivers can access the converted files straight away through the original social and messaging platforms, without the need to access outside services and therefore without the need to perform time-consuming registrations.

We've decided to not include User Authentication since the application is relatively simple and thus there's no need to keep track of the user's personal information. Also, the compression takes only a couple of seconds, which means that, once compressed, the user can download the media and there's no need to record it anywhere other than in the user's device.

2. Functional specification

Our application's functional specification divides in 4 parts: the first is related with the use cases and narrative, where we present the user stories and the flow of execution of an example use case; the second is related to the mock-up created prior to the start of the development phase; the third is related with the different application components and the way they interact; the fourth and final one is related with the allowed media types per platform.

2.1 Use Cases and narrative

For our application, considering the description in the first chapter, we've identified the following user stories:

- As a User, I want to upload a picture or a video to compress so that I can then send it via the desired platform.
- As a User, I want my picture or video to be compressed in a way that the final size (in bytes) is equal or less than the limit of the platform where I want to send the media so that I don't need to use other 3rd party services.
- As a User, I want to select the desired platform so that the application automatically identifies the media limit.
- As a User, I want to compare the original media with the compressed one so that I can see if there's any perceptually lossy compression.
- As a User, I want to download the compressed media so that I can then send it via the desired platform.

With the above use cases, we believe our application answers the problem raised and described in the first chapter.

2.2 User interface and user interactions

Before the start of the development phase, we've elaborated a mock-up of our application, which includes a draft version of the main screens and the interaction possibilities an user can have with them.

Since we already knew the technology that was going to be used, as well as its strengths and limitations, the mock-up is quite similar to what we obtained in the final version of the application (c.f. User Manual chapter).

The only major difference was that we didn't manage to have the Compressed Screen for the videos as we had planned. Instead of having both videos side by side, we ended up with a link to both, original and compressed. This problem was related to an error on the Video and VideoPlayer libraries from Kivy we couldn't correct.

There were also small differences and additions, improving the user's QoE (Quality of Experience):

- We introduced a 'Go Back' button in both in the Platform Selection and File Selection screens, to be able to return to the previous screen.
- We introduced a 'Return to Homepage' button in the Compressed Screen.
- Instead of the download button on the Compressed Screen, we have a link to the folder were the media was automatically downloaded to.

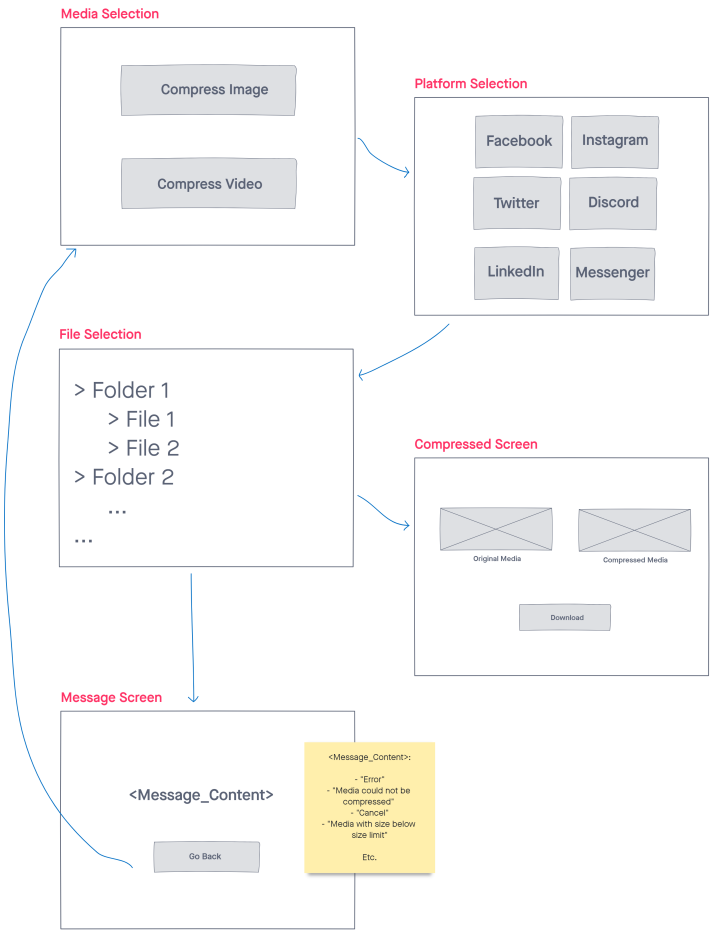

Figure 2.1: Mock-up.

