Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN


# Mobile app with steganography functionalities

**Estudante:** Jorge Rodríguez Vázquez

**Dirección:** Óscar Fresnedo Arias

A Coruña, September de 2023.

*"Whether impossible or laughable, we continue to walk the path of men"*

*Kamina*

**Acknowledgements**

To my parents, your love and support have been my bedrock. You ignited my curiosity and fortified my determination. This accomplishment is yours as much as it is mine.
To my peers, friends and teachers, your influence on my journey has been indelible.
In deepest gratitude, thank you all. Here's to the chapters yet unwritten.

## Abstract

Steganography is the practice of hiding information within other data, such as images, audios, videos, etc. In this research, we consider applying this useful technique to create a mobile application that lets users conceal their own secret data inside other media formats, send that encoded data to other users, and even perform analysis to images that may have been under a steganography attack.

For image steganography, lossless compression formats employ Least Significant Bit (LSB) encoding within Red Green Blue (RGB) pixel values. Reciprocally, lossy compression formats, such as JPEG, utilize data concealment in the frequency domain by altering the quantized matrices of the files.

Video steganography follows two similar methods. In lossless video formats that permit compression, the LSB approach is applied to the RGB pixel values of individual frames. Meanwhile, in lossy High Efficient Video Coding (HEVC) formats, a displaced bit modification technique is used with the YUV components.



## Resumo

A esteganografía é a práctica de ocultar determinada información dentro doutros datos, como imaxes, audio, vídeos, etc. Neste proxecto pretendemos aplicar esta técnica como visión para crear unha aplicación móbil que permita aos usuarios ocultar os seus propios datos secretos dentro doutros formatos multimedia, enviar eses datos cifrados a outros usuarios e mesmo realizar análises de imaxes que puidesen ter sido comprometidas por un ataque esteganográfico.

Para a esteganografía de imaxes, os formatos con compresión sen perdas empregan a codificación Least Significant Bit (LSB) dentro dos valores Red Green Blue (RGB) dos seus píxeles. Por outra banda, os formatos de compresión con perdas, como JPEG, usan a ocultación de datos no dominio de frecuencia modificando as matrices cuantificadas dos ficheiros.

A esteganografía de vídeo segue dous métodos similares. En formatos de vídeo sen perdas, o método LSB aplícase aos valores RGB de píxeles individuais de cadros. En cambio, nos formatos High Efficient Video Coding (HEVC) con compresión *con perdas*, úsase unha técnica de cambio de bits nos compoñentes YUV.

**Keywords:**

- steganography
- encoding
- decoding
- jpeg
- rgb
- lossy
- lossless
- image
- video
- bits

**Palabras chave:**

- esteganografía
- codificación
- decodificación
- jpeg
- rgb
- con perdas
- sen perdas
- imaxe
- video
- bits

# Contents

# List of Figures

# List of Tables

# Chapter 1
# Introduction

Steganography, deriving its name from the Greek words *στεγανος* "steganos" meaning "covered" or "hidden" and *γραφος* "graphos", "writing" encompasses the investigation and implementation of methodologies designed to conceal messages or entities within designated hosts, referred to as carriers. The main objective of steganography is to render the presence of these concealed elements imperceptible to observers.

Although the term "steganography" is rather recent, dating back to the 16th century, the practice itself in its various forms, is an art that has been in use for hundreds of years.
Perhaps one of its earliest appearances in our history is the one referred to by Herodotus in "The Histories" [2]. This story narrates how the Athenian general *Histieo* asked *Aristagoras of Miletus* to rebel against the king of Persia. To do so, *Histieo* shaved the head of a slave, tattooed the message on his skull, and waited for the hair to grow back before sending it to its destination. There, after shaving it a second time, the message could be revealed.

In contemporary times, steganographic techniques have significantly advanced to include digital procedures, rendering analog steganography almost obsolete. The aim of this research is to develop a user-friendly mobile application that provides an explicit and comprehensive explanation of steganographic concepts while offering the ability to perform steganographic operations on a reasonably wide range of media.

## 1.1 Motivation

In the current landscape, the availability of open source steganographic tools remains limited. Moreover, the options for such tools that can be utilized by individuals without specialized training are even scarcer. The primary objective of this project is not to introduce pioneering or infallible steganographic techniques but rather to utilize existing ones in a controlled environment where the user has complete flexibility to customize the involved parameters without encountering excessive complexity. Thus, this work aims to offer an accessible approach to

the field of steganography, designed with default configurations to provide untrained users with an aesthetically pleasing interface and moderate customization options.

Within the **Steg'o** platform, users are empowered to seamlessly embed images or text into videos or images, subsequently transmitting them using the built-in chat functionality. Additionally, the system incorporates features such as message decoding and a basic image analysis system.

This journey originated as an initial concept, pursued as a personal project, yet it expanded far beyond its original scope. The subject matter is highly captivating, and although I acknowledge the current practical applications to be somewhat constrained, there is inherent value in acquiring knowledge irrespective of temporal considerations. Over the course of this project, I have expanded my initial idea to heights I would never have expected, greatly surpassing my early expectations. I express sincere gratitude for this unforeseen trajectory of growth and development.

## 1.2 Objectives

The main objectives of this project can be summarized as follows:

- **Understanding** the use of steganography in diverse environments and situations.
- **Analysis** of various methods and approaches to steganography by considering different file formats and media types.
- **Development** of a **front-end** mobile app that serves both as a chat application and steganography tool. The application should enable users to send encoded elements as well as encode their own data and customize the encoding process.
  Additionally, the application should provide functionalities for the analysis of potential encoded data.
  It should be aimed at both individuals knowledgeable in stenographic procedures and those without prior education in this domain.
- **Conception** and **refinement** of an **algorithm** designed to enact diverse steganographic procedures across a wide range of media types, aiming to achieve unnoticeable data concealment.
- **Implementation** of a **back-end** system that will enable the mobile app to process the users requests and return the embedded data through an Application Programming Interface (API).

## 1.3 Memory structure

This memory is organized as follows:

- Chapter 1 presents an introduction where the project theme, its motivation and objectives are exposed.
- Chapter 2 presents an overview of the fundamental components involved in a steganographic process. Subsequently, in Chapter 3, an exploration of current market options and the state of the art is conducted.
- The following chapter (4) elucidates the tools employed for the development, providing concise explanations for each tool's election and reasoning.
- Chapter 5 digs into the processes undertaken to establish an appropriate development methodology.
- Chapters 6 and 7 illustrate the project planning approach employed, based on sprints, along with an exhaustive analysis of requirements and use cases.
- Chapter 8 shows the design choices behind the architecture of **Steg'o**.
- Chapter 9 delineates the processes followed by the application in performing steganographic operations across diverse media formats.
- The next chapter (10) analyzes the steganographic processes to decide on optimum parameters.
- Lastly, Chapter 11 offers a comprehensive, step-by-step overview of the implemented use cases and the sprints conducted throughout the development process.

Chapter 2

# Steganography: algorithms & solutions

The present work delves into the realm of steganography, which offers an abundance of techniques for concealing information in numerous file formats. This chapter aims to present a brief overview of fundamental concepts that underlie any steganographic process.

## 2.1 Basic steganography overview

Figure 2.1 presents the block diagram for any fundamental steganographic procedure that can be divided into its several integrative components. This components are as follows:

- **Payload**: data that is covertly communicated.
- **Carrier**: data that conceals the **Payload** within itself.
- **Embedding process**: steganographic algorithm utilized to encode the **Payload** within the **Carrier**.
- **Stego data**: data that results from the **Embedding process**, which has the appearance of the **Carrier** data, but containing the **Payload**.
- **Communication channel**: medium through which the **Stego data** will be transmitted.
- **Extracting process**: steganographic algorithm used to decode the **Stego data**, which retrieves the **Payload** and the **Carrier** data.

Note that the nomenclature used herein is subject to modifications, and there are numerous synonyms of these fundamental components. However, for the purposes of this work, only the names previously specified will be employed.

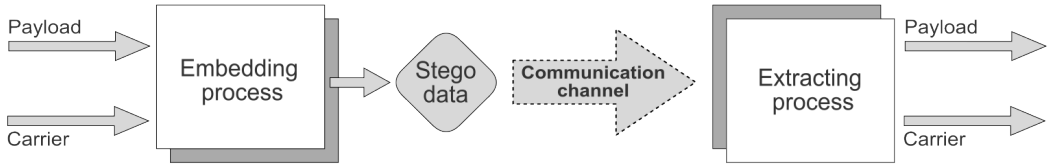



Figure 2.1: General steganography process functionality

According to *Eric Cole* in [3], in any steganographic methodology, three main security goals should be pursued: **privacy**, **robustness** and **undetectability**. These properties are interconnected and can be represented in a radar chart as Figure 2.2 shows:

- **Privacy:** refers to the capacity to maintain confidentiality such that unauthorized individuals are unable to access secret information.
- **Robustness:** refers to the ability of the system to maintain the integrity of any embedded secret information without causing damage or corruption during the transmission process.
- **Undetectability:** comprises two distinct requirements that are:
  - Inability to detect the presence of a hidden secret message within a medium by solely analyzing the characteristics of the medium.
  - Impossibility of verifying or negating the presence of a secret message within the **Carrier**.

A detailed explanation of how these concepts are implemented in **Steg'o** will be provided in Chapter 9.

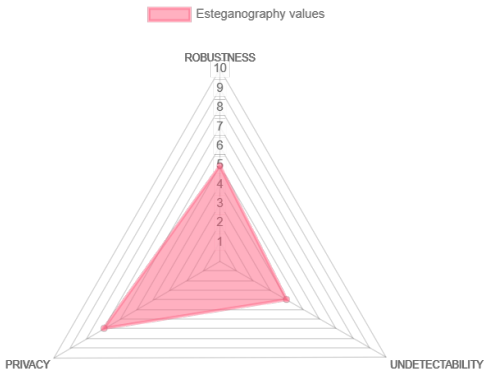



Figure 2.2: Radar chart of a hypothetical steganographic process

Chapter 3

# Exploring other market options

This chapter aims to examine some representative steganography application proposals currently available in the market. Specifically, the study will focus on evaluating the main alternatives found within the *Android* and *iOS* application stores. Each app will undergo scrutiny to determine if it offers all the necessary requirements considered for the application that will be developed in this project.

## 3.1 Main alternatives

### 3.1.1 PixelKnot

*PixelKnot* [4] is probably the main steganography app for **Android**. Due to its **easy-to-use** and **user-friendly UI**, it is by far the most popular app on this list. With over a **hundred thousand downloads**, this app can perform steganographic operations as hiding and extracting text information inside an image. It cannot perform video or audio esteganography. Data encryption is also avaliable for this app.

### 3.1.2 NoClue

Developed for **Android**, *NoClue* [5] is intended for concealing **textual messages** within **image** and **video** files. This application makes use of various commonly used image formats such as **JPG**, **PNG**, and **ICO**, as well as video formats including **MP4** and **AVI**, as carriers for embedding textual information. Aditionally, NoClue offers the functionality to incorporate **password protection** to safeguard the resulting carrier files from unauthorized access.

### 3.1.3 StegDroid

**Android** steganography tool designed to conceal **encrypted messages** within **audio** files. *StegDroid* [6] employs a technique known as **Echo Steganography** to embed concealed mes-

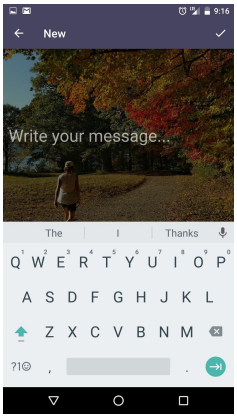
sages as an echo of the original audio.

### 3.1.4 Steganography Master

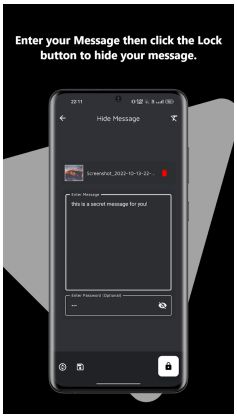
*Steganography Master* [7] is a free **Android** app designed for the concealment of **text information** within **images**. This application provides users with the capability to encode and decode hidden **text information** embedded within images. It also offers support for a large range of image formats (**PNG**, **JPG**, **ICO** or **BMP**, among others).

### 3.1.5 Pictograph

*Pictograph* [8] is an **iOS** steganography tool that allows you to hide **text information** in **images** by performing Least Significant Bit (LSB) spatial esteganography.

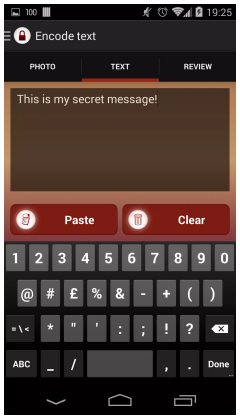



(a) Pixelknot

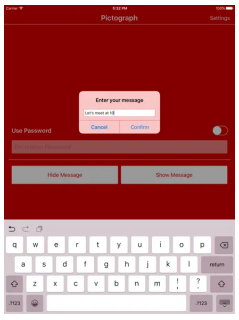



(b) NoClue


(c) StegDroid




(d) Steganography Master




(e) Pictograph

Figure 3.1: Examples of the graphical interfaces for the different steganography apps

## 3.2 Conclusion

Based on the aforementioned analysis, a comprehensive summary table has been formulated, showing the features and services offered by these applications. This analysis can be seen in Table 3.2. Additionally, an evaluation of the User Experience (UX) has been incorporated, measuring the applications' behavior and the quality of their interfaces.

| | **PixelKnot** | **NoClue** | **StegDroid** | **Steganography Master** | **Pictograph** | **Steg'o** |
|---|---|---|---|---|---|---|
| **Android** | YES | YES | YES | YES | NO | YES |
| **IOS** | NO | NO | NO | NO | YES | YES |
| **Image steganography** | YES | YES | NO | YES | YES | YES |
| **Video steganography** | NO | YES | NO | YES | YES | YES |
| **Audio steganography** | NO | NO | YES | YES | YES | PARTIAL |
| **Encryption** | YES | YES | YES | NO | YES | YES |
| **Sharing capabilities** | NO | NO | NO | NO | NO | YES |
| **Analysis capabilities** | NO | NO | NO | NO | NO | YES |
| **UX rating** | 6 | 7.5 | 6 | 6.5 | 6 | 8.5 |

Table 3.1: Measurement and comparison of steganographic application features

After conducting an exhaustive analysis of the currently available applications in the market, it is evident that a majority of them exhibit deficiencies in certain domains. Hence, it can be inferred that users could benefit from an application that integrates most of their functionalities into a singular entity, thereby simplifying the implementation of steganographic procedures.

In this sense, **Steg'o** emerges as an application that meets the needs of users while introducing novel functionalities alongside comprehensive explanations and guidance for less experienced individuals.

# Chapter 4
# Development tools

This chapter will detail the technologies and tools used for the development of **Steg'o**, both its *front-end* (client) and *backend* (server) sides.

## 4.1 Server side

For the creation of **Steg'o** *backend*, a combination of several technologies was used. The programming language chosen for both the API and the processing of data sent was *Python*. The database chosen to contain both the users' information and their messages and processed data was *Firebase Firestore*.

### 4.1.1 Python

**Python** [9] is a high-level interpreted programming language renowned for its emphasis on code readability. It finds extensive utilization in the development of diverse applications, including *Instagram*, *Netflix* or *Spotify*, among others. It is also widely used in web applications, software development, data science and Machine Learning (ML). *Python* exhibits a multi-paradigm nature, offering support for object-oriented and imperative programming, along with limited provisions for functional programming. Furthermore, *Python* is an interpreted, dynamic, and cross-platform language.

Languages such as *C*, *Rust* or *C++* were also taked into account to develop the *backend* of **Steg'o**, mainly because of speed and efficiency reasons; however, they were ultimately discarded. The decision to adopt *Python* as the prevailing programming language for the *backend* development primarily lies in its seamless integration with a wide range of data processing and image manipulation libraries.

*Python* also enjoys widespread popularity, resulting in a vast community of users who readily share extensive documentation and freely accessible code. Consequently, locating relevant resources or concrete implementation use cases is comparatively not as complex

compared to faster languages like *C* or *C++*. In addition, the use of libraries such as *numpy* provides a very significant speed increase while processing large amounts of data, which further tipped the balance towards the use of *Python*.

### 4.1.2 Flask

**Flask** [10] is a *Python*-based web application framework, developed by *Armin Ronacher* and his team, known as *Poocco*. It builds upon the Werkzeug WSGI toolkit and the *Jinja2* template engine, both of which are projects developed by *Poocco*.

The *Flask* framework stands out as a highly acclaimed and contemporary web application framework, widely recognized for its popularity in its field. It offers seamless adaptability, allowing for convenient expansion of its functionalities. Moreover, *Flask* is capable of accommodating intricate and sophisticated applications, rendering it a suitable choice for scaling up to meet the demands of complex projects.

#### Web application framework

A web application framework, or simply a web framework, is a comprehensive set of libraries and modules designed to facilitate web application development. It abstracts away low-level intricacies such as protocols and thread management, enabling developers to focus on building applications.

In contrast to **Django**, another widely popular web development framework for *Python*, Flask adheres closely to Pythonic principles. This makes it particularly great for small or capacity-constrained projects. In this sense, Flask was chosen over other potential web frameworks, such as *Django*, due to its inherent simplicity. For the requirements in the development of **Steg'o**, *Flask* proves to be more than sufficient, as it will solely be utilized for its API management capabilities.

### 4.1.3 Firebase

**Firesbase Cloud Firestore** [11] is a highly adaptable and scalable *NoSQL* database designed for server-side development, mobile devices, and web applications.

Developed by *Firebase* and *Google Cloud*, it provides efficient data synchronization between client applications through real-time listener objects. It also offers offline functionality for mobile devices and web applications, allowing the development of responsive applications that work seamlessly even in scenarios with limited network latency and intermittent internet connectivity. Furthermore, *Cloud Firestore* seamlessly integrates with other various Firebase and Google Cloud products, including *Cloud Functions*, thus facilitating the integration of other systems in the application like the push notification system employed in **Steg'o**.

The adoption of a cloud-based database solution was driven by the considerable magnitude of data that may need to be stored for each individual user.

### 4.1.4 Javascript

The utilization of **JavaScript** [12] in this project is limited, as its purpose is solely to facilitate the administration of user notifications via the Firebase push notification system. In this instance, the *JavaScript* environment employed operates within *Node*, an an independent runtime environment separated from a web browser.

## 4.2 Client side

In this section, the tools employed for the development of **Steg'o** frontend will be presented. The frontend codebase is built upon the **Flutter** framework in conjunction with the **Dart** programming language.

### 4.2.1 Flutter

**Flutter** [13] is a free open-source mobile User Interface (UI) framework, developed and released by *Google* in May 2017. It enables the creation of native mobile applications using a singular codebase, thereby removing the need of multiple codebases and languages to create an app for different environments. *Flutter* is based on widgets. In *Flutter*, everything is a widget. Widgets are basic building blocks that the developer can combine to build an entire application. They represent UI components like buttons, text, sliders, and so on.

*Dart*, a programming language introduced by *Google* in 2011, is an imperative requirement for *Flutter* development. Over the years, *Dart* has undergone significant advancements and refinements, making it an essential prerequisite. According to [14], Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a flexible execution runtime platform for app frameworks.

It employs robust type safety with static type checking for variable-value alignment, commonly known as **sound typing**. Type annotations are optional due to type inference, enhancing flexibility. Additionally, Dart's built-in sound null safety ensures non-null values by default and guards against null exceptions using static code analysis.

When *Flutter* was launched, it primarily supported mobile app development. Now, it supports app development on six different platforms: *iOS*, *Android*, *web*, *Windows*, *macOS* and *Linux*.

## 4.3 Development Tools

This section will present the tools employed in the planning and development of **Steg'o**across the entirety of its development cycle.

### 4.3.1 Android Studio

**Android Studio** [15] is the official Integrated Development Enviroment (IDE) used for Android application development. Based on the powerful *IntelliJ IDEA* code editor, it augments its features with additional functionalities that enhance productivity during the creation of Android apps. Some of these services are:

- A flexible build system based on *Gradle*.
- An efficient and comprehensive *Android* emulator.
- Seamless integration with *GitHub*.
- A wide collection of frameworks and testing tools.
- Real-time live edits facilitating immediate code updates on both emulators and physical devices.

As the primary choice for Android application development, and considering the development team's prior familiarity with this platform, picking Android Studio as the main IDE for the development of the app front-end was no challenge.

### 4.3.2 PyCharm

**PyCharm** [16], an IDE for *Python*, is a versatile platform developed by *JetBrains*. It is widely recognized and employed for the purpose of *Python* application development.

*PyCharm* emerges as an optimal choice for *backend Python* development due to its robust integrated development environment (IDE), which enhances coding efficiency. It offers advanced debugging capabilities, facilitates seamless integration with version control systems, fosters collaborative coding practices, streamlines project management, and ensures codebase consistency throughout the software development lifecycle.

*Pycharm* provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and support for *Django* web development. In the last years, *PyCharm* has garnered significant adoption by renowned organizations including *Twitter*, *Facebook*, *Amazon*, and *Pinterest*, who have chosen it as their preferred *Python* IDE.

### 4.3.3 Overleaf

**Overleaf** [17] is a web-based online editing tool for *LaTeX* documents. *LaTeX* is employed to generate scholarly and scientific texts of superior typographic precision within academic and research settings. *Overleaf* features an integrated compiler, ensuring near-instantaneous visualization of the document. It also provides a collaborative text editing platform that proves highly advantageous for concurrent drafting and revision of documents. In this project, *Overleaf* was employed for writing this report.

### 4.3.4 Balsamiq

**Balsamiq Wireframes** [18] is a tool designed for creating visual prototypes for different type of applications. With its intuitive interface and wide range of pre-designed objects to choose from, *Balsamiq* is one of the best choices for an easy and fast approach to prototype any kind of mobile or desktop app.

During the project's mockup stage and its early development, *Balsamiq* was utilized to generate basic designs, showing the essential information to be included in the different app's screens and its visual presentation.

### 4.3.5 Git

**Git** [19] is a distributed version control system that tracks changes in any set of computer files. It is usually employed for coordinating work among programmers collaboratively developing source code during software development.

In this project, *Git* was employed for managing both the *front-end* and *backend* source code.

### 4.3.6 GitHub

**GitHub** [20] is a commercial enterprise owned by *Microsoft* that provides a cloud-based service for hosting *Git* repositories. Primarily, it facilitates streamlined adoption of *Git* for an efficient version control and collaborative work among individuals and teams. *GitHub* stands as an exceptional choice for version control due to its seamless integration of collaborative features, robust repository management, and extensive developer community. These attributes collectively establish *GitHub* as the optimal solution for promoting a structured and collaborative environment throughout the software development cycle. During the development, *GitHub* was used to manage code versions and commit information from the project's *Git* repositories.

### 4.3.7 ffmpeg

The tool **ffmpeg** is a comprehensive, cross-platform software suite that allows users to record, convert, and stream audio and video. It includes the command-line tool ffmpeg itself, as well as associated libraries and utilities. The main components of the ffmpeg suite are:

- **ffmpeg (command-line tool):** Used for converting multimedia files between different formats. It supports a vast number of audio and video codecs, making it a go-to tool for many professionals and hobbyists alike.
- **ffprobe:** A tool for analyzing multimedia streams and extracting information about them.
- **ffplay:** A simple multimedia player based on the *ffmpeg* libraries.

In the scope of this project, the use of *ffmpeg* proved to be extremely useful as a pivotal instrument throughout the development and implementation phases for the diverse video steganography methodologies integrated in **Steg'o**.

In the final version of the app, *ffmpeg* assumes a main role performing tasks such as frame extraction, comprehensive frame analysis, format conversion, video import and export, among others. For this purpose, the python *subprocess* module was employed, enabling the seamless execution of Command Line Interface (CLI) commands within *Python*.

### 4.3.8 Jpegio

The **Jpegio** python package, authored by *dwgoon* [21], has been created to address the tasks of importing, exporting, and manipulating JPEG files. This package facilitates the seamless retrieval of quantized matrices inherent to JPEG files, enabling their modification without the need for recompression.

This feature is extremely useful in steganographic contexts, where automated and fast data insertion finds prominence. The efficacy of *Jpegio* is further accentuated by its integration with the *numpy* package, enhancing the speed and efficiency of this process.

# Chapter 5

# Methodology and development

A software development methodology, as stated in [22], is "a set of rules and guidelines that are used in the process of researching, planning, designing, developing, testing, setup and maintaining a software product". Hence, the adoption of a suitable work methodology is imperative to ensure the successful development process and timely fulfillment of the product requirements.

This chapter aims to explain the different available alternatives for software development, and the reasons for the choice of a specific methodology according to the project characteristics.

## 5.1 Development methodologies

Software development methodologies can be categorized into two primary groups: **heavy-weight** (or traditional) methodologies and **agile** methodologies [23].

Traditional methodologies follow a disciplined and rigorous approach in the development process. They are characterized by a comprehensive definition of all the requirements at the project's outset, resulting in limited flexibility for changes during the product development. In [24], the author criticizes the **bureaucratic nature** of these methodologies, making the development pace suffer as a consequence of their entanglement in their own rules. Moreover, the numerous phases of the software development in *heavy-weight* methodologies follow a sequential order, meaning that work on the next phase cannot commence until the previous one is finished.

Some widely utilized *heavy-weight* methodologies include:

- **Incremental**: process of software development where requirements are divided into multiple standalone modules of the software development cycle. In this model, each module undergoes a comprehensive lifecycle encompassing the stages of requirement

analysis, design, implementation, and testing. Subsequent releases of the software module introduce additional functionality, building upon the foundation established in the previous release. This iterative process continues until the entire system is successfully realized.

- **Waterfall**: sequential development process that progresses systematically through all project phases. Each phase is executed in a complete and self-contained manner before the subsequent phase commences, following a metaphorical waterfall-like flow.
- **Spiral**: development lifecycle approach employed to effectively manage risks. It integrates iterative development principles with key components of the **Waterfall model**. Widely utilized by software engineers, the spiral model is particularly advantageous for addressing the challenges associated with sizable, costly, and intricate projects.

On the other hand, *agile methodologies* facilitate the seamless adaptation of work practices to project circumstances at each phase of its development. In this sense, these approaches enable a high level of flexibility and immediacy in customizing the project and its development to the specific conditions of the environment. In this kind of methodologies, the development cycle is structured in increments, with the deployment of a functional version of the product at the end of each increment.

*Agile methodologies* offer several advantages, including cost reduction, enhanced product quality, early error detection, more stable version control or the usage of more relevant metrics. Two of the most commonly employed agile methodologies are **Scrum** and **Kanban**.

## 5.2 Scrum

**Scrum** is an iterative and incremental framework employed for product development and work management. It focuses on prescribing effective team dynamics to foster system adaptability within constantly changing environments. It follows a short number of management practices and principles to help deliver a quality product. The main *scrum* elements are:

- **Scrum roles:** people who participate in a Scrum project. Scrum defines three roles:
  - **Development team:** It only includes people who work to complete the tasks. Its size can vary from project to project, but typically, it stands around **3** to **9** people, which include roles such as developers, testers, designers, and sometimes others, depending on the nature of the product.
  - **Product Owner:** It is the voice of the stakeholders and is responsible for ensuring that the team delivers value to the business.

  - **Scrum Master:** It makes sure that the Development team works by following the values and practices of Scrum.

- **Scrum artifacts:** physical elements that help develop products according to the Scrum methodology. The most important are:

  - **Product Backlog:** refers to a dynamic list of features, requirements, enhancements and fixes that must be completed for the project to succeed.
  - **Sprint Backlog:** list of tasks that the *Development Team* must complete during a specific sprint. It must be a sample of the entire *Product Backlog* list.
  - **The increment:** is the current version of the product under development.

- **Scrum events:** meetings or temporal events that happen at a certain time and serve specific purposes during the development. The most important ones are:

  - **Sprints:** period during which the *Scrum* team works together to achieve an established product goal.
  - **Daily Scrum:** brief session in which the team members report back and plan for the day.
  - **Sprint Planning:** it is held at the begin of each *Sprint*. During this event, the team estimates the work to be completed during the next *Sprint*.
  - **Sprint Review:** it is held at the end of every *Sprint*. During this event, the team runs through work items they completed during the Sprint. The product owner discusses the current state of the backlog and the project's delivery dates, incorporating the state of the market, technology, and the latest feedback.

## 5.3 Kanban

The **Kanban** method presents itself as a robust framework for designing, managing, and enhancing flow systems tailored to knowledge work environments [25]. By visually representing the flow of work, imposing limitations on the Work in Progress (WIP), and emphasizing the completion of tasks before initiating new ones, organizations can effectively implement the *Kanban* method to foster continuous improvement. Some of the most important practices in *Kanban* include:

- **Limiting work in progress:** create limits in the volume of WIP content, and use these limits to determine when to initiate new tasks. This approach can optimize the flow of work, reduce lead times, enhance quality, and achieve more frequent deliveries.
- **Explicit policy formulation:** clearly defined policies help elucidate the development process rather than using a mere listing of different stages in the workflow. These

policies should be concise, straightforward, well-defined, visible, consistently enforced, and easily modifiable by the individuals involved in the service.

- **Implementation of feedback loops:** feedback loops constitute an indispensable component of any system that seeks to facilitate evolutionary change within variable environments.

## 5.4 Scrum vs Kanban: Differences

According to *Attlassian* in [26], "Kanban is a project management framework that relies on visual tasks to manage workflows, while Scrum is a project management framework that helps teams structure and manage their work through a set of values, principles, and practices."

*Kanban* methodology revolves around the **visual representation** of tasks, imposing limits on work in progress and optimizing efficiency. *Kanban* teams prioritize **minimizing the duration of projects**, aiming to streamline the entire process from initiation to completion. It is particularly beneficial for teams facing numerous incoming requests of varying priority and size. In contrast to *Scrum*, which demands a **high level of control over the project scope**, *Kanban* allows for a more adaptable workflow.

On the other hand, *Scrum* teams commit to delivering a functional increment of work within **designated time intervals**. Its objective is to establish iterative feedback loops that enable acquisition and integration of customer insights. This approach allows obtaining an expected and high-quality version of the final product.

## 5.5 Methodology decision

For the sake of flexibility with the project requirements, heavy-weight methodologies are not the best choice, so in this case, it's up to agile methodologies.

Among *Scrum* and *Kanban*, and considering the project's scope and the need for comprehensive documentation and iterative evolution, **Scrum** emerges as the most suitable methodology.

The adoption of **Scrum** in this project is justified due to its ability to foster transparency, focus, efficient time management, adaptability, continuous improvement, enhanced communication, and risk mitigation. **Scrum** facilitates continuous **improvement through regular feedback loops**, allowing the developer to refine their skills and enhance productivity over time, which can be very important in a project with a single developer.

## 5.6 Application of Scrum to the project

To effectively implement the Scrum methodology in the project, certain adaptations were introduced to accommodate it to the specific circumstances and characteristics of this project. These modifications take into consideration the fact that the Scrum team comprises only two individuals: a **tutor** and a **student**.

Regarding the distribution of roles within the Scrum framework, they were allocated between the two team members as follows:

- **Development team:** this role was taken on by the student.
- **Product owner:** this role was assumed by the student, as he is responsible for the project idea and managing the system requirements through the product backlog.
- **Scrum Master:** this role was played by the project tutor as the responsible person for the supervision tasks and the proper application of the methodology's principles.

The project lifecycle follows the principles outlined by **Scrum**, which involves dividing it into multiple Sprints. In particular, the duration of each Sprint will be variable, ranging from **10 to 15 days**, on average.

The inclusion of daily meetings in the project lifecycle is omitted due to factors as the team size and project nature, which make this characteristic unnecessary.

**Sprint Reviews** are held up at the end of every Sprint, in which the Development team shares its progress with the Scrum Master, and the functionalities to be implemented in the next increment are programmed. Hence, the Sprint Review and the Sprint Planning have been combined into a single meeting at the end of each Sprint for convenience.

During the project's inception, the requisites were defined to compose an initial depiction of the **Product Backlog**. Subsequently, this backlog underwent iterative enhancements through a process of dynamic refinement.

Chapter 6

# Strategic Planning and Requirements Analysis

## 6.1 Planning overview

The project planning process involves a methodical organization of various tasks and allocation of necessary resources to accomplish them. However, future planning can pose challenges and lead to frustration when the scope of the project is not fully defined. Scrum provides a solution for this problem as it introduces the term of Sprints and Sprint Plannings. As described previously, the Scrum team collaboratively determines the incremental Sprint and selects items from the Product Backlog to address in the new Sprint.

Each sprint was scheduled to have a duration of **10 to 15 work days** approximately, corresponding to an estimated completion time of **50-75 hours** (calculated as 5 hours per day, and assuming a standard 5-day work week). Over the course of the project's progression, a total of **12 sprints** were executed. As a result, the initial projected duration for the project was 750 hours, yet the actual duration extended close to **825 hours**. This lengthening of the estimated timeframe can be attributed to several factors, such as the creation of functional prototypes and the adaptation to the employed technologies.

The following are the tasks scheduled to be performed in each sprint of the project.

### 6.1.1 Sprint 1 (02/01/2023 - 20/01/2023)

Tasks performed:

- Exploration of existing techniques for image steganography.
- Evaluation of diverse implementation approaches, weighing down associated difficulties and prospects.

- Setup and construction of the project within the designated IDEs, namely *PyCharm* for Python development and *Android Studio* for Android development.
- Inspection of prerequisites concerning communication between the frontend and back-end components.

**Estimated duration:** 75 hours.
**Real duration:** 75 hours.
**Notes:** The project adhered to the estimated schedule without any deviations. While there were initial challenges encountered during the project's creation within Android Studio, these issues were swiftly resolved within a short time.

### 6.1.2 Sprint 2 (23/01/2023 - 10/02/2023)

Tasks performed:

- Usage of the mock-up creation framework *Balsamiq Wireframes* to perform the design and implementation of a prototype for the Android application ***Steg'o***, which would help the team to better visualize the development direction.
- Creation and configuration of a *Firebase* database, following the construction of a well-structured diagram to ensure its proper implementation.
- Beginning of the development process for a image steganographic algorithm in the spatial domain, resulting in the successful implementation of a basic operational framework by the end of the Sprint.

**Estimated duration:** 75 hours.
**Real duration:** 75 hours.

### 6.1.3 Sprint 3 (13/02/2023 - 24/02/2023)

Tasks performed:

- Investigation into the operations required for the implementation of frequency domain image steganographic algorithms.
- Beginning of preliminary development for a basic JPEG encoder/decoder in *Python*, which would facilitate manipulation of the image quantization procedure for data embedding.
- Beginning of *Android* application development, entailing the design and implementation of user registration and login interfaces.

**Estimated duration:** 50 hours.
**Real duration:** 50 hours.
**Notes:** Challenges arose in making the JPEG encoding prototype functional, needing a thorough debugging process to achieve resolution.

### 6.1.4 Sprint 4 (27/02/2023 - 17/03/2023)

Tasks performed:

- Continuation of the development of frequency domain steganographic algorithms.
- Creation of the *Flask* API that will allow communication between *frontend* and *backend*.
- Conception of the application's chat screen, where the user can add others and start a conversation.

**Estimated duration:** 50 hours.
**Real duration:** 75 hours.
**Notes:** Complications were encountered during the optimization of the frequency steganography algorithm. Through a process of trial and error, enhanced efficiency was achieved through the utilization of numpy-optimized loops.

### 6.1.5 Sprint 5 (20/03/2023 - 07/04/2023)

Tasks performed:

- Conclusion of JPEG encoder/decoder development.
- Testing of the JPEG encoder/decoder.
- Development of the Operation Selection Screen and Image Analysis Screen in the *frontEnd*.
- Enhanced database user management in Firebase through document relationships to optimize efficiency.

**Estimated duration:** 75 hours.
**Real duration**: 75 hours.

### 6.1.6 Sprint 6 (10/04/2023 - 21/04/2023)

Tasks performed:

- Creation of Steganographic Options Screen for **Steg'o** Android application.
- Improvement of Image Analysis Screen for **Steg'o** Android application.