In the mock-up above, we can see that the entry screen (the one on the top-right of the picture) is the Media Selection screen, where the user can select if the media to compress will be a picture or a video.

Then, on the Platform Selection screen, a list of platforms is displayed and the user must select the desired platform in order for the application to know which is the applicable size limit.

Then, a File Selection screen allows the user to select the desired file. Double clicking a file starts the compression, and the outcome will determine the next screen.

If the compression was successful, the Compressed Screen appears with both the Original and Compressed media as well as the option to download the compressed media.

Otherwise, the Message Screen displays the reason why the media was not compressed and a 'Go Back' button redirects the user to the starting Media Selection Screen.

Note 3: This mock-up was created using the InVision tool.

2.3 Application components and interactions

In this section, we present two diagrams: the first for the organization and interaction between Python files / groups of files; the second for the interaction between the screens.

Starting with the diagram for the organization and interaction between Python files / groups of files, we can see bellow that the Interface package contains the interface.py file, where the necessary logic for the screen events are handled. Some of those events use the functions defined in the functions.py file, under the Logic package. Still in the Interface package, the interface.kv file contains the structure / layout of the application screens.

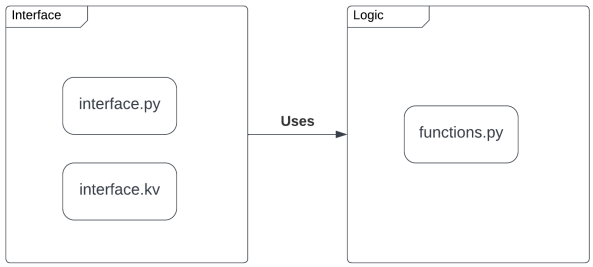

Figure 2.2: Python components.

The diagram for the screen interactions represents what has already been explained in the previous section (c.f. User interface and user interactions). The Platform Selection screen is displayed once the media was properly selected on the Media Selection screen. Once the User selects the platform on the Platform Selection screen, the File Selection screen appears. After successfully selecting the media to compress, the compression starts. On the compression completion, there are two possible transitions. If the compression finished with success, then the Compressed Screen is presented. Otherwise, the Message Screen displays the reason why the compression was not successful, allowing the User to start the process all over again from the Media Selection screen.

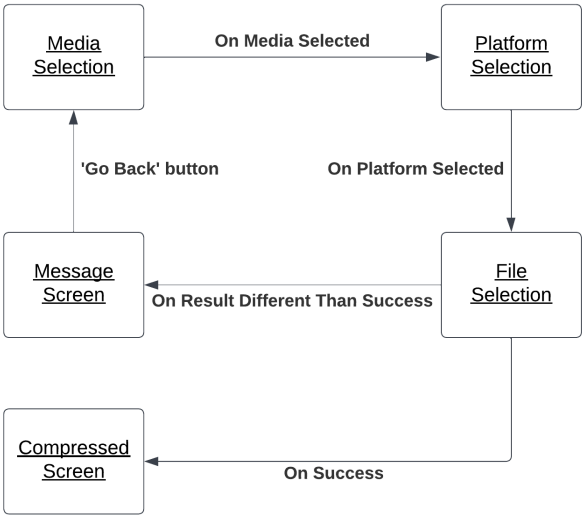

Figure 2.3: Screen interactions.

2.4 Allowed media types

For each platform, there's a list of allowed file extensions to be uploaded. Bellow, we present the list for the pictures and videos allowed extensions in each of the platform we support inside our application:

Allowed picture file extensions:

- Facebook: JPEG, BMP, PNG, GIF, TIFF
- Instagram: JPEG, PNG, BMP
- Twitter: JPEG, GIF, PNG
- LinkedIn: JPEG, PNG, GIF
- Discord: PNG, JPG, JPEG, GIF
- Messenger: PNG, GIF, JPG, JPEG

Note: At LinkedIn, GIF is not supported on Profile and Background pictures, but we do not deal with this problem since it's platform specific and is up to the user to decide where to use the compressed media.

Allowed video file extensions:

- Facebook: MP4 (recommended), almost all types supported
- Instagram: MOV, MP4
- Twitter: MP4, MOV
- LinkedIn: MP4 (recommended), ASF, FLV, MPEG-1, MPEG-4, MKV, WebM, H264/AVC, MP4, VP8, VP9, WMV2, WMV3
- Discord: MP4, MOV, webm
- Messenger: MP4 (recommended), almost all types supported.

With this in mind, our compression logic handles incompatible media types. For pictures, if the extension of the media to compress is not in the list of valid extensions for that platform, then it is by

default converted to JPG. For videos, our application handles both MP4 and MOV format, as these are supported by all platforms. Only LinkedIn does not support MOV formats. The loaded videos are converted to MP4, as this format is supported by all platforms.

3. Tech stack

To develop our application, we used the Python programming language and some useful libraries for different purposes:

- **kivy** - this cross-platform Python framework was used to build the interface of the application
- **Pillow** - this imaging library was used to perform all the tasks related to pictures (from compression to file management - save and open)
- **FFMPEG** - this video library was used to perform all the tasks related to videos (from compression to file management - save and open)

In order to make the code we've developed understandable, we've documented with comments the functions and the code wherever we found the need to do so. In other cases, by using understandable variables and functions names, we made our code straightforward to understand.

4. User Manual

After running 'python interface.py', as it's mentioned in the README file that goes with the code, a new window opens on the Media Selection screen. Here, the user can select one of two options: 'Compress Images' or 'Compress Video'.

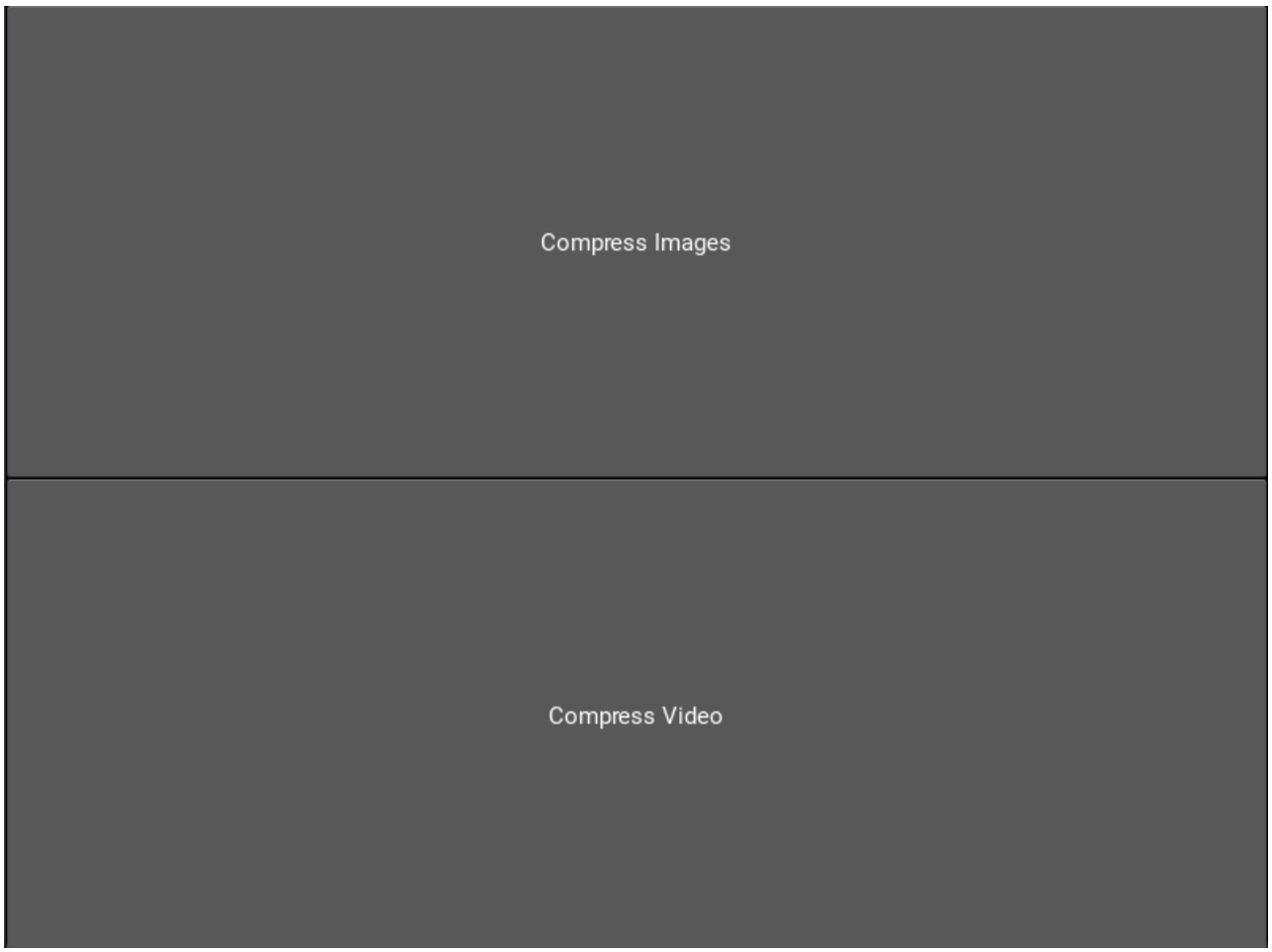

Figure 4.1: Media Selection screen.