- Research of existing video steganographic procedures.

**Estimated duration:** 50 hours.
**Real duration:** 50 hours.

### 6.1.7 Sprint 7 (24/04/2023 - 05/05/2023)

Tasks performed:

- Replacement of Steganographic Options Screen for image Encoding Decoding Screen for **Steg'o** Android application.
- Creation of Loading Screen for **Steg'o** Android application.
- Added customization options to Image Analsysis Screen for **Steg'o** Android application.
- Basic implementation of spatial domain video steganography.

**Estimated duration:** 50 hours.
**Real duration:** 50 hours.
**Notes:** Issues emerged when attempting to retrieve embedded data in video steganography, and a viable solution was not attained within the sprint timeframe.

### 6.1.8 Sprint 8 (08/05/2023 - 26/05/2023)

Tasks performed:

- Development of notification system based on Firebase push system for **Steg'o** Android application.
- Significant performance enhancement has been achieved in the already implemented steganographic algorithms through the utilization of vector operations facilitated by the *numpy* library.
- A comprehensive revision of the JPEG encoding/decoding algorithm has been undertaken to optimize the speed of JPEG processing, employing the *jpegio* module. Consequently, it is no longer mandatory to re-encode the image in order to modify the quantized values.

**Estimated duration:** 75 hours.
**Real duration:** 75 hours.
**Notes:** Difficulties with the refactoring of JPEG quantized matrices, particularly in relation to block division. However, these issues were successfully addressed in time.

### 6.1.9 Sprint 9 (29/05/2023 - 09/06/2023)

Tasks performed:

- The *ffmpeg* framework was used to implement the algorithm for steganography in High Efficient Video Coding (HEVC) videos.
- Extensive experimentation was conducted to evaluate the impact of encoding variables on the level of data loss during the video compression procedure.
- Noteworthy enhancements were made to the visual design of the Android application, encompassing significant overhauls to the chat interface as well as the encoding and decoding screens.
- Numerical data was incorporated into the analysis screen to augment image analysis capabilities.

**Estimated duration:** 50 hours.
**Real duration:** 50 hours.
**Notes:** Several doubts surfaced during experiments with HEVC video encoding, especially inquiries into why yuv420p exhibited less data loss than yuv444p during the encoding process.

### 6.1.10 Sprint 10 (12/06/2023 - 30/06/2023)

Tasks performed:

- Developed a basic audio steganography system for embedding seeds in videos.
- Enhanced the mobile application to provide comprehensive configuration options for video steganography.

**Estimated duration:** 75 hours.
**Real duration:** 75 hours.

### 6.1.11 Sprint 11 (03/07/2023 - 21/07/2023)

Tasks performed:

- Refactoring of the *backend* source code, grouping operations in common classes and testing API performance for greater maintainability and future codebase expansions.
- Beginning of code refactoring in *frontend*, where the performance of database calls was improved, and a state management solution was implemented using the **Provider package** for *Flutter*.

**Estimated duration:** 50 hours.
**Real duration:** 75 hours.
**Notes:** The code refactoring process extended beyond initial expectations due to the previous code's inadequate organization, making logical partitioning a challenging task.

### 6.1.12 Sprint 12 (24/07/2023 - 18/08/2023)

Tasks performed:

- Finished the refactoring of the *frontend* code. Improved the appearance of the user menu, home screen, data encoding/decoding screens.
- Added history functionality, which allows the user to view the last encodings or decodings performed, access their details and download the data.
- Completed the documentation of the memory.

**Estimated duration:** 75 hours.
**Real duration:** 100 hours.
**Notes:** The memory took much longer than estimated due to constant changes in order not to exceed page limit and to try not to remove relevant data for the reader.

## 6.2 Project Costs

This section explains the estimation of the project cost, which is derived from resources allocation and committed times. The costs can be categorized into three distinct areas: human costs, technical costs and indirect costs.

### 6.2.1 Human Costs

| | **Junior Programmer** | **Project Manager** | **Scrum Master** | **Total** |
|---|---|---|---|---|
| **Hours worked** | 825 | 15 | 12 | 852 |
| **Salary (€/hour)** | 11 | 21 | 12 | 44 |
| **Total** | 9.075 | 315 | 144 | **9.534** |

Table 6.1: Salary calculation for each of the roles involved in the project

In order to provide an accurate estimation for this kind of costs, we make the assumption that the project was carried out by a **junior programmer**, one **project manager** and a **scrum master**. The corresponding salaries (euros (€) per hour) and time (hours) devoted to the project of each individual are outlined in Table 6.2.1. The hourly fees for the project manager, scrum master and junior fullstack developer were extracted from [27], [28] and [29], respectively, assuming a 40 hour work week.

### 6.2.2 Technical Costs

This section will analyze the prices of the tools used during the development of the project that had an expense and influenced its total cost. As seen in table 6.2.2, the total cost of the technical resources is 195.93€, which results from the sum of 7 months of **PyCharm Professional** license and **Firebase Spark Plan**.

| Technical resource | Monthly cost | Total |
|---|---|---|
| **PyCharm Professional** | 24,90€ | 175€ |
| **Firebase Spark Plan** | 2,99€ | 20,93€ |
| **Android Studio** | Free | Free |
| **Total cost** | | **195,93** |

Table 6.2: Technical cost analysis

### 6.2.3 Indirect costs

| | Personal Laptop | Desktop Computer | Total |
|---|---|---|---|
| **Used months** | 7 | 1 | 8 |
| **Price (€)** | 1.400 | 1.500 | 2.900 |
| **Lifespan (months)** | 72 | 60 | 132 |
| **Total cost (€)** | 136,11 | 25 | **161,11** |

Table 6.3: Indirect cost analysis

Taking into account the indirect expenses, they primarily pertain to the cost of the utilized computing devices, including a **personal laptop** (A) exclusively allocated for code development, and a **desktop computer** (B) used primarily for conducting tests. Assuming a standard lifespan of six years for system A, and five years for system B, their operational cost is calculated by dividing the purchase price by the lifespan in months and then multiplying it by the duration of usage. These three measures are presented in terms of price and cost denominated in euros (€), while time is measured in months (refer to Table 6.2.3).

### 6.2.4 Total costs

Based on the previous data, and considering the information provided in Table 6.2.4, the sum of the three kinds of costs, outputs at **9.891,04€**.

| Expense | Total (€) |
|---|---|
| **Human costs** | 9.534 |
| **Technical costs** | 195,93 |
| **Indirect costs** | 161,11 |
| **Total cost (€)** | **9.891,04** |

Table 6.4: Total cost analysis

### 6.2.5 Project summary

Table 6.2.5 shows an abstract of the project, where the most important metrics are brought into play. The project concluded in a satisfactorily, with various noteworthy achievements. While there remain potential areas for enhancement, the overall conclusion is greatly positive.

The envisioned objectives have been successfully achieved, culminating in the development of a steganographic application designed to serve a diverse spectrum of users. **Steg'o** not only provides guidance for novice users but also empowers seasoned enthusiasts with a high degree of flexibility.

| | |
|---|---|
| **Duration** | 228 days (02/01/2023 - 18/08/2023) |
| **Development time** | 825 hours |
| **Total costs** | 9.891,04 € |

Table 6.5: **Steg'o** project summary

# Chapter 7
# Analysis

The analysis phase, an integral part of the software life cycle, encompasses the examination of essential features and desired functionalities to be incorporated into the product under development. Its primary objective is to exhaustively comprehend the system requirements.

## 7.1 System actors identification

In the process of addressing system requirements, it is crucial to identify potential users of the system and their necessities, as they will interact with the application in distinct ways depending on them.
An actor refers to any external entity that interacts with the system and requires specific functionalities from it. This encompasses not only human operators but also external systems, as well as abstract entities like time.
In this particular case, there are only four types of actors, the **non-registered user**, the **registered user** the **Firebase Database** and the **API system**.

## 7.2 Requirements analysis

Requirements analysis is a pivotal software engineering labour that serves as a critical link connecting the system-level definition of software and its subsequent design.
This analytical process allows system engineers to precisely delineate the operational attributes of the software, enclosing its functionality, data management, and performance considerations.
Moreover, it eases the determination of the software's interface with other system components, while concurrently establishing essential constraints that must be taken into account by the developers.

### 7.2.1 Functional requirements

A functional requirement in software development defines a specific behavior that a software system or component must be able to follow in order to perform correctly. It describes what the system should do, in response to specific inputs or conditions.
In this case, the functional requirements of **Steg'o** will be presented by grouping them into the various functionalities of the app.

- User management:
  - Users can be registered in the system.
  - Registered users can authenticate themselves.
  - Registered users can view their profile.
  - Registered users can edit their profile.
  - Registered users must be able to log out.
- Chat functionality procedures:
  - Registered users can send chat requests to other users.
  - Registered users can accept or decline requests from other users.
  - Registered users can view their chats into the chat screen.
  - Registered users can send text, images or videos to other users.
  - Registered users can copy messages from themselves or other users in their chats.
  - Registered users can delete messages from themselves or other users in their chats.
- Steganographic procedures:
  - Registered users can embed images into other images.
  - Registered users can embed text into images.
  - Registered users can embed images into videos.
  - Registered users can embed text into videos.
  - Registered users can decode embedded data from videos.
  - Registered users can decode embedded data from images.
  - Registered users can configure the embedding procedure.
  - Registered users can see their encoding history.
  - Registered users can see their decoding history.
- Image analysis procedures:
  - Registered users can compare the pixel values of two images of the same size.
  - Registered users can view the differences between the images in a heatmap.
  - Registered users can perform changes in the heatmap, like changing its opacity, colours,...
  - Registered users can view a numerical analysis of the comparison between the images.

## 7.3 Use cases

This section will detail the use cases of **Steg'o** according to the identified functional requirements.

Table 7.1: Register user

| **UC-01** | **Register user** |
|---|---|
| **Description** | Allows users to be registered within the system. |
| **Involved actors** | App user |
| **Pre-conditions** | - |
| **Post-conditions** | The user is registered |
| **Basic Path** | 1. The user fills the form with the required parameters.<br>2. The user presses the register button.<br>3. The system processes the information and displays a notification. |
| **Exceptional Path** | 2. Any form field is empty or wrongly written: Exception |

Table 7.2: Authentication

| **UC-02** | **Authentication** |
|---|---|
| **Description** | Allows users to be authenticated within the system |
| **Involved actors** | App user |
| **Pre-conditions** | UC-01: User needs to be registered in the system. |
| **Post-conditions** | The user is authenticated |
| **Basic Path** | 1. The user fills the authentication form with the required parameters.<br>2. The user presses the authenticate button.<br>3. The system processes the information and authenticates the user. |
| **Exceptional Path** | 2. Any form field is empty or wrongly written: Exception |

Table 7.3: Edit profile

| **UC-03** | **Edit profile** |
|---|---|
| **Description** | Allows users to see their personal info, as well as modify it |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | UC-02: User needs to be authenticated in the system. |

*(continues on the next page)*

Table 7.3 – *(continues from the previous page)*

| **UC-03** | **Use Case** |
|---|---|
| **Post-conditions** | - |
| **Basic Path** | 1. The user presses the profile button.<br>2. The user can see their personal info as well as edit it.<br>3. The user can return to the main page and the info is saved. |
| **Exceptional Path** | 2. Username is too large or contains unsuported characters: Exception. |

Table 7.4: Send request

| **UC-04** | **Send request** |
|---|---|
| **Description** | Allows users to send chat requests to other users |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | UC-02: User needs to be authenticated in the system. |
| **Post-conditions** | - |
| **Basic Path** | 1. The user presses the requests button.<br>2. The user can send requests by writing the email of the receiver. |
| **Exceptional Path** | 2. Email does not exist or is wrongly formatted: Exception. |

Table 7.5: Accept/Decline request

| **UC-05** | **Accept/Decline request** |
|---|---|
| **Description** | Allows users to accept or decline chat requests from other users |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | UC-02: User needs to be authenticated in the system. |
| **Post-conditions** | If the chat request is accepted, the chat is added to the chat list. |
| **Basic Path** | 1. The user presses the requests button.<br>2. The user scrolls to the received requests screen.<br>3. The user can accept or decline a request. |
| **Exceptional Path** | |

Table 7.6: Log out

| UC-06 | Log out |
|---|---|
| **Description** | Allows users to log out of their account. |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | UC-02: User needs to be authenticated in the system. |
| **Post-conditions** | The user is logged off the system and rerouted to the Login screen. |
| **Basic Path** | 1. The user presses the log off button. |
| **Exceptional Path** | |

Table 7.7: View chat

| UC-07 | View chat |
|---|---|
| **Description** | Allows users to enter a chat room where they can send messages to another user. |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | UC-02: User needs to be authenticated in the system |
| **Post-conditions** | |
| **Basic Path** | 1. The user presses the respective chat.<br>2. The user is rerouted to the chat screen corresponding to that chat. |
| **Exceptional Path** | |

Table 7.8: Send message

| UC-08 | Send message |
|---|---|
| **Description** | Allows to send a text message. |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system<br>2. UC-07: User needs to be within a chat room. |
| **Post-conditions** | The message is sent to the receiver. |
| **Basic Path** | 1. The user writes a message.<br>2. The user presses the send button.<br>3. The text message is sent via Firebase to the receiver. |
| **Exceptional Path** | |

Table 7.9: Send data

| UC-09 | Send data |
|---|---|
| **Description** | Allows to send an image or a video. |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system<br>2. UC-07: User needs to be within a chat room. |
| **Post-conditions** | The image or video is sent to the receiver. |
| **Basic Path** | 1. The user selects the image/video message button.<br>2. The user selects a file to send.<br>3. The file is sent via Firebase to the receiver. |
| **Exceptional Path** | |

Table 7.10: Remove messages

| UC-10 | Remove messages |
|---|---|
| **Description** | Allows users to remove messages |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system<br>2. UC-07: User needs to be within a chat room. |
| **Post-conditions** | The selected messages are removed from the database. |
| **Basic Path** | 1. The user selects the messages to remove.<br>2. The user presses the remove button.<br>3. The user accepts to remove the messages.<br>4. The messages are removed. |
| **Exceptional Path** | |

Table 7.11: Copy messages

| UC-11 | Copy messages |
|---|---|
| **Description** | Allows users to copy messages |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system<br>2. UC-07: User needs to be within a chat room. |
| **Post-conditions** | The selected messages are copied to the users clipboard. |

*(continues on the next page)*

Table 7.11 – *(continues from the previous page)*

| UC-11 | Use Case |
|---|---|
| **Basic Path** | 1. The user selects the messages to copy.<br>2. The user presses the copy button.<br>3. The messages are copied. |
| **Exceptional Path** | |

Table 7.12: Download encoded data

| UC-12 | Download encoded data |
|---|---|
| **Description** | Allows users to download encoded data |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system. |
| **Post-conditions** | The data is downloaded and saved in the local device. |
| **Basic Path** | 1. The user opens an encoding history element.<br>2. The user selects the download button. |
| **Exceptional Path** | 2. The data selected by the user is not encoded yet: Exception. |

Table 7.13: Download decoded data

| UC-13 | Download decoded data |
|---|---|
| **Description** | Allows users to download decoded data |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system. |
| **Post-conditions** | The data is downloaded and saved in the local device. |
| **Basic Path** | 1. The user opens a decoding history element.<br>2. The user selects the download button. |
| **Exceptional Path** | 2. The data selected by the user is not decoded yet: Exception. |

Table 7.14: Encode data

| UC-14 | Encode data |
|---|---|
| **Description** | Performs esteganographic operations in the data provided |
| **Involved actors** | Authenticated App user |

*(continues on the next page)*

Table 7.14 – *(continues from the previous page)*

| UC-14 | Use Case |
|---|---|
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system |
| **Post-conditions** | The data is processed and the user is directed to the encoded data screen. |
| **Basic Path** | 1. The user picks the encoding options, payload and carrier data.<br>2. The user selects the encode button.<br>3. The user is directed to the loading screen. |
| **Exceptional Path** | 2. The data selected by the user is not complete or is not compatible: Exception. |

Table 7.15: Decode data

| UC-15 | Decode data |
|---|---|
| **Description** | Performs esteganographic operations to decode data from the file provided |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system |
| **Post-conditions** | The data is processed and the user is directed to the main screen. |
| **Basic Path** | 1. The user selects the file to decode.<br>2. The user writes the password if there is any.<br>3. The user selects the decode button.<br>4. The user is directed to the loading screen. |
| **Exceptional Path** | 2. The password written is not correct: Exception. |

Table 7.16: Analyze image

| UC-16 | Analyze image |
|---|---|
| **Description** | Analyze an image to search for esteganographic activity |
| **Involved actors** | Authenticated App user |
| **Pre-conditions** | 1. UC-02: User needs to be authenticated in the system |
| **Post-conditions** | The data is processed and the user is directed to the analysis screen. |
| **Basic Path** | 1. The user selects the files to analyze.<br>2. The user selects the analyze button.<br>3. The user is directed to the loading screen.<br>4. The user is directed to the analysis screen. |

*(continues on the next page)*

Table 7.16 – *(continues from the previous page)*

| UC-16 | Use Case |
|---|---|
| **Exceptional Path** | 2. The images do not match in size or format: Exception. |

### 7.3.1 Non-functional requirements

Non-functional requirements enclose a set of specifications clarifying the operational capabilities and constraints of a system, with the primary aim of enhancing its functionality and behaviour. These requirements delineate the system's operational proficiency, encompassing attributes such as performance, security, dependability, data integrity, and other factors. A list of non-functional requirements has been created for **Steg'o**, as shown in table 7.17.

Table 7.17: Non-functional requirements for **Steg'o**

| ID | Requirement Description |
|---|---|
| NFR01 | **Steg'o** chats shall have a response time of no more than **3 seconds** for **95%** of the operations. |
| NFR02 | **Steg'o** image steganographic functionalities shall be able to encode/decode a message within **5 seconds** for images up to **8MB** in size. |
| NFR03 | All transmitted data shall be encrypted using modern cryptographic standards to ensure user privacy and data security. |
| NFR04 | **Steg'o** app shall support concurrent usage by up to 1,000 users without significant performance degradation. |
| NFR05 | **Steg'o** chats shall store messages in Firebase with a retrieval time of no more than **6 seconds** for recent messages (last 7 days). |
| NFR06 | In the event of a failure in the API connection, the chat app shall notify users and attempt to reconnect every 15 seconds. |
| NFR07 | The app shall be compatible with the latest three versions of Android. |
| NFR08 | All user passwords stored in Firebase shall be hashed and salted to ensure their security. |
| NFR09 | **Steg'o** shall provide clear and concise error messages to users in the event of any failures or issues. |

## 7.4 Diagrams

Figure 7.1 illustrates a simplified flowchart representing the application's functionality. It is important to note that this diagram provides a foundational overview and does not encompass all potential scenarios within the realm of steganography. Nevertheless, it serves as an adequate tool for gaining insight into the inner mechanisms of the application.

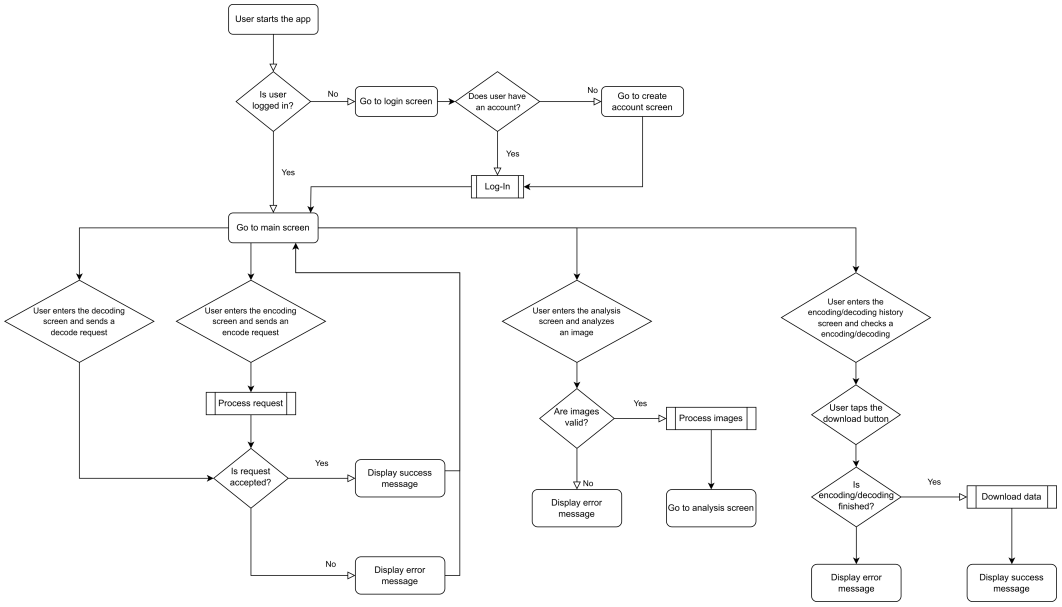



Figure 7.1: **Steg'o** simplified flowchart

Figure 7.2 shows a class diagram depicting the architecture of the **Steg'o** *backend* system. Within this diagram, two principal classes, named the "Obfuscator" and "Deobfuscator," assume responsibility for executing core operations and overseeing the management of API handling. Additionally, the diagram showcases the presence of four different "codecs," which are dedicated to the implementation of separate encoding and decoding functionalities, each tailored to specific steganographic requirements.

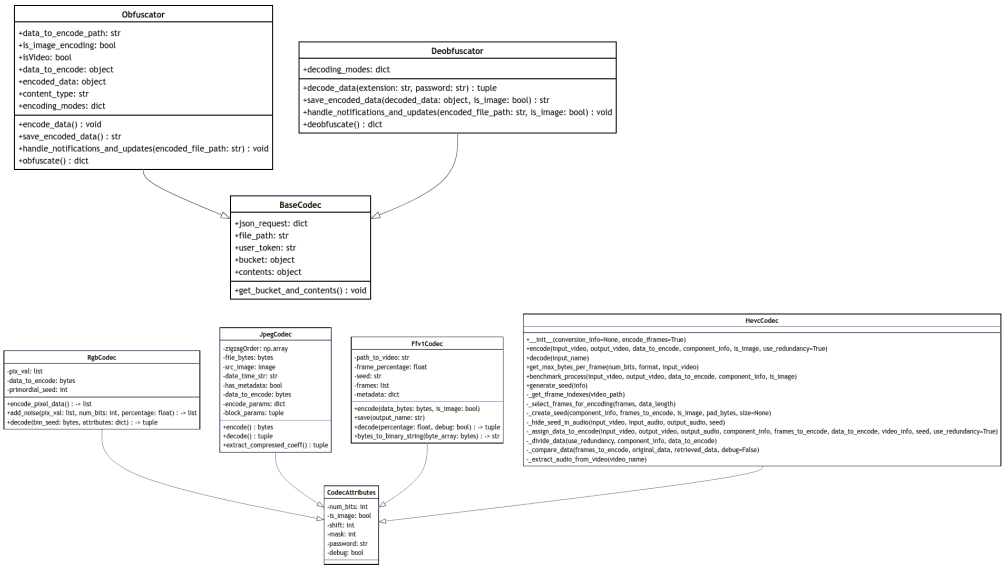



Figure 7.2: **Steg'o** backend class diagram

The sequence diagram denoted as 7.3 describes a specific case involving the processing of an image for performing spatial steganography on it.

As mentioned previously, within the application's architecture, the **Obfuscator** class assumes responsibility for overseeing the **input** and **output** procedures associated with the encoding operations carried out within the API. In contrast, the **codec** module is dedicated to the intricacies of the encoding process itself, immediately following a series of preliminary actions involving **format normalization** and **seed generation** operations.

This ensures a clear distinction of roles and responsibilities within the system.

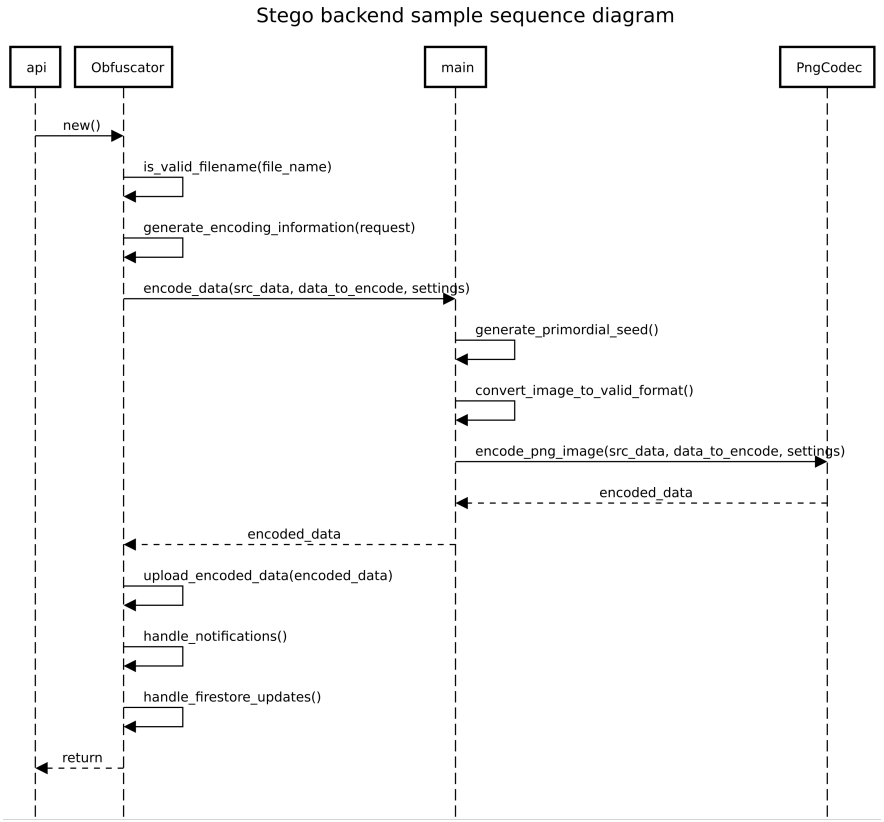



Figure 7.3: **Steg'o** spatial steganography sample sequence diagram

Chapter 8

# System design and general structure

This chapter will explain the general design choices behind **Steg'o**, both in its backend and frontend sections.

## 8.1 Arquitecture

### 8.1.1 System global arquitecture

The software solution utilize an architectural approach based on the client-server model. In this model, a client program is responsible for making requests, while a server program handles the corresponding responses. For visual representation of the client-server model, refer to Figure 8.1.

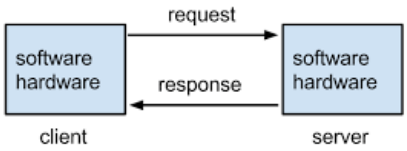



Figure 8.1: Visual explanation of client-server model [1]

In the implemented project, the client component is an *Android* mobile app developed in *Flutter*. Though the application can theoretically run in both *Android* and *iOS*, as it is programmed entirely in a multiplatform language, only *Android* devices were tested.

Users can interact with the application through their respective mobile phones, initiating requests that will then be received and processed by the server. The server makes use of a REpresentational State Transfer (REST) API as the mechanism to access the application's functionality and business logic. Communication between the client and server is provided through Hypertext Transfer Protocol (HTTP) requests.

On the server side, the implementation was carried out by using *Python* and *Firebase*, where the user data was stored.

Figure 8.2 presents an overview of the comprehensive architecture employed in the developed software solution.

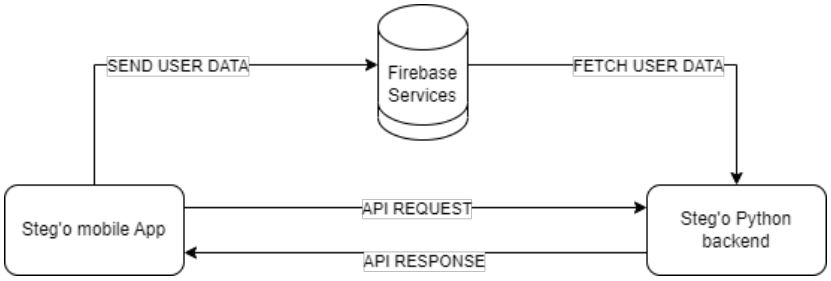



Figure 8.2: Visual overview of **Steg'o** architecture

### 8.1.2 Frontend design pattern

The chosen design pattern for the **frontend** was the Provider pattern [30], which enables state management in applications via a cohesive system of models and views. This pattern is incorporated within the Flutter framework and can be accessed through the "provider" package.

This design pattern is grounded in the utilization of models and views to establish an abstraction layer between the application and the underlying data. The views gain access to the data intended for visual representation on the device through an intermediary entity referred to as the **"provider"**. As suggested by its name, the provider provides the screens with processed data to ease their accurate visualization.

Upon the modification of the data within its corresponding model, all current views with a provider of that kind undergo state updates. This mechanism produces a highly adaptable and effortless approach to state management in mobile applications. Such an approach improves system scalability and promotes the potential for code reusability within models, either via inheritance or direct integration within alternative views.

### 8.1.3 API REST

The **backend** of the system utilizes a REST API architecture, which serves as an access point for the application to access its implemented functionality. This type of architecture was chosen because of its versatility and speed, as it provides access points to business logic that can be used by a wide variety of systems. The REST application is organized according to a layered system, which is prevalent in architectures of this nature.

The application exposes the following endpoints: "**/deobfuscate**" and "**/obfuscate**"

These endpoints facilitate the encoding and decoding of diverse data types, as the particular data type and encoding configuration is specified within the HTTP packet body. Upon completing this initial step, the required class is instantiated and executed,depending on whether it is an encoding or decoding operation.

The data sourced from Firebase is downloaded by using a path specified within one of the parameters. Subsequently, invocations to the codecs are made. These codecs assume responsibility for the encoding and decoding processes associated with each specific data type.

There are currently four codecs: **JpegCodec**, **PngCodec**, **Ffv1Codec** and **HevcCodec**.

Each of the aforementioned codecs is versatile, performing both encoding and decoding operations relating to any compatible data format, as long as it is in byte format. Subsequent to the successful execution of the operation, the resultant outcome is uploaded to Firebase.

Furthermore, the document detailing the executed operation is updated to ensure accessibility for users through Firebase's historical interface. Additionally, users are alerted of the completed operation via a push notification system.

## 8.2 Database

**Firebase**, a cloud database managed by *Google* and implemented using **NoSQL** [31], was chosen for the administration of this system. This choice was mainly due to the ease of **Firebase** to manage user data, profile pictures and other relevant elements natively, in addition to providing several complementary systems such as **Push Notifications** for *Android* and *iOS*. Firebase is widely used in mobile development due to its flexibility, ease of management, and offline caching and data management systems.

*Firebase* provides two types of *NoSQL* databases: **Realtime Database** and **Cloud Firestore**. These databases differ from traditional relational databases in terms of their data modeling approach.

### 8.2.1 Realtime Database

**Firebase**'s original database offer is **Realtime Database**. It stores data as JavaScript Object Notation (JSON) objects, enabling real-time synchronization among connected devices. Some of its features are:

- **Offline support:** When offline, any modifications made are automatically synced once the client regains an internet connection.
- **JSON data structure:** Data is organized hierarchically, resembling a tree structure where nodes represent objects.

- **Real-time updates:** Changes made to the data are immediately synchronized across all connected clients.

### 8.2.2 Cloud Firestore

**Cloud Firestore** is a more advanced database provided by *Firebase*. It also operates as a *NoSQL* database, but it utilizes a document-based approach for data storage. Some of its features are:

- **Document-based data model:** Data is organized into collections of documents, with each document containing key-value pairs.
- **Offline support:** Clients can perform read, write, and query operations on data even when offline, ensuring uninterrupted functionality.
- **Real-time updates:** Similar to the *Realtime Database*, data changes are instantly propagated to all connected clients.

In this project, *Firestore Database* was chosen as the preferred solution due to its status as the most recent offering from *Firebase* for managing document-based databases, coupled with the development team's prior experience with it.

Theoretically, within a document-oriented database, each document within a collection has the potential to possess unique characteristics that distinguish it from all other documents. Consequently, the utilization of a Unified Modeling Language (UML) class diagram becomes impractical, as classes assume shared attributes and behaviors among their instances. However, in real scenarios, documents belonging to the same collection often exhibit common elements. In such cases, a UML diagram can be employed to represent these shared elements and relationships.

In the subsequent diagram (Figure 8.3), nested collections are depicted using **one-to-many** relationships (for instance, between the **Chat collection** and the **Message collection**), while reference relationships are represented by **one-to-one** relations.

In the context of the **users** collection, each user is assigned a collection of **chats** and a collection of **requests**. Each of these collections contains documents that reference specific **chats** and **requests**. This design choice is built on the need to store user-specific data alongside references, exemplified by the inclusion of metrics such as **unread message counts** for distinct **user-chat pairs**.

The references in this context point to singular documents, with each document within the **Chats** collection containing a reference to a corresponding **User** document. So, for each document in the **Chats** collection within a **User** document, there is only one **Chat** to which it refers.

Conversely, each user maintains an exclusive collection of encoding and decoding operations. This collection houses documents generated as users execute operations, containing data assigned to individual encoding or decoding processes, encompassing factors such as implicated files, bit allocation, bit shifts, and so forth.

In contrast to the previous examples, these collections operate under distinct principles, since they do not have to be shared among more than one user. Consequently, the reliance on references for storage is not needed within these contexts.

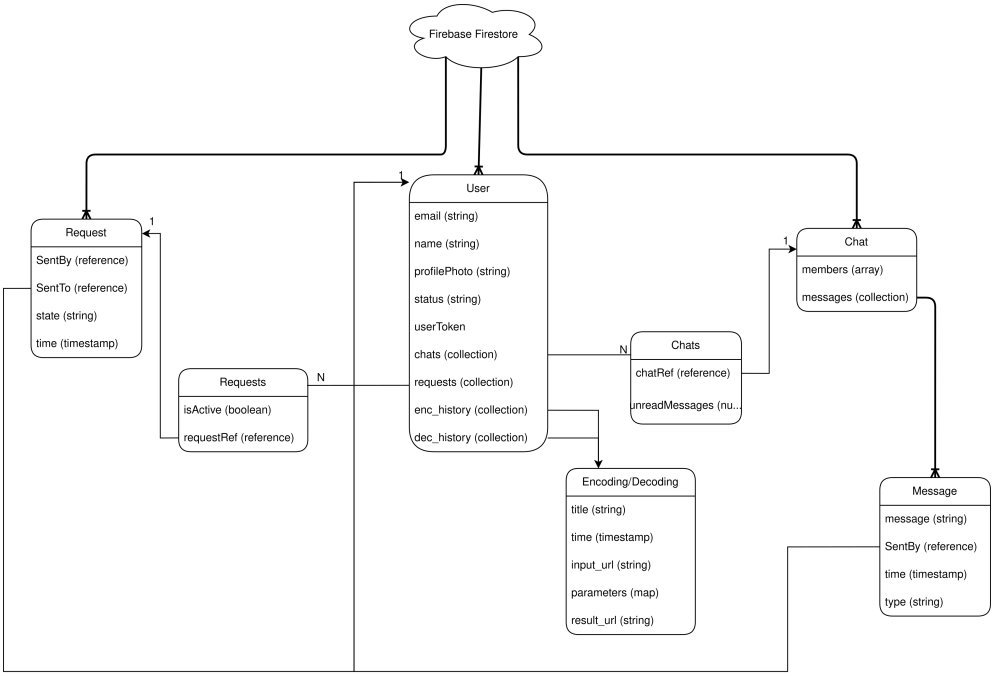



Figure 8.3: Diagram of collection and document relationships in **Steg'o** Firestore enviroment

## 8.3 User Interface Design

The UI design plays a crucial role in the success of mobile applications within the current market landscape. The way in which data is presented to users through the UI can significantly influence the adoption and popularity of an application. However, in chapter 3 was observed that many existing steganography applications prioritize functionality over design, resulting in usable yet aesthetically unappealing products.