Upon selecting one of the options, regardless of which, the user is then redirected to the Platform Selection screen where there are 6 different options, one for each supported platform: Facebook, Instagram, Twitter, LinkedIn, Discord and Messenger. There's also a 'Go Back' button that leads the user to the previous screen.

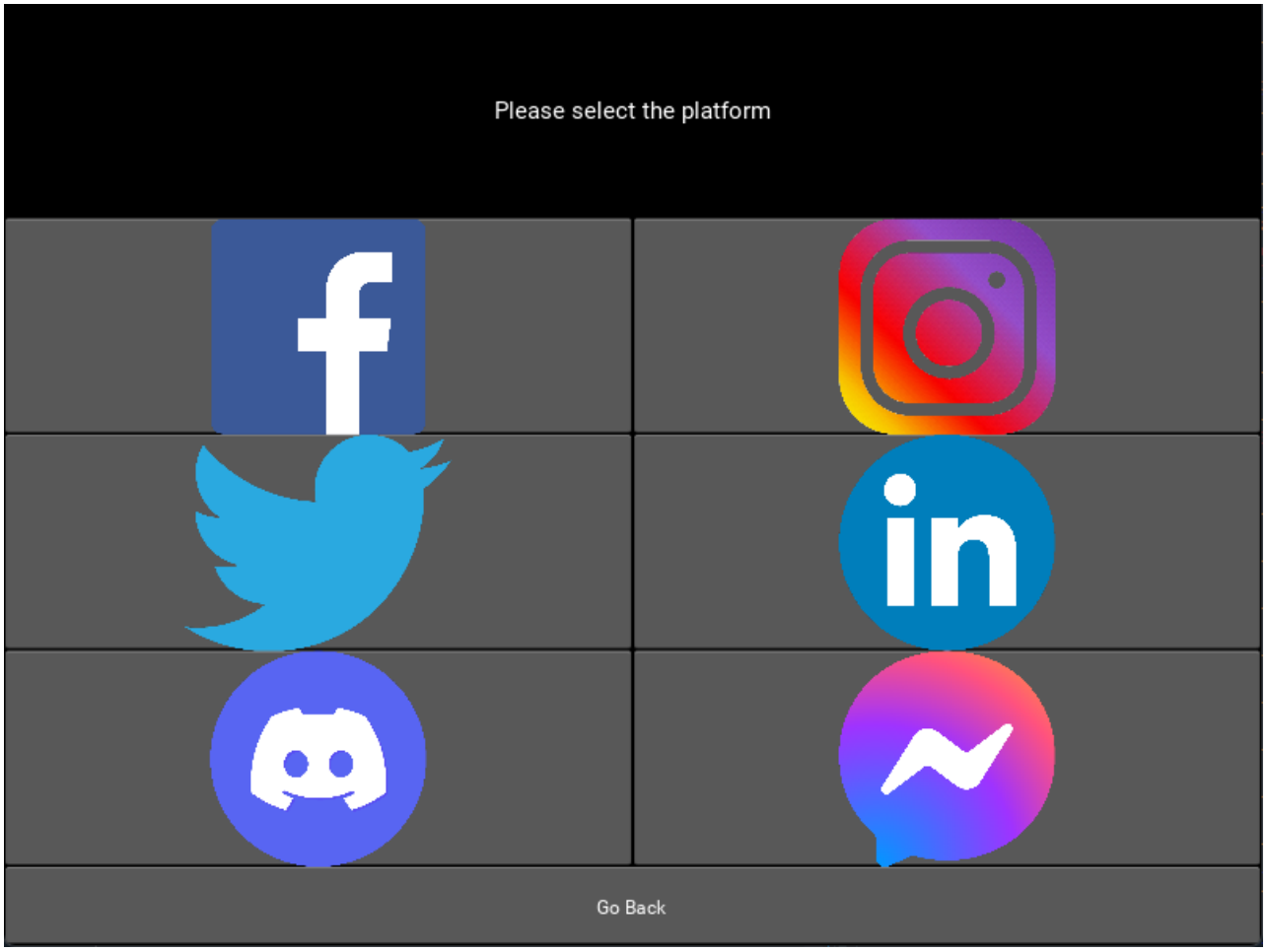

Figure 4.2: Platform Selection screen.