To address this issue, **Steg'o** aims to give a twist to the conventional approach by offering an application that not only boasts visually appealing design but also provides a seamless and comfortable user experience. Furthermore, **Steg'o** is committed to supporting users with comprehensive manuals and explanations to facilitate their understanding and utilization of the application's core features.

### 8.3.1 Mockups and Wireframes

In the initial stage of the development process, *Balsamiq Wireframes* was employed to create several *mockups* for visualizing the intended appearance of the UI across various screens within the mobile application. This approach enables a simplified representation of the overall screen design, also facilitating effective client-developer communication.

### 8.3.2 App general design

The application's development adhered to the established principles of mobile application design. The following list highlights some of them:

- **Identify the target customer:** The design of user interactions should account for the diverse requirements and preferences of distinct user groups. In **Steg'o**, the target customers are both experienced steganography enthusiasts and people that just want to have fun using its features, without prior knowledge of its inner workings.
- **Prompt and guide the user:** Prompting user feedback and providing guidance are crucial for ensuring satisfactory user interaction outcomes and fulfilling user needs in subsequent operations.
- **Maintain consistency:** The application must demonstrate consistency in its design, appearance, and behavior throughout its various elements. **Steg'o** reflects this principle by utilizing Material Design [32], developed by Google, which enables developers to have a cohesive design in all their screens, by using the packages provided.

## 8.4 Brand image

Corporate identity encompasses the distinctive attributes that distinguish a company or personal brand, from others. These attributes comprise various elements, including corporate design, nomenclature, fonts, logos and colours.

In the preceding section, the general design principles followed by **Steg'o** were explained. Now, the focus shifts to the specifically crafted name and logo, which form integral components of its corporate identity.

### 8.4.1 App name

The selection of an application's name holds significant importance as it serves as first impression for users when searching in app stores. Therefore, it is crucial for the name to be both simple and clear. Additionally, it is imperative for the app name to be original in order to differentiate the application from competitors.

After conducting an extensive study involving various alternatives, the chosen name for the application is **Steg'o**. The rationale behind the selection of this nomenclature encompasses multiple factors:

- The initial component, "**Steg**" alludes to the field of steganography, which involves the concealment of information within other media.
- The subsequent element, represented by the letter "**o**" draws inspiration from the *Stegosaurus*, an ancient dinosaur species.

The branding identity embodies a fusion of these two concepts, with a predominant green color scheme evoking associations with dinosaurs. Moreover, a deeper connection exists between the realms of steganographic procedures and paleontology, as the user metaphorically excavates hidden elements or data from an ostensibly deserted setting.

### 8.4.2 App design

**Steg'o** is designed with simplicity in mind, aiming to ensure user-friendliness even for individuals lacking prior experience. However, it also provides a large variety of functionalities to satisfy the needs of advanced users.

To facilitate the user's interactions, drop-down menus and information icons have been strategically incorporated into the application, serving as guidance and assistance tools for users during their interactions with the system. Additionally, the color palette employed in the interface features two variations of green, purple and a dark gray shade, imparting a visually appealing and approachable appearance. The **Steg'o** logo (Figure 8.4) was conceived by *Skygge* [33], a proficient illustrator with a great background in design and logo creation.

Figure 8.4: **Steg'o** logo

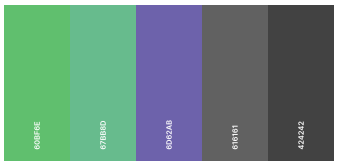



Figure 8.5: **Steg'o** color palette

Chapter 9

# Steganography methods

This chapter will provide a comprehensive description of the different procedures implemented in the **Steg'o**'s back-end, as well as the steganographic methods and strategies that were investigated for possible development, and subsequently chosen for the current project.

## 9.1 Steg'o seeding methods

The steganographic techniques employed by **Steg'o** utilize a seed-based system to produce pseudo-random values that determine the location for data insertion. Below, a more specific explanation of this seed-based system is provided, together with a couple of illustrative examples for better comprehension.

The following figures depict the pixels that have been subjected to steganography in an illustrative image. In Figure 9.1 (*left*), the pixel modifications are conducted in a linear manner, resulting in an alteration that is easily discernible to an external observer.

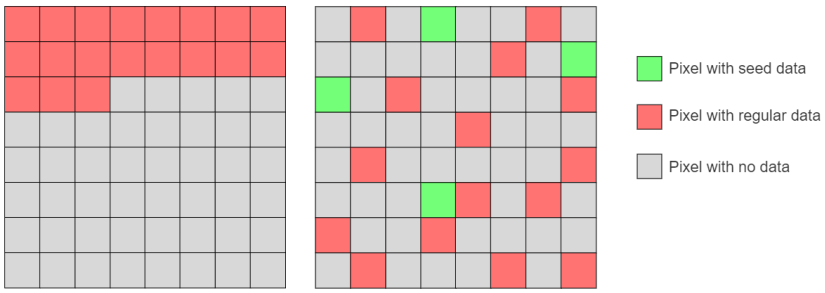



Figure 9.1: Pixels modified without a randomization technique (left) and using a seed-based randomization system (right)

In contrast, Figure 9.1 (*right*) demonstrates a different approach, as the same number of modifications have been applied to the image pixels but have been distributed in a pseudo-

random manner via a seeding system. This results in a more intricate and elusive alteration that is less evident and more challenging to detect.

It is noteworthy that this methodology can be applied to any data format characterized by values arranged in a mesh structure. This randomization technique described can be extended not only to image steganography approaches that do not directly alter pixel values (e.g., frequency domain steganography), but also to individual frames within video sequences.

In image steganography using **Steg'o**, the image encoding seed plays a crucial role. The seed consists of 12 bytes that are partitioned and used to generate a list of pseudo-random values for data insertion.

Specifically, the image encoding seed has the following structure:

| **dataLen** + controlByte0 + **JPEGspecs** + controlByte1 + X + **numBitsString** + controlByte2 |
|---|

Therefore, an example of possible seed would be the following:
**x03,x17** / **EÝÃ$** F 5 **x02** 1

- The **dataLen** field is composed of **3 bytes** that indicate the length in bytes of the data to be encoded. In the case of data encoded as RGB values, it specifies the **width** and **height** of the image in **12 bits per component**. In this example, the value is '**x03,x17**', which, when translated to an integer number, reveals that the lenght of the encoded data is **207.895 bytes**.
- The **JPEGspecs** field is composed of **4 bytes** and only used in frequency domain encoding. It specifies the minimum and maximum number of AC coefficients to be encoded, with values ranging from **1 to 63**. The concept for AC coefficients is explained on further detail in page 52.

  In the seed example, the value of **JPEGspecs** is '**EÝÃ$**', which can be represented as a **32-bit value**. Then, the last **12 bits** can be extracted and divided into **two 6-bit values**, that when converted to integer numbers show that the range of modifiable DC elements is between **12** and **36** per quantized JPEG matrix.
- The **numBitsString** field is a **1-byte** value that contains various parameters such as the number of bits per byte that are modified, the bit shift of the modified bits, or whether the image has been encoded using spatial or frequency domain steganography.

  Using the previous example, the **numBitsString** has a value '**x02'**, i.e., its binary value can be represented as 00000010. If split correctly, the values for the different parameters can be disclosed. Beginning by the most significant bit (left):

  - **Bit shift value**. This variable can range from **0 to 7** and represents the shift of the bits in the encoding. For example, with a bit shift of 3, the encoded (modified) bit

will be the fourth less significant bit within each byte (i.e, 1111**0**111). In the previous example, the bit shift was 000, so the encoded bit will be the less significant bit of each byte.

- **RGB boolean value**. A single bit that specifies if the encoding is done in the spatial or frequency domain, respectively. In this example, the RGB boolean bit is set to 0, so the encoding is done in the frequency domain.
- **Number of bits** to encode per byte of original data. This variable is composed by **four bits** and can range from **0 to 8**, specifying how many bits will be encoded per byte of original data. In the current example, its value is 0010, so the number of bits that will be modified per byte will be 2, thus, 11111**00**.

- The **controlBytes** field consists of **3 bytes** with a constant value that is inserted in specific positions to detect whether the seed has undergone modifications. In this case, the control bytes are set to '**/F1**'.

Even if a seed is employed to generate a series of positions for data insertion, the question is how we can accurately determine the location of the seed (see Figure 9.1). To address this concern, the **metadata** associated with the respective image or video is brought into play. Specifically, by extracting the **creation date** of the image or video and converting it into a string representation in the form of a timestamp, this **'primordial seed'** can be utilized in a similar way to the ordinary seed to determine its position within the data.

The proposed solution exhibits high flexibility by enabling the generation of diverse **'primordial seeds'** based on the image metadata. Furthermore, it can be seamlessly applied to any constant component involved in image creation and transmission, such as the **password** employed for data encryption, if used.

## 9.2 Image steganography

Image steganography can be divided into two major groups:

- **Spatial domain** steganography: changes made to the image are carried out directly on the pixel values of the image.
- **Frequency domain** steganography: changes made to the image are carried out on the frequency values of an image that allows it, such as those using JPEG format.

In this application, both of these techniques were performed for different image file formats.

### 9.2.1 Spatial domain steganography

Steganography in the spatial domain involves the manipulation of the Red Green Blue (RGB) values of pixels in a given image to embed data at specific bit locations. This technique is also called Least Significant Bit (LSB) because the modification is often performed in lower significance bits, which minimizes possible detection. Given this direct modification of the pixel values, this technique can only be performed in lossless image formats, such as PNG.

An example of applying modifications to the RGB values in a image is showed in Figure 9.2, with the extent of alteration in the pixel values being influenced by the number of modified bits per byte and their shift.

This particular form of steganography is vulnerable to both visual detection by a third party and steganographic analysis. Additionally, if any modifications are made to the image, such as applying filters or recompressing, it is much less likely that the information concealed within the image is retrievable than if using frequency domain steganography.

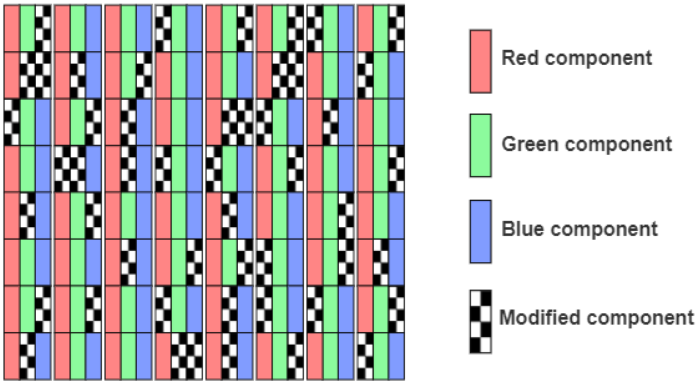



Figure 9.2: Visual explanation on RGB components modification

On the other hand, this strategy is a simpler and a much more useful way of concealing data if the user prioritizes the data insertion size over the undetectability of the data. Each pixel can be modified using LSB techniques and visual detection is only likely to happen if the number of bits modified surpasses **2 bits / byte**. For example, in an **1920x1080** image, there are **2.073.600** pixels, which mades up **6.220.800** RGB components that can be modified. Assuming a bit depth of **8 bits per component**, if a 2 bit LSB steganography attack is performed in this image, the potential data insertion could be up to **1,55 MB**.

The following figure (9.3) illustrates various forms of data insertion applied to an image. The leftmost picture (9.3a) depicts an image in which **7 bits per byte** have been modified. In the middle figure (9.3b), the modification consists of **2 bits per byte**, while the figure on the

right side (9.3c) represents the original one, unmodified image.

It is clearly discernible that the image with **7-bit** modification exhibits prominent artifacts that are totally visible to the human eye. On the other hand, the image with **2-bit** modification displays artifacts that, while present, are scarcely perceptible.

(a) 7 bits / byte modifications (b) 2 bits / byte modifications (c) Original image

Figure 9.3: Spatial domain steganography performed at different bit levels

The following Python code exemplifies the deployment of bit shifting and masking techniques to recreate a steganographic procedure on an integer array, utilizing the *Numpy* python package [34] for array manipulation.

```
rgbValues[shuffledIdx] = (rgbValues[shuffledIdx] & mask |
    dataToInsert << shift)
```

Listing 9.1: Python code for replacing bits of data with other data

In this particular case, the array should be derived from the RGB color values of the image pixels. Nevertheless, this methodology can be suitably adapted to manage diverse data representations, with the only requirement of being an **8-bit depth** data format, for shifting and masking purposes.

This steganography method is probably the simplest of all implemented, since the encoded values are stored in the RGB components of the image itself in a clear and visible form for a thorough analysis. The evaluation of the security level encompasses three parameters, previously explained in Chapter 2.

In this case, we can conclude that:

- **Privacy:** By introducing the option to encrypt the inserted data, the method ensures **maximum data privacy**.
- **Robustness:** This particular steganography method exhibits limited robustness due to the susceptibility of the encoded information to even minor alterations in pixel values. Consequently, when anti-steganographic attacks are employed by the use of **filters**, subtle **distortions**, or the addition of **noise** layers to images or videos, the use of this method to perform a lossless steganographic process becomes unfeasible.

- **Undetectability:** As with any steganographic process, the ability to remain undetected is inversely proportional to the volume of data inserted. For instance, if we embed a **50-character sentence** into a **1920x1080 pixel** image using **1 bit per component**, the proportion of modified pixels would be approximately $1 \times 10^{-6}$% of the total pixels, an minuscule quantity that significantly inhibits data detection. However, it is worth noting that this spatial steganography method is relatively susceptible to detection, and therefore its rating in this regard may not be high.

### 9.2.2 Frequency domain steganography

Frequency domain steganography takes the basic operations that occur in spatial steganography one step further. This approach is commonly employed in file formats that utilize mathematical transformations for compression purposes. In this case, the transform used is the Discrete Cosine Transform (DCT). The DCT is prominently employed in compression algorithms such as JPEG, where its primary function is to represent the visual data of an image as a function of cosine waves. This section aims to provide a concise explanation of JPEG compression algorithm and its implications for image steganography.

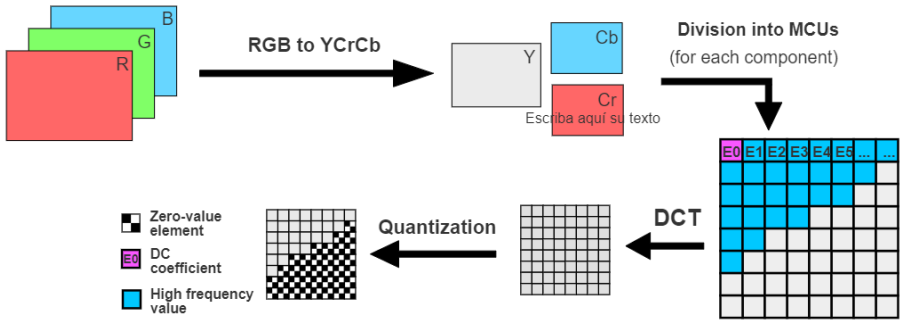



Figure 9.4: Overview for most relevant stages in JPEG compression

#### JPEG: A brief insight

JPEG employs a series of procedures to compress an image file effectively. (Figure 9.4 Initially, the RGB components of the image are converted to the *YCrCb* (or *YUV*) color space, which represents the **luminance**, **red chrominance**, and **blue chrominance** of each individual pixel. This representation is widely used in various compressed formats, as it allows the luminance component to be separated from the chrominance components. This separation enables further optimization in compression, as human visual perception is more sensitive to changes in luminance than in color variations. By leveraging this characteristic, **chroma subsampling** can be applied to the *YCrCb* image (refer to page 58), resulting in not only

reduced file size but also decreased bandwidth consumption during transmission.

Once the image is transformed into the *YCrCb* color space and the **chrominance subsampling** procedure is applied, the resulting image components are then partitioned into Minimum Coding Units (MCUs), consisting of $8 \times 8$ pixel blocks. When the image MCUs are set, the compression process begins.

In this sense, a series of sequential steps are undertaken with the objective of reducing the file size through the implementation of multiple techniques. To begin with, a two-dimensional DCT is applied to every MCU of each component. This transformation operation aims to convert the **luminance** and **chrominance** blocks containing the physical image data into numerical blocks that represent the pixel information in the frequency domain. It is important to note that this step **does not result in any loss of information**. After the DCT application, a frequency $8 \times 8$ block is obtained for each spatial $8 \times 8$ MCU.

Upon the completion of the frequency domain conversion, the quantization process is initiated. To accomplish this, an $8 \times 8$ quantization matrix of predetermined values is employed, although the specific matrix utilized for this step may vary depending on the encoder or file under consideration. The purpose of this matrix is to eliminate (or represent with less accuracy) the values linked to the higher frequencies, which are positioned in the lower-right section of the frequency blocks for the MCUs (Figure 9.4). This strategy holds significant importance as high-frequency values exhibit **minimal perceptibility** to the human visual system. Consequently, their removal from an image does not affect substantially to its visual quality, but it does in its size.

Typically, the maximum value within each frequency block corresponds to the lowest frequency component, positioned in the upper left corner. This particular value is denoted as the **DC coefficient**. In contrast, the remaining values corresponds to higher frequencies and consequently contribute less significantly to the final representation of the image, resulting in smaller numbers. These values are commonly referred to as AC coefficients. For an illustrative example of the DCT application and quantization process in a real MCU, refer to Figure 9.5.

$$\begin{bmatrix} -415 & -33 & -58 & 35 & 58 & -51 & -15 & -12 \\ 5 & -34 & 49 & 18 & 27 & 1 & -5 & 3 \\ -46 & 14 & 80 & -35 & -50 & 19 & 7 & -18 \\ -53 & 21 & 34 & -20 & 2 & 34 & 36 & 12 \\ 9 & -2 & 9 & -5 & -32 & -15 & 45 & 37 \\ -8 & 15 & -16 & 7 & -8 & 11 & 4 & 7 \\ 19 & -28 & -2 & -26 & -2 & 7 & -44 & -21 \\ 18 & 25 & -12 & -44 & 35 & 48 & -37 & -3 \end{bmatrix}$$

(a) DCT matrix

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

(b) Quantization matrix

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -3 & 4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(c) Result of quantization

Figure 9.5: Example of the DCT and quantization operations

In the quantization process, the DCT matrix values are divided by quantization matrix values, resulting in a removal of almost every high frequency value in the original data, as

they are all converted to zero elements.

Figure 9.6 illustrates the utilization of the techniques used in the last stages of the JPEG compression procedure. Firstly, **Zig-Zag rearranging** is performed for each of the individual quantized MCUs to group the zero values generated in the previous step (quantization). This technique is based on ordering the frequency coefficients within each MCU according to their frequency relevance (from low to high frequencies). Then, **Delta encoding** (also known as Differential Pulse-Code Modulation) is applied to normalize the encoded values with respect to the initial DC value of the first MCU. Following the **Delta encoding** procedure, a Run Length Encoding (RLE) technique is carried out, which allows to reduce the size of the grouped values generated in the previous **Zig-Zag encoding** step.

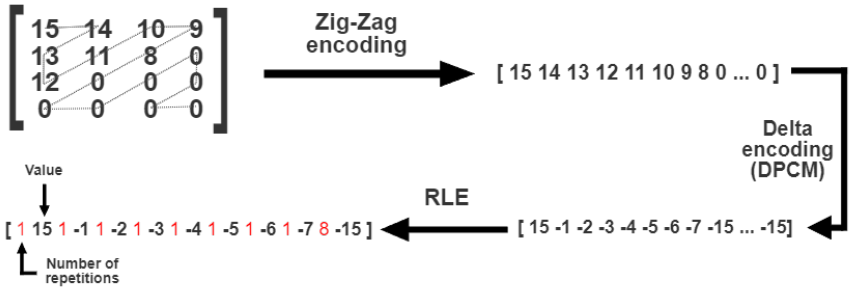



Figure 9.6: Procedure for last stages of JPEG compression with an example matrix

As the last step, a **Huffman tree** is employed to allow variable-length encoding, thereby achieving a further reduction in the overall file size. For a detailed explanation of Huffman coding and Huffman trees, please refer to the work in [35].

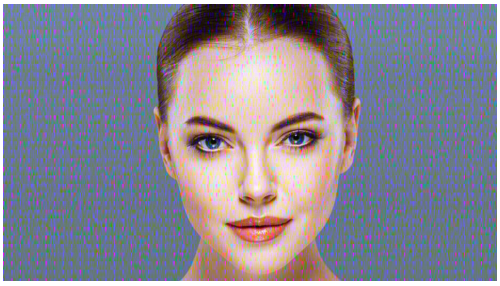
Following with the explanation on frequency domain steganography, in this particular scenario, the alterations are applied to the MCU components which are obtained after the quantization process. This approach ensures that the loss of data experienced during quantization is not propagated to the data to be concealed, resulting in a lossless steganographic procedure.

However, this methodology presents certain drawbacks. Given that the highest frequency components of the image have already been eliminated, modifications made to quantized matrices will be more discernible.
Hence, two main problems arise:

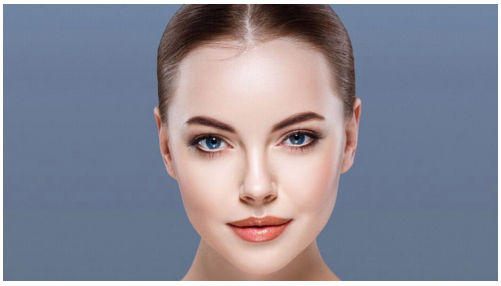
1. Firstly, if data is embedded in the **high frequency values** (bottom right) of each MCU, the compression process carried out by the aforementioned algorithms will not perform correctly since quantization-removed values may regain non-zero status. As a result,

this can significantly increase the file size and arouse suspicions regarding a possible analysis.

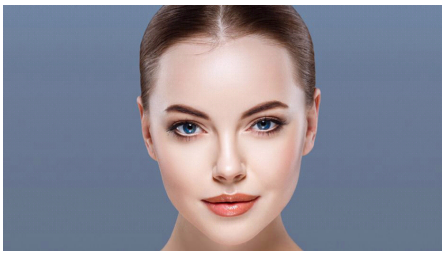
2. Alternatively, when data is inserted into **low frequency values**, even minor alterations (1 or 2 bits) may be distinctly noticeable in the visual representation of the compressed image. The impact becomes more pronounced if the DC coefficient of the MCUs is modified, as this change propagates throughout the entire file.



(a) 7 bits / byte modifications with 12-16 modified elements per block



(b) 2 bits / byte modifications with 18-32 modified elements per block



(c) Original image

Figure 9.7: Frequency domain steganography performed with different configurations

Therefore, the optimal solution to this issue is to abstain from modifying the **DC coefficients** and instead limit the alterations on a few elements with medium or low frequencies that already possess moderate values. By restricting the modifications to **1 or 2 bits** in these elements, the visual impact becomes practically imperceptible.

Thereby, the number of elements to be modified per MCU must be determined. Through extensive experimentation, a consensus has been established, indicating that an optimal range of modifications lies between **16 and 30 elements per block** (each block comprises 64 elements). This range has been proven to be effective in minimizing artifacts while concurrently maximizing the capacity for insertable data. For visual reference, Figure 9.7 presents various examples illustrating several bit levels and quantities of modified elements.

The following fragment of code (9.2) illustrates a simplified implementation of the frequency domain steganographic procedure in images.

```
1 yBlocks = yQuantMatrix[:yRows - yColumns % A, :yColumns - yColumns %
      8].reshape(-1, 8, 8)
2 yShuffle = virtualShuffle(xBlockNum * yBlockNum - 1, xBlockNum *
    yBlockNum - 1, seed, seedBlock)
3 yBlocksEncoded = encodeElements(dataToEncode, yBlocks, blockChanges,
      yShuffle, dataToEncodeLenght, mask, shift)
```

Listing 9.2: Python code for performing frequency domain steganography in an image

As observed, line 1 performs the task of partitioning the quantized values drawn from the image into distinct matrices of dimension $8 \times 8$. Following that, line 2 initiates the creation of an insertion list, which is constructed by employing the seed. This list will subsequently be utilized to embed the data at its respective positions. At last, line 3 consequently invokes the function responsible for executing the steganographic process. This function replaces a predefined number of bits in the selected coefficients of the quantized arrays that correspond to the indexes in the insertion list.

Frequency domain steganography presents a greater level of difficulty in data detection and removal compared to its spatial domain counterpart. The following points highlight its efficacy:

- **Privacy:** Similar to spatial steganography, frequency domain steganography enables encryption of the embedded data, thereby **maximizing data privacy**.
- **Robustness:** In this form of steganography, robustness significantly improves compared to spatial steganography. Accessing the data becomes **more complex**, and modifying the encoded data values is not as straightforward as applying filters to the image or altering specific pixels. Through the utilization of DCT and subsequent division into blocks, a substantial change in the image is required to modify quantized values, transcending the simpler approach of spatial steganography.
- **Undetectability:** As previously mentioned, the detectability is directly linked to the amount of data being inserted. However, this technique presents greater difficulty in conducting conclusive analysis, thereby **enhancing its resistance** against potential attackers and increasing its undetectability.

## 9.3 Video steganography

Video steganography has witnessed a surge in its utilization partially due to the sudden emergence of new methods for generating and sharing video content such as social networks and video sharing platforms.

Most of the variants of video steganography are derived from the "simple" principles of image steganography. By employing analogous techniques, the capacity for embedding data within a video is substantially augmented, while concurrently making use of a lower proportion of the carrier data for payload insertion. It is important to mention that in the case of video steganography in **Steg'o**, each frame is encoded with its own seed, as individual images, and the "primordial seed" is encoded in the audio of the file, using an LSB technique (as explained in section 9.1, the primordial seed is just a seed used to recall the exact position of the main seed in the encoded data).

In **Steg'o**, two different strategies can be found when performing steganography in a video: lossless and lossy steganography.

### 9.3.1 Lossless steganography

Lossless steganography involves directly modifying the pixel RGB values of video frames. This process closely resembles spatial domain image steganography, but it presents an additional challenge regarding the remarkable increase of the video size compared to the original. This size increase is due to the requirement of a lossless encoding of the carrier video, which should ensure the preservation of the pixel values during the encoding process.

Lossless encoding in videos is not commonly employed because it results in an output media that is impractically large for most purposes. To illustrate this, consider a **4-second** video consisting of **43 frames** in the **.mp4** format (a **lossy** format), with an original size of 451kB. If this video is recompressed using the lossless *FFV1* codec, its size expands up to **7MB**. This represents a **size increase** of nearly **1400%**, rendering it unfeasible for general usage on longer videos.

On the other hand, employing lossless video formats for encoding payloads offers the advantages of speed and efficiency, particularly when working with short videos. In such cases, where the carrier video is not lengthy, this approach can be a more viable solution, as it avoids introducing losses and enables encoding of substantial payload sizes.

### 9.3.2 Steganography in compressed lossy formats

Performing steganography in lossy video formats entails greater complexity compared to its application on images. As exposed in Section 9.2.1, this form of steganography involves modifying the RGB values of an image to embed data while minimizing perceptible artifacts.

In the video realm, the process is relatively similar, although much more arduous. Video codecs such as High Efficient Video Coding (HEVC) (also called *H.265*) employ a large variety of techniques to compress video media while minimizing data loss from the perceptual perspective. Next, a brief overview of two of the main techniques employed for video encoding is provided.

**Inter-Frame Coding overview:**

Among these techniques, we can stand out the **Inter-Frame Coding**, where "key" frames, also known as **intra-frames** or **I-frames**, are selected and completely encoded using a lossy algorithm like JPEG while the remaining frames are encoded as motion vectors, attempting to accurately predict the motion within each frame relative to the "key" frame or frames. There are two types of "non-key" frames, the so called **B** and **P frames**.

Each **P-frame** is generated through forward prediction, utilizing a preceding I or **P-frame**. **B-frame** creation also employs backward prediction, which combined with forward prediction results in a method called **"bidirectional interpolation prediction"**. In this kind of interpolation, the forward prediction is determined first, followed by the backward prediction. Subsequently, the encoder determines the weighting ratio to calculate the average of both predictions. However, due to the significant number of errors introduced by **B-frames** (larger than those in **I** and **P frames**), they are not employed for subsequent predictions in the video frames.

The set of consecutive frames positioned between two keyframes is commonly referred to as a Group of Pictures (GOP). Typically, the number of frames per group ranges from **nine to sixteen** or even more, although this parameter can be adjusted by the user during the encoding procedure.

Consequently, in the present context, modifying each frame individually is unfeasible, since most of the frames lack explicit pixel information. Therefore, the most viable solution is to modify the frames that are fully encoded, i.e. the **intra-frames**. Although these frames are subject to *lossy* compression, frequency domain steganographic techniques can be performed in them, as previously done in images, to embed data in *lossy* formats.

However, in this case, this specific approach has not been considered for the implementation of **Steg'o** application. Making use of this technique would involve modifying the *H.265* standard itself to introduce a backdoor in the quantization process, enabling data insertion. Given the complexity and time-consuming nature of this process, coupled with the requirement of modifying an established and extensively tested standard, we have opted to disregard this path.

Instead, an alternative approach that aligns more closely with the basis of the spatial domain steganography has been followed. This technique, inspired by the work of **Dr. Galiano** in [36], involves modifying the **luminance** and **chrominance** values of **intra-frames** in a video before the encoding takes place.

This process bears a strong resemblance to the previously explained method for spatial steganography involving RGB values, but with an important distinction: the modified values pertain to the **luminance** (**Y**) as well as the **blue** and **red chrominances** (**Cb** and **Cr**) of the frames, rather than color values per se. As explained before in page 51, the human brain

exhibits lower sensitivity to chrominance values, particularly in the red component. This understanding can be exploited to execute chroma subsampling in both images and videos, removing a large number of chrominance values with minimal discernible impact.

**Chroma subsampling overview:**

A prevalent form of chroma subsampling is exemplified by the **4:2:0** scheme, where the **luminance** component is fully sampled, while the **chrominances** are sampled at **half the resolution**. Consequently, for every four **luminance** samples, **two chrominance** samples are selected, both horizontally and vertically. Thus, the number of values for the chrominance components will be **four times smaller** than in the luminance one after the subsampling.

Something similar happens in the case of **4:2:2 subsampling**, with the distinction that two samples of **Cr** and **Cb** chrominances are acquired for every four samples of luminance (the reduction factor is hence two instead of four). This results in halving the chrominance resolution horizontally, while preserving the full resolution vertically. Thus, the number of values for the chrominance components will be two times smaller than in the luminance one. In 4:4:4 chrominance subsampling, the entire image (both luminance and chrominances) is sampled in its entirety. Figure 9.8 shows the results of performing this three different chroma subsampling procedures in a pixel grid.

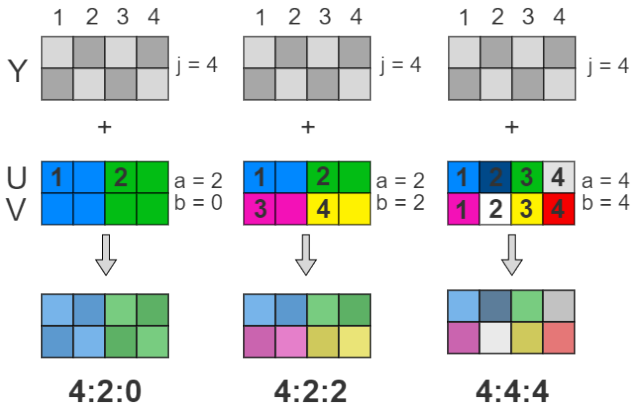



Figure 9.8: Different schemes of chroma subsampling with different level of values reduction

The level of chroma subsampling should be taken as a relevant factor, as it directly impacts on the amount of data lost during the encoding process. A lower subsampling rate, implying a reduced capacity for sample collection, results in a greater loss of data due to compression artifacts introduced by the encoder. In contrast to previous techniques that offered lossless data insertion, the current approach entails the risk of irreversible data loss, thus making it impossible to retrieve the payload information.

To address this challenge, several measures can be implemented:

- Firstly, data is inserted into the *YCbCr* components of the video rather than into the RGB components. This choice is motivated by the observation that color RGB values experience more substantial losses compared to **luminance** and **chrominance** values during the compression procedure.
- Secondly, data insertion occurs exclusively in the **intra-frames**. Naturally, this has an impact on the data capacity available for insertion, which depends on the GOP size. The larger the GOP size, the lower the capacity to conceal data, but the smaller the file size.
- Thirdly, data insertion employs a **shifting-based technique**. This shift is utilized to mitigate data loss during encoding, as the **most significant bits** of a byte are **less susceptible** to be removed during compression. In the opposite site, modifying the most significant bits can introduce visual distortions that can be perceptible for other users. Although the shift value can be user-defined, empirical testing indicates that a shift of **4 bits for luminance and 5 bits for chrominance** is the **minimum acceptable** choice for minimizing data loss.
- Lastly, **redundancy techniques** can be applied to data insertion. The payload can be replicated in the insertion procedure, therefore increasing the probability of successful data recovery. In conjunction with a **data recovery technique** based on loss percentages derived from numerous tests, this redundancy-based approach **significantly enhances** the likelihood of recovering the inserted data.

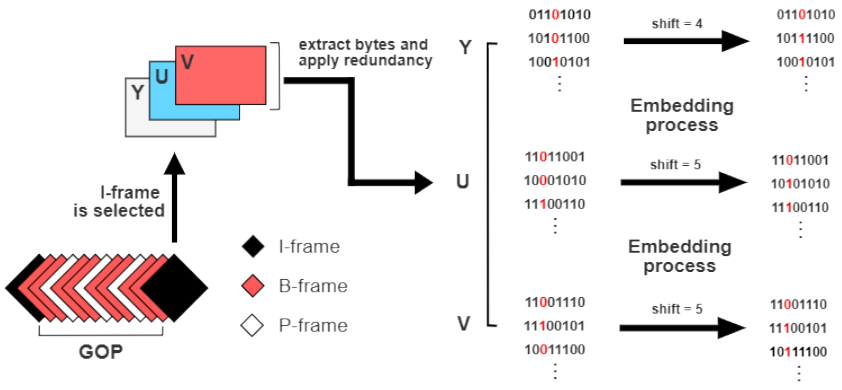



Figure 9.9: Sequence of steps for ensuring data recovery in the considered approach

Without the combined use of these methodologies, the data loss rate stands at approximately **80%**. However, by employing the aforementioned preventive approaches, it is possible to significantly diminish the data loss to approximately **3%** or even less, contingent upon the specific video being encoded.

# Chapter 10

# Tests

This chapter focuses on the **analysis** of **steganographic algorithms** implemented at the functional level.

In implementations such as steganography in HEVC videos, it is crucial to have a comprehensive understanding of how different video export and encoding configurations are interconnected and how they affect the potential recovery of the inserted data depending on the considered parameters. The interrelationships between encoding variables become particularly important for optimal decision-making to minimize losses and file sizes while maximizing the possible data insertion, considering that this steganographic functionality in **Steg'o** does not allow for lossless data recovery due to video compression.

## 10.1 HEVC encoding parameters testing

A total of **650 tests** were conducted for this analysis, making use of **5 video files** of **different resolutions** and **formats**. These tests measured the **execution time** of data encoding and decoding processes in HEVC video steganographic procedures, as well as the **percentage of data loss with** and **without redundancy**, and other significant variables. The collected data was then saved in an *Excel* file for detailed analysis.

Several relevant configurations were evaluated for HEVC video encoding:

1. Firstly, the chrominance subsampling, specifically **yuv420**, **yuv422**, or **yuv444**, was modified. This variable determines the type of subsampling used in video encoding, which has a substantial impact on both the video's size and its visual quality.
2. Additionally, the Constant Rate Factor (CRF) was tested, as it determines the visual quality value of the exported video. This variable values range from **0 to 51** and can significantly affect the **video's size**.
3. The Quantization Parameter (QP) was another variable inspected in the tests. This parameter determines the level of **spatial detail** retained in the final video. Lower values

of QP indicate **higher video quality** at the expense of **larger file size**.