Upon selecting one of the options, regardless of which, the user is then redirected to the File Selection screen. Here, the computer directories are listed alongside the files within them. For each file, there's also, on the right, the indication of the size. In this screen, there's also a 'Go back' button that redirects the user to the previous screen upon clicking on it.

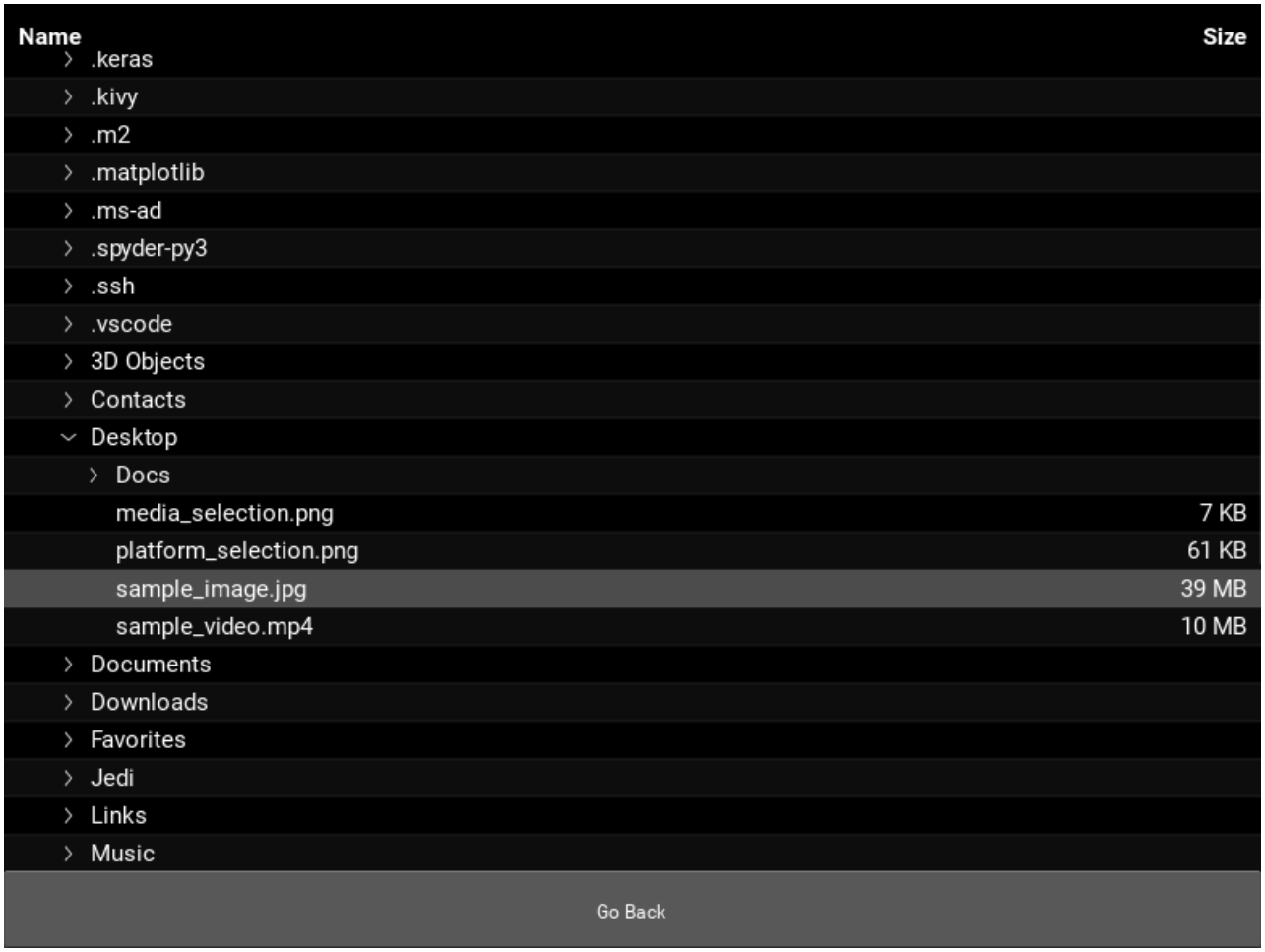

Figure 4.3: File Selection screen.

By double-clicking a file, there are three different possibilities. Whether a video or a picture was selected, if the size of the chosen file is already bellow the selected platform size limit, then the user is redirected to the Message Screen, where the message 'Media already with less than file size limit' appears. In this screen, there's the possibility to close the application by clicking on the 'Close Application' button or return to the homepage (i.e. Media Selection screen) by clicking on the button on the left.

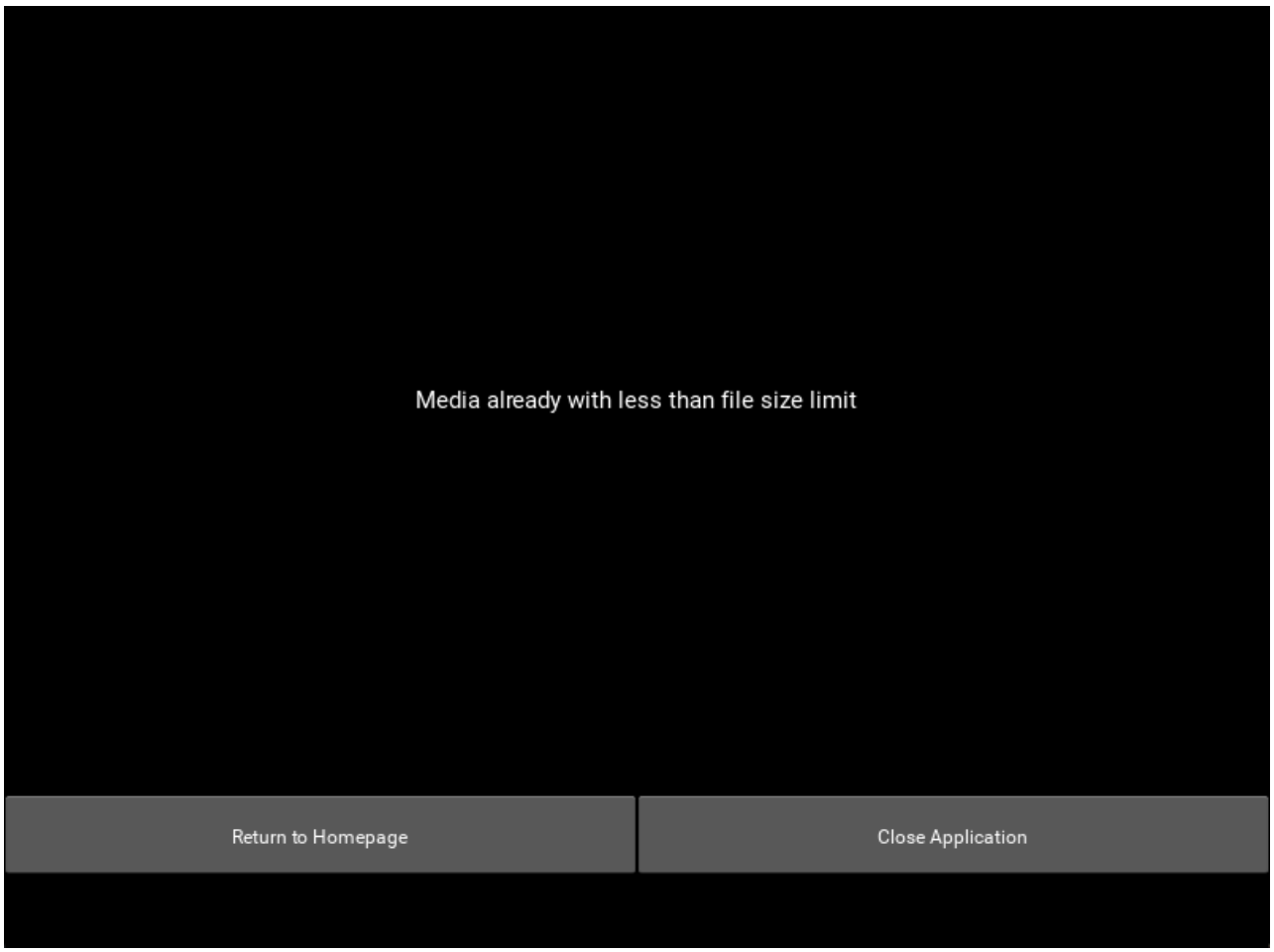

Figure 4.4: Message screen.

Now, depending on the media type, if the compression was successful, the user is redirected either to the Compressed screen (image) or to the Compressed screen (video).