4. Lastly, the GOP variable controls the size of the frame groups, as explained in page 57. This variable greatly **influences** the **size of the encoded file**, as a lower GOP value results in more intra frames being encoded in the video, leading to a significant increase in the number of bytes used.

| Variable | Value | Data loss rate (%) | | File size change (%) | |
|---|---|---|---|---|---|
| | | Median | Std | Median | Std |
| CRF | 0 | 0.34 | 5.61 | 145.30 | 147.46 |
| | 4 | 1.69 | 5.51 | 108.65 | 108.00 |
| | 8 | 5.84 | 11.30 | 75.50 | 77.65 |
| QP | 0 | 1.71 | 8.63 | 87.05 | 119.87 |
| | 4 | 2.02 | 8.85 | 86.03 | 104.08 |
| | 8 | 3.35 | 9.03 | 82.10 | 99.97 |
| GOP | 3 | 2.04 | 9.61 | 95.95 | 116.22 |
| | 6 | 1.32 | 7.41 | 83.20 | 123.23 |
| Format | yuv420p | 0.66 | 4.83 | 87.95 | 114.31 |
| | yuv422p | 1.10 | 5.61 | 96.85 | 126.20 |
| | yuv444p | 5.71 | 11.50 | 87.05 | 118.41 |

Table 10.1: Statistic measures based on HEVC steganography testing

Based on prior experimental findings, it was determined that in order to attain a suitably diminished data loss, certain trade-offs, such as compromising QP and CRF, must be made, consequently leading to an augmented file size. As seen in table 10.1, when the CRF surpasses **4**, the data losses increase very fast, leading to a potential doubling or tripling of losses in certain cases.

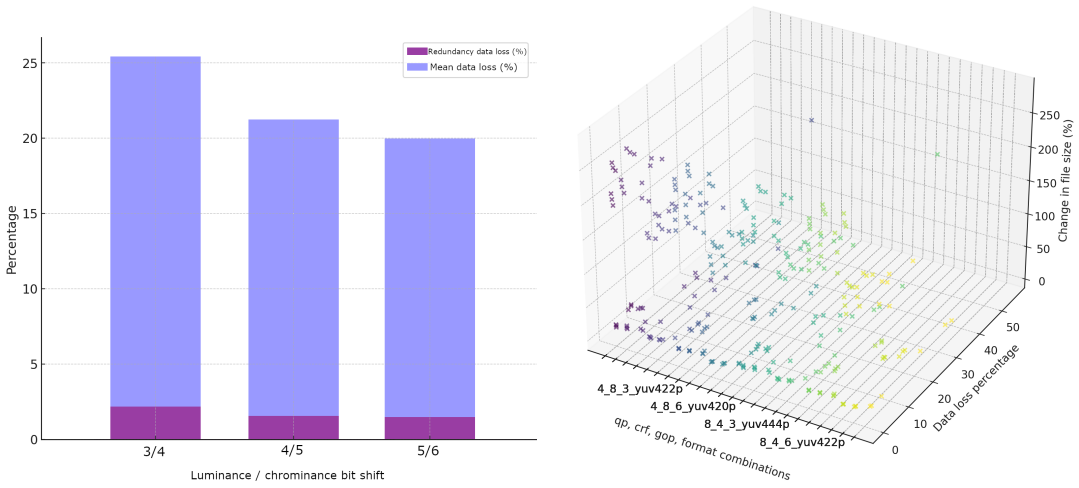
During these experiments, a counter-intuitive observation emerged: **elevating** the **chrominance subsampling** rate to preserve more chrominance values (**yuv422** and **yuv444**) resulted in a significant **data loss increase**. This finding contradicts all expectations, as theory suggests that reducing the amount of removed coefficients during the subsampling should retain more chrominance values in the final image, thereby enhancing data recovery. The main hypothesis for this fact is that the encoding process amplifies overall losses in **yuv444** and **yuv422** encoded videos to compensate for the file size increase associated with these encoding formats.

Regarding the Group of Pictures (GOP) size, two different alternatives were examined: **3** and **6** frames. These values are relatively small compared to the standard GOP range of **16 to 21 frames** for HEVC video (see Section 9.3.2). Although the reduction in GOP value was optional, it was aimed at holding larger data insertions within the video files. The GOP size should not significantly impact the data losses; however, these tests revealed that a GOP of **3** is excessively **small**, adversely affecting both the final **file size** and **data loss**.

Analyzing the data in table 10.1, and given the range of the **file size change**, the observed **standard deviations** in this context (which are around **77,6** to **147,4** for different groupings) appear to be reasonable. The standard deviations represent a spread of values, and given the high volatility of change in file size values, the calculated standard deviations don't seem unusually high.

In the context of data loss rate, the standard deviation ranges from **4,83** to **11,5**. This difference in losses is due to different **file configurations** and **videos**. For a video where the modified frames contain a lot of different colors and **shapes**, the data loss will be significantly higher than for a smoother video where the colors do not change much. This is because of the optimizations used in the encoding process to remove irrelevant information and reduce file size, which can lead to some colors not being represented as faithfully.

The analysis of data loss also contemplates the shift of the inserted bits with respect to the less significant bit in each byte, a crucial factor to be taken into consideration. In this research, the conducted tests considered various bit shifts, namely **4, 5, and 6 bits** for **chrominance**, and **3, 4, and 5 bits** for **luminance**. For instance, when the bit shift is set to **5** and the original byte is **00000000**, the modified bit will be the sixth bit from the right, as the first bit is considered the number 0. Consequently, the resulting byte would be **00100000**.




(a) Graphical measure of bit shift relation with data losses

(b) 3D representation of encoding variables and data losses/file sizes

This information holds significant importance as higher bit shifts make the steganography process more discernible to potential observers.

Conversely, excessively deep shifting may render most of the data recovery unattainable, leading to substantial **data losses**. Figure 10.1a illustrates the data loss percentages encountered when employing various levels of bit shift in a **conventional approach** and, employing data recovery techniques based on **redundancy**.

The results indicate that the incorporation of **redundancy** significantly **mitigates data loss**, reducing it from an average of **26%** to a **mere 2%** in the scenario involving **3 bits of bit depth** for **luminance** and **4 bits** for **chrominance**. This represents a **reduction above 1000%** in terms of data loss.

Figure 10.1b presents the three-dimensional graphical representation of the interplay among video encoding variables (QP, CRF, GOP and chrominance subsampling), in relation to data loss percentage and the corresponding increase in file size.

For a comprehensive exploration of this dataset, **three interactive plots** are available as follows:

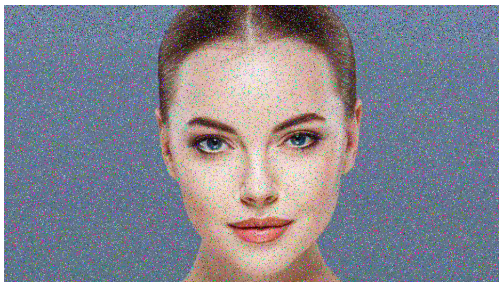
- To observe the **three-dimensional** representation illustrating the effects of data loss and file size alterations, employing a **4-bit depth for chrominances** and a **3-bit depth for luminance**, please click here.
- To observe the **three-dimensional** representation illustrating the effects of data loss and file size alterations, employing a **5-bit depth for chrominances** and a **4-bit depth for luminance**, please click here.
- To observe the **three-dimensional** representation illustrating the effects of data loss and file size alterations, employing a **6-bit depth for chrominances** and a **5-bit depth for luminance**, please click here.

The concept of data loss percentages might seem highly abstract at first glance; therefore, an illustrative visual representation has been incorporated below, presenting the result of inserting an image within a HEVC video by employing different **levels of bit shifts**.

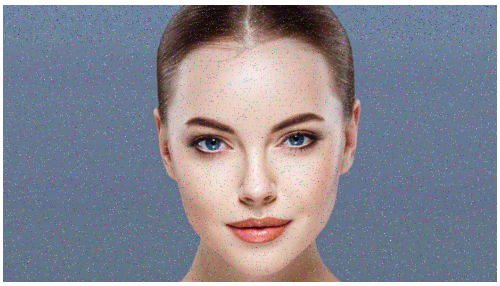
The impact of the insertion process according to different shift levels significantly influences the amount of **retrievable data**. This phenomenon can be visualized more effectively through graphical representations. In this sense, the **differences in color** of some pixels in figure 10.2 represent instances of decoding failures, indicating errors introduced during the process of recompression of the **Carrier** video, which inevitably affects the amount of recoverable data.

As seen in 10.2, by inserting the information in bits farther away to the most significant bit of each byte, the information retrieved after the re-encoding process will have a greater amount of random colored pixels, which implies that a bigger quantity of information is lost.

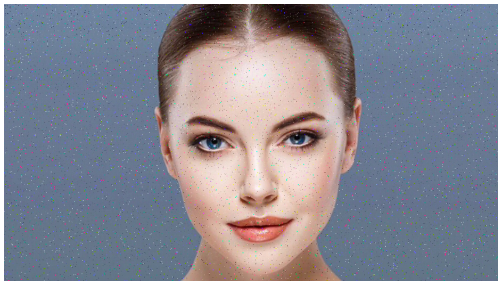
The encoding process for the frames shown in Figure 10.2 has been performed with the following configuration: **GOP:** 6 frames, **CRF:** 4, **QP:** 8, **Subsampling format:** yuv420p.



(a) 2 bit shift encoding for chrominance (1 for luminance)



(b) 4 bit shift encoding for chrominance (3 for luminance)



(c) 6 bit shift encoding for chrominance (5 for luminance)

Figure 10.2: HEVC image encoding performed at different bit levels

# Chapter 11
# Implementation

## 11.1 Sprint 1: Project Setup

The first sprint began with a thorough investigation to identify existing steganography methods suitable for different types of media, as well as those that could be applied to this specific project. It was determined that spatial and frequency steganography were the most viable techniques capable of enabling the required volume of data insertion necessary for the application to be practically useful, as these methods enable the incorporation of a significant amount of data into carriers with relatively small dimensions.

Upon the study completion, the subsequent course of action involved the creation of two *Github* repositories dedicated to storing the project's codebase for the *Front* and *Back* ends. These repositories also serve the purpose of maintaining a version control system to track the evolution of the project during its development phase.

### 11.1.1 Frontend Setup

The next phase involved the development of the basic *Flutter* framework. By default, *Flutter* generates a sample application that serves as a integer counter, with a button that modifies the counter each time it's tapped.

Figure 11.1a illustrates the folder structure established by *Flutter*, which determines the organization of the project files.

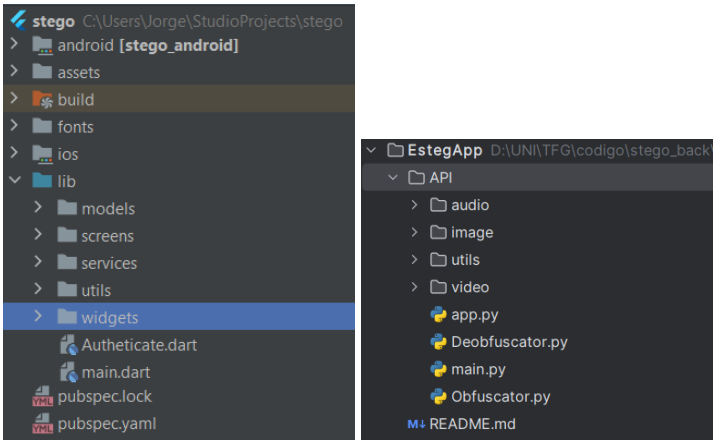
The most important components within this structure encompass:

- **android:** Folder containing the code for the Android application.
- **build:** Folder consisting of Flutter dynamics and the translation of Dart code to Kotlin for Android and Swift for iOS.
- **ios:** Folder that contains the application code for iOS. Modifications to these files are necessary only for specific configurations on this operating system, as Flutter does not

offer native support.

- **lib:** Folder that serves as the central directory for the project, housing the entire file structure used in its development.
- **pubspec.yaml:** This file serves as the backbone of the Flutter framework, as it allows for comprehensive configuration of the application's behavior. It eases the addition of external media files and third-party packages employed during development.

Now, it's time to create a *Firebase* instance and connect it with the app. To accomplish this, a *Firebase* project needs to be created on the *Firebase* web service, specifying the required components. In this particular case, the utilized *Firebase* components include *Cloud Storage*, *Cloud Firestore* and *Firebase Authentication*. Following the completion of these configurations, the following packages must be added to pubspec.yaml: firebase_core [37], firebase_auth [38], cloud_storage [39], cloud_firestore [40].




(a) Android Studio file structure (b) Pycharm project structure

Figure 11.1: Project structure in the different IDEs

### 11.1.2 Backend Setup

After the completion of the *frontend* and Firebase configuration, the next step involves creating a PyCharm project for *backend* development. To accomplish this, a new project is created with a hierarchical structure of folders.

Within this structure, a main file (*main.py*) is established to execute the API, while subfolders contain code executed by the available application services. The file structure can be observed in Figure 11.1b.

Once this is done, it is necessary to access the main.py file to enable the server-side *Firebase* configuration. To achieve this, the *Firebase* project keys must be exported via its web

service and securely stored in the server file system. By safeguarding these keys from extraction or unauthorized modifications, real-time access to *Firebase* service data is facilitated. Such access enables seamless file uploads and downloads for *frontend* and *backend* communication. Integrating the *Firebase* keys into the code merely requires the addition of a few lines in the *main.py* file specifying their location within the file system.

## 11.2 Sprint 2: Mockups, Firebase and Image steganography

### 11.2.1 Mockups creation

In order to ensure the development of coherent and realistic mockups, it is essential to consider the user requirements and proposed functionalities of the app. Thus, the functionalities were categorized into four primary groups:

- The first group encompasses **chat mockups**, which visually represent the screens associated with the chat functionalities of the application.
- The second group consists of **configuration mockups**, responsible for visually representing the screens related to the application's settings.
- The third group comprises **encoding/decoding mockups**, which visually depict the screens related to the application's steganographic functionalities.
- Lastly, the fourth group encompasses **analysis mockups**, which visually represent the screens associated with the application's steganographic analysis functionalities.

The mockups were developed using *Balsamiq Wireframes*(Figure 11.2).

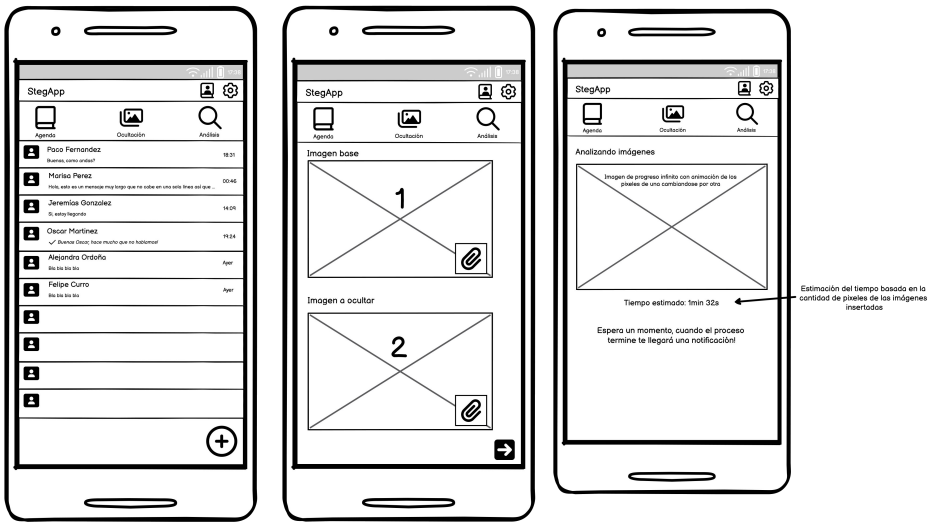



Figure 11.2: Examples of wireframes created at the start of the project

### 11.2.2 Firebase data relations

Once the initial mockups of the application's visual design and core functionalities are created, the next step involves designing the relationships between collections and documents within *Firebase Firestore* for the project.

In *Firebase*, data is stored in collections, which are essentially lists of documents representing objects with distinct values for their attributes. For instance, the **"users"** collection may consist of three documents identified by the *IDs* **1**, **2**, and **3**, each containing attributes such as **"first name"**, **"last name"**, and **"phone"**. In addition, documents can be interconnected through references, that essentially act as **foreign keys** in relational databases. These references establish connections to other documents or collections located elsewhere in the database. As an illustration, a user document within the **"users"** collection might contain a reference to a document in the **"chats"** collection, representing a chat created by that particular user.

Following these basic principles, an initial model for the database to be used in **Steg'o** was designed. The graphical representation of this model can be found in page 42, showcasing various collections such as **"users"**, **"chat"**, **"request"**, and **"message"**.

**Steg'o** users possess the ability to create chats, and upon creation, a reference to these chats is added to their respective user document. Moreover, each chat has within it a collection of **"message"** documents, which represent the different types of items exchanged between users in a specific chat.

Another essential collection in this model is **"request"**, which contains chat requests initiated by users. Once created (accepted by the recipient), these requests are referenced in the documents of both the requester and the receiver users involved in the exchange.

### 11.2.3 Image steganography: first prototype

In this sprint, development of a basic prototype for steganographic operations in images in the spatial domain was started. As explained in section 9.2, this approach involves replacing RGB values in an image to encode data within it.

The initial iteration of the algorithm focused on testing the feasibility of performing these modifications using the designated image manipulation libraries. As a result, the algorithm operated at a basic level, and the execution speed of this initial iteration was significantly poor compared to subsequent versions, with a performance lag of up to **50 times slower**. As the number of pixels in an image increases, the execution time grows exponentially when iterating through each RGB component. Although the efficiency was not entirely unsatisfactory, it fell short of desired expectations. Furthermore, this version **does not incorporate a seeding system**, resulting in a sequential modification of the pixels.

## 11.3 Sprint 3-6: Frequency steganography, API and screens

### 11.3.1 Frequency domain image steganography

In the subsequent phases, the research and development labours concerning frequency domain steganography were initiated. As explained in subsection 9.2.2, this particular technique involves the manipulation of frequency domain values of image blocks, making the detection and protection against potential attacks much harder, while also increasing the complexity of its implementation.