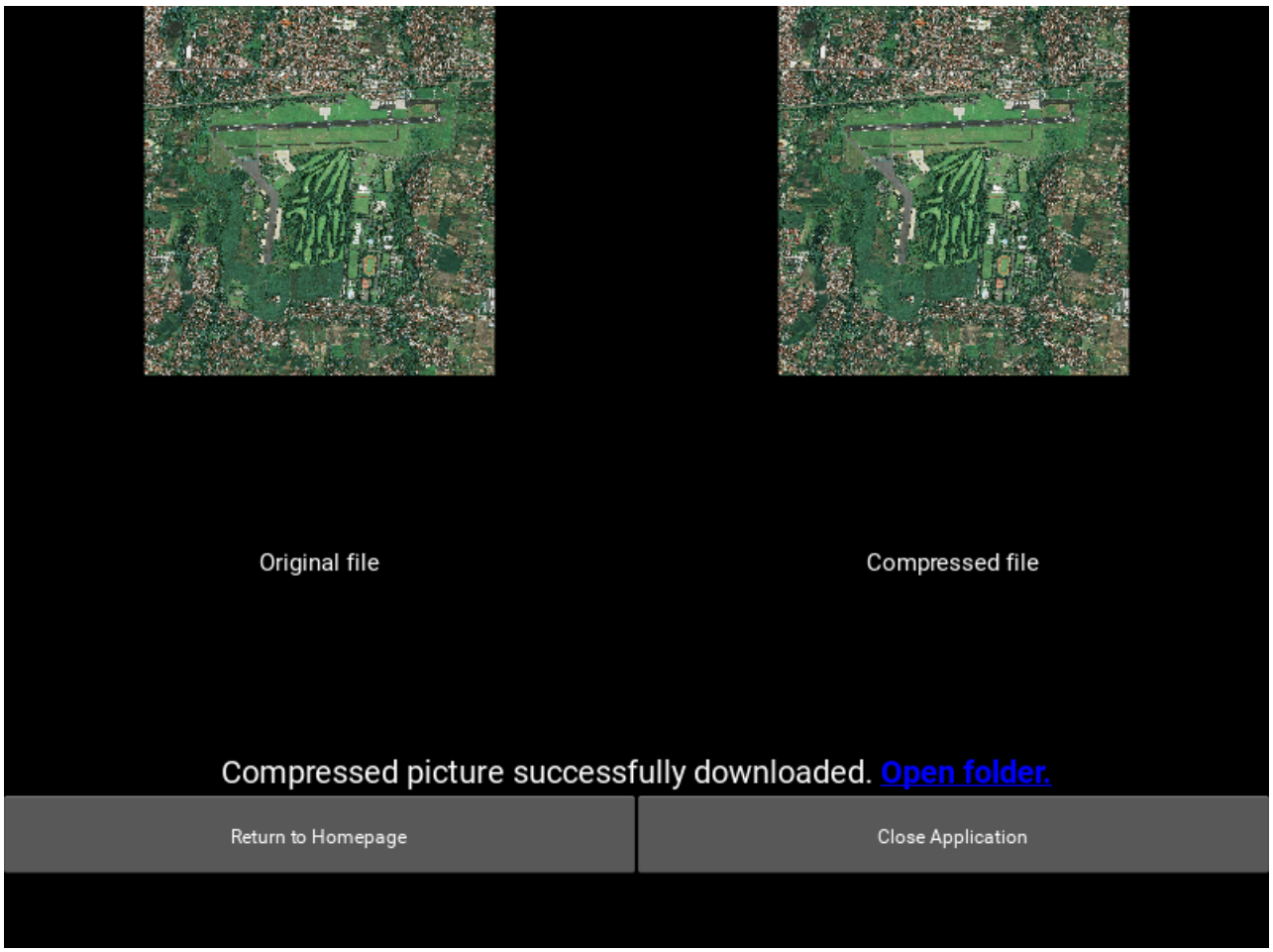

Figure 4.5: Compressed screen (image).

In the Compressed screen (image), on the left, the user can see a preview of the original file and, on the right, the preview of the compressed file.

On the bottom, the success message is followed by the link to 'Open folder', which opens the folder where the compressed image was automatically saved.

The two options on the bottom of the screen are the same displayed in the Message screen.

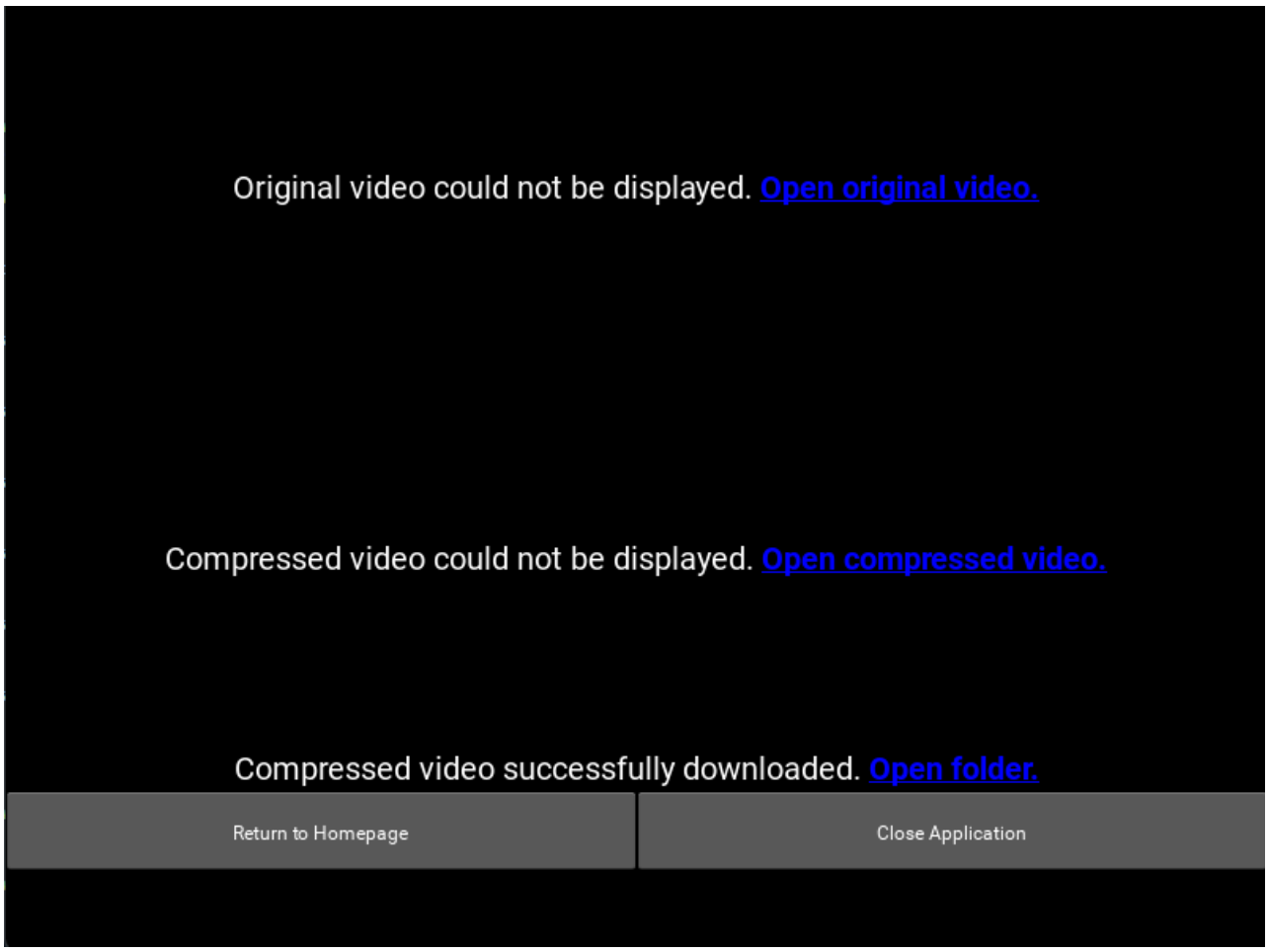

Figure 4.6: Compressed screen (video).

In the Compressed screen (video), there are two messages: one for the original video and one for the compressed video. Each one contains a link to open the corresponding video on the default media player of the computer the program is being run.

On the bottom, the success message is followed by the link to 'Open folder', which opens the folder where the compressed video was automatically saved.

The two buttons on the bottom are the same as the ones for the Message screen.

5. Lessons learned

To conclude this report, we are proud to say that we did our best to achieve the so called MVP for our idea.

Our application is far from being perfect, error-free or even the best in terms of performance. However, it makes possible to achieve the essence of the reason behind its construction: compress

videos and images to be within a size that is equal or less than the limit established by each platform.

Knowing the constraints that the authors of this project had, the current state of the project was the best we could achieve. There is, however, a list of improvements we identify as the next-steps:

- Include a loading screen or message upon the file selection to give the user a more close information as possible to what's happening.
- Display side-by-side the original and compressed videos in the Compressed screen (video).
- Create an executable to run this application on each computer without the need to run the 'python interface.py' command.
- Create a mobile version (Android and iOS) for this application.

During the development of this project, the major difficulties we found were in terms of technology. We had a lot of problems with Kivy and some other libraries we then discarded due to the number of errors we were getting. Most of the problems were solved by finding an alternative or checking on online forums for help and solutions that others implemented after facing the same kind of problems. Those errors are responsible for the lack of implementation of the features described in the next-steps.