The initial stage of this process encompassed an exploration of potential libraries or freely available code capable of encoding and decoding JPEG images. This necessity arises from the fact that altering the quantized matrices of an image needs intervention during the image encoding process, thereby requiring low-level access to specific procedures executed during the JPEG encoding.

During the first iteration, it was decided to utilize the implementation by Github user *fangwei123456* [41] for encoding JPEG images. This code was modified to acquire a more profound and comprehensive understanding of the encoding process, enabling precise modifications at appropriate points to persist in the final image. Once the functionality of this code was comprehended, steganographic encoding capabilities were incorporated within the iterative loop responsible for processing the image in blocks and performing the DCT.

For the JPEG decoding process in this iteration, the code provided by *Yasoob Khalid* at [42] was employed. By making some changes at specific locations, access to the decoding system was obtained, allowing for data extraction as an archaic steganographic decoding process.

The code employed in these sprints was mainly utilized to understand the inner workings of frequency domain steganography, and was not followed upon in later sprints. Instead, a completely new implementation was used, involving the use of *jpegio* (subsection 4.3.8).

It also should be noted that the encoding and decoding methods employed in this sprint for JPEG implementation are basic and lack customization. Ideally, an external JPEG encoder optimized for this task would be preferable. In any way, modifications of this nature should not be expected to yield comparable efficiency to that of a genuine JPEG encoder.

### 11.3.2 API development

Simultaneously, while working on frequency domain steganography, the development of an API that will establish the connection between the frontend and backend components was initiated. *Flask*, a *Python* library specializing in creating and managing REST APIs, was utilized for this purpose (for more info about *Flask*, refer to page 10).

To configure the endpoints, several functions were implemented, each with different input

parameters. These functions receive data from the mobile app and determine the appropriate action to execute based on the context. For instance, if an HTTP request with the destination **"/obfuscate"** is sent, the code responsible for encoding the data sent into another format will be executed, and the corresponding codec will be initiated (for more info about codecs, refer to page 41). The remaining parameters, such as the input data type or the data type to encode, are obtained from the HTTP request body. Reader can look up Figure 8.2 for a visual representation of this process.

After considering all possible scenarios for encoding and decoding, the definition of the required endpoints are complete, making the API fully functional.

### 11.3.3 Screens creation

During the implementation of the front-end using the *Flutter* framework, a basic user registration and login screen were developed, incorporating email and name as authentication mechanisms. This approach is supported by *Firebase Auth*, enabling direct integration with the employed database. Such integration simplifies the login and logout processes, allows for user status tracking, and providing access to other user-related data on the client side.

Once the initial screens were implemented, a main screen was designed to accommodate various application functionalities. Users can navigate through this screen to the different options by using a drop-down floating menu.

The first section to be implemented was the chat section, drawing inspiration from existing applications such as *WhatsApp* [43], *Telegram* [44], and *Discord* [45]. A list with the active chats for the user was created, displaying **user profile pictures**, **user names**, the **most recent message sent**, and the **timestamp** of the message. To enhance the user experience, modifications were made to the chat list display. When a new message is received, a **notification bubble** is generated alongside the corresponding chat, indicating the number of unread messages. Additionally, the chat associated with a received message is **automatically displaced** to the top of the list, helping the user to locate current conversations.

Following the successful implementation of chat listing functionality, development began on the individual chat screen. This screen presents messages sent between two users, arranged in chronological order from most recent to least recent, emulating the behavior of other popular chat applications. In this user interface, the message list functions as a paginated list, enabling the initial loading of a predetermined number of messages. As the user scrolls up the list, additional messages are dynamically loaded, ensuring a seamless user experience without any disruptions such as stuttering, lag, or unnecessary updates.

Each message is treated as an **independent entity**, facilitating the implementation of features like message selection. Messages can be enclosed within a colored area to enable **bulk operations** such as **copying** or **deleting** multiple messages simultaneously. The profile

picture and name of the other user involved in the chat are displayed at the top of the screen, allowing for easy identification of the ongoing conversation. Additionally, a **back arrow** is provided to enable users to navigate back to the list of chats.

Messages sent on different days are visually differentiated by the inclusion of date indicators positioned between them. These indicators provide a clear reference to the chronological order of the messages. Please refer to Figure 11.3a for an illustration of these features.

Other screens developed during these sprints include the **encoding/decoding screen** and the **analysis screen**. In an initial iteration, a user interface featuring selectable buttons was introduced to facilitate the selection between various media encoding and decoding options. However, this approach was quickly discarded in favor of a more integrated solution, where all encoding and decoding functionalities were consolidated into two distinct screens.

The encoding screen offers users the option to select between images/video and text using a top-mounted slider. When images are selected, users can choose an image to embed within another image or video. The encoding process also allows for the configuration of various parameters, such as the number of bits to be inserted, their shift, the type of steganography to be employed, or if the data will be encrypted, among others.

After selecting the necessary elements, users can proceed with the encoding process or view an estimate of the data insertion capacity available for the selected values, represented as a percentage.

Estimating the data insertion capacity for a given image is a relatively straightforward process.

- In the context of spatial steganography, the method begins by first calculating the total number of RGB components within the image. This count is then multiplied by the user-specified number of bits designated for insertion.

  Consequently, this computation yields the total number of bits available for insertion. To convert this figure into bytes, a simple division by 8 is performed, providing the byte capacity for embedding data within the image.

- However, in frequency domain steganography, the process becomes somewhat more intricate. Initial calculations involve dividing the image size by 8 to determine the number of MCU required for JPEG compression.

  Subsequently, a series of calculations follows, beginning with the multiplication of a randomly selected value within the range defined by the minimum and maximum number of elements designated for encoding within each MCU. This value is then multiplied by the number of bits allocated for insertion for each respective element. As in spatial steganography, the result must be divided by 8 to show the encoding capacity in bytes.

Please refer to Figure 11.3b for visual representation of this screen. The text insertion screen provides a simpler interface. Users have the option to select a text file from their device or manually enter the text to be encoded into a text box. Once the text input is finalized, users can customize encoding parameters and receive an estimate of the data insertion capacity achievable with the chosen values.

Regarding the image analysis screen, a basic implementation was created, enabling users to select two images from their device for pixel-value comparison. Further exploration and enhancement of this functionality were planned for subsequent development sprints.

## 11.4 Sprint 7: Screens changes and video steganography

### 11.4.1 Screens changes

During this sprint, a loading screen was implemented for **Steg'o** in order to establish a seamless transition between the offline and online functionalities of the application. This loading screen serves as visual feedback during encoding, decoding, and data analysis processes. It incorporates an animated logo, providing users with reassurance that the application is actively performing an online operation that may require some time.

Another notable modification was made to the analysis screen. Previously, the visual appearance of this screen was not well-defined as its implementation was still under consideration. In this sprint, a *heatmap*-style representation was selected, presenting image modifications through a range of colors. For reference, see Figure 11.3f.

When an image is subjected to analysis, it is divided into two layers: the **original image layer** and the **modification layer**. The modification layer consists of pixels with predefined values corresponding to specific ranges of pixel modifications. For instance, if a pixel value varies (with respect to the original value) within the range of **1 to 8 bits** (where each pixel is represented as a **24-bit** value with **8 bits** allocated to each component), it will be depicted as **green** within the *heatmap* layer. Likewise, if the modifications fall within the range of **9 to 16 bits**, the color representation will be **orange**. Conversely, modifications falling within the range of **17 to 24 bits** will be depicted as **red**.

By default, pixels that have not undergone any modifications are represented as transparent. This visual representation enables users to easily discern and comprehend the extent of modifications applied to an image by comparing it with an unaltered reference image. By examining the positions of these modified pixels and other distinctive characteristics, experienced users can determine whether the image has been subjected to **steganography** or, alternatively, a form of **recompression**.

In addition to this feature, the sprint introduced the capability to customize the range of modified bits. For example, users can adjust the range associated with the green color repre-

sentation from **1 to 8** modified **bits** to **1 to 4** modified **bits** using a slider located at the bottom of the image. Furthermore, users now have the option to customize the colors representing each range, providing enhanced flexibility for personalization and improved visualization. For instance, the user can choose to assign blue as the color representation for modifications falling within the range of **17 to 24 bits**, rather than the default red.

These enhancements significantly increases the analysis capabilities of the application and facilitate the identification of patterns or other distinct characteristics associated with steganographic algorithms.

### 11.4.2 Lossless video steganography

Following the development of these new screens, research began on steganographic algorithms in video formats. Initially, this functionality was considered optional, as its feasibility within the given timeframe was uncertain during the planning phase. The divergence in the analysis arose due to the intricate nature of estimating such task without the beginning of its development.

Upon completion of the investigation into video steganographic algorithms, a decision was made to practically implement a basic algorithm for embedding data into videos utilizing spatial domain steganography. To accomplish this, it was imperative to export the video in a lossless format. Consequently, the *FFV1* codec with *AVI* format was chosen for this purpose. By introducing several modifications to the code used for spatial steganography in images, an algorithm capable of modifying specific pixels within particular frames of a video to embed data was successfully implemented.

In this initial iteration, the seeds employed to randomize the positioning of frames and pixels were consistently encoded in a sequential manner within the first frame of the video. Although this may not be the optimal approach, it proved sufficiently effective for basic testing and performance evaluation.

Once the encoding process was completed, the subsequent step involved implementing the decoding algorithm. This algorithm extracted the seeds, subsequently utilized to locate the embedded data within the video frames and pixels. To extract the data, a properly modified version of the image spatial steganography algorithm was employed.

## 11.5 Sprint 8-10: Notifications, numpy and HEVC

### 11.5.1 Pushup notifications

The initial objective of this sprint involved configuring push notifications to enable the transmission of notifications to **Steg'o** users. This functionality allows users to receive notifications

upon completion of specific operations or when they receive messages/requests from other users. To achieve this, the *Firebase Spark* plan was activated, enabling remote operations through the utilization of *Firebase Functions* triggered by specific events within the managed applications. These operations encompass a range of actions such as sending notifications, executing customized code to modify the database, or performing more intricate tasks. For the purpose of this project, the focus was on implementing push notifications.

The first step involved configuring push notifications within the *Firebase web interface*. Then, a basic code snippet was developed and implemented to test the functionality. This code is executed whenever there is a change in a **"chat"** document, and subsequently sends a notification to the intended receiver/s if they are disconnected from the application. Two distinct use cases were considered for this functionality:

- Firstly, when the application is in the **foreground**, notifications should only be displayed if the user is not actively using the chat screen.
- Secondly, when the application is in the **background**, notifications should always be received by the user.

However, an implementation challenge arises in the second scenario. In the tested version of *Firebase*, incorporating the functionality of receiving push notifications while the application is in the background results in the creation of a **duplicate instance** of the application. This issue leads to significant memory consumption and causes performance issues, such as slow response times and lag across various screens within **Steg'o**. To mitigate these problems, the utilization of background push notifications has been temporarily disabled until *Google* resolves the bug responsible for the creation of duplicate application instances.

### 11.5.2 Numpy and algorithm optimization

After implementing **push notifications**, the next step involves enhancing the efficiency of the steganographic algorithms employed in the *backend*. To achieve this objective, a complete evaluation of the algorithms was conducted, resulting in substantial code revisions for both spatial and frequency steganography.

The updated code now employs libraries such as *numpy*, facilitating batch operations on arrays with a large amount of elements. This modification yields efficiency improvements ranging from a **tenfold** to **fiftyfold** decrease in runtime, depending on the specific image.

The benefits of these modifications are particularly pronounced for large images, as certain operations exhibit constant execution time regardless of array size. However, without the utilization of *numpy*, the computational speed experiences a significant slowdown when dealing with larger arrays.

During this evaluation, techniques like **bulk masking** were also employed to quickly apply a mask to all elements of a *numpy* array.

Therefore, bit substitution operations, previously executed individually by converting numbers into binary representation, eliminating specific bits, and performing subsequent substitutions, have now been condensed into a few basic **AND-OR operations**. This facilitates the instantaneous removal and replacement of bits across all array elements.

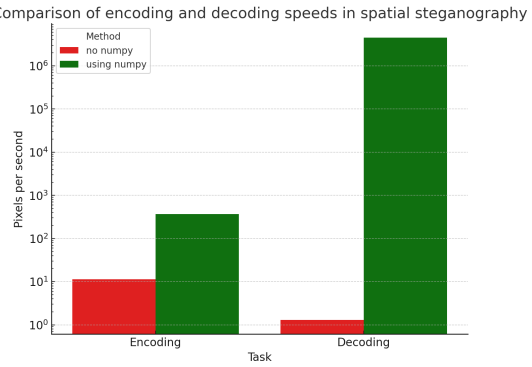



Figure 11.4: Before and after performance in spatial steganography

The discernible discrepancy in execution time between the two algorithm versions is visually depicted in graph 11.4, which presents the mean pixels/second processed from five carefully selected images.

In the case of frequency domain steganography, the complexity of the problem becomes more evident. In addition to employing the *numpy* library for enhanced execution efficiency, it was noticed that the main challenge was lying not in the steganographic operations themselves, but rather in the JPEG encoding process. Consequently, alternative approaches had to be explored, since the extensive execution times rendered steganographic operations on large images virtually impractical. During the course of this research, a *Python* library called *jpegio* [46] was located, which specializes in the manipulation of internal values within JPEG files. Initially, this library was overlooked due to limited documentation and relative obscurity.

To evaluate its runtime improvements and customization capabilities, tests were conducted using this library. The results concluded that it represents a **significant enhancement** over the previous solution (subsection 11.2.3). Following numerous tests and experiments, an initial version of frequency domain steganography for images utilizing *jpegio* was implemented. This version employs *jpegio* to read and export the array of quantized values from the image file to a *numpy* array, thereby facilitating convenient manipulation. Subsequently, the array is divided into **8x8 blocks** to enhance visualization and aid in debugging. While this division process may marginally slow down the overall procedure, its adoption was justified by the numerous benefits it offered to developers.

Once the division is completed, the quantized values are modified by using **vectorial** *numpy* operations. Afterwards, these matrices are reassembled and exported as a new image. Comparatively, this process exhibits significantly improved speed when compared to the previous version. In the various tests conducted, the speed increase was estimated to be approximately **25,000%**, which is a massive improvement.

This sprint's outcome is a completely **new codebase**, greatly enhanced in comparison to its previous iteration. The final result **bears no resemblance** to the initial implementation illustrated in page 69.

### 11.5.3 HEVC video steganography

Finally, an algorithm was created to perform steganographic operations on lossy videos, using the *H265 codec*. The rationale behind this decision, as illustrated in section 9.3, stems from the limitations of resulting video sizes for steganography in lossless videos. Consequently, the algorithm focuses on modifying the **luminance** and **chrominance** values of the **intra frames** of a video to embed data within specific pixels. The development of these operations drew inspiration from the research conducted by Dr. Galiano and his colleagues [36], where similar techniques were applied to HEVC videos by altering the **chrominance** values within **4x4 pixel matrices**.

One of the targets behind the development of this algorithm was driven by the interest surrounding its efficacy in the context of **data loss**. As the algorithm is implemented on a **lossy** video codec, a certain amount of data is inevitably lost during the encoding process. The student was particularly interested in determining the **compression** and **encoding parameters** that had the greatest impact on observed data losses. To address this, an extensive investigation was undertaken, and the outcomes of which were detailed in section 10.1.

## 11.6 Sprint 10-12: Code refactoring and history

### 11.6.1 Code refactoring

This sprint began with a complete refactoring embracing both *frontend* and *backend* codebases, aimed at enhancing maintainability and code readability.

In **Steg'o** *frontend*, the decision was made to implement a state management solution utilizing the Provider package. This particular solution empowers developers with the ability to effectively compartmentalize logic and interface components by employing observers, which, in turn, signal application screens to refresh as necessary, facilitating the presentation of updated data. For an in-depth exploration of the Provider pattern, refer to subsection 8.1.2.

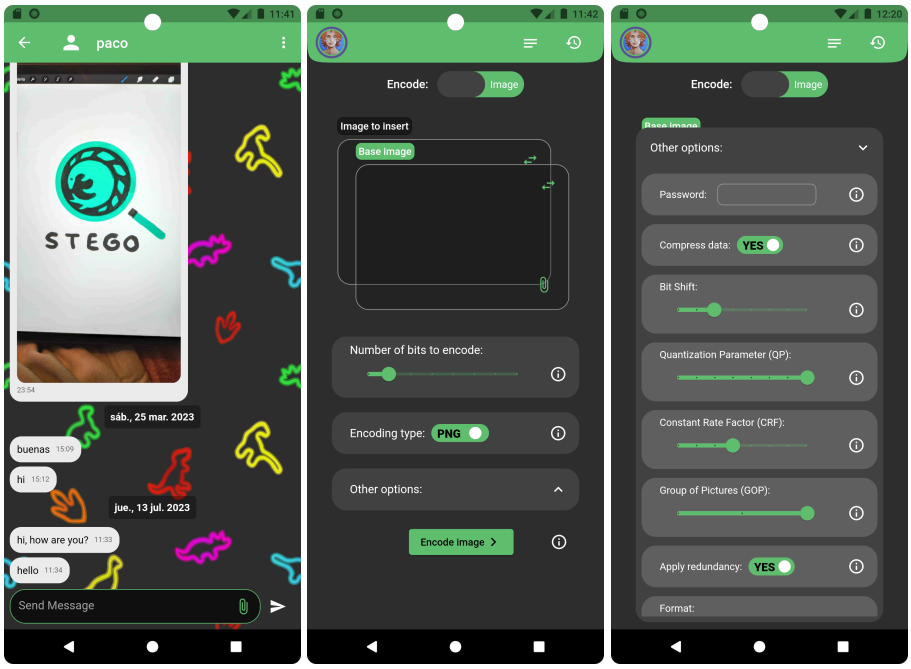
On the backend, the work was partitioned into distinct classes, which were named as "**codecs**". This technique not only allows for a *finer-grained* approach to task allocation but also encourages the sharing of common functionalities across various components. Moreover, it represents an optimal approach when employed in an API supporting concurrent users, as each object maintains its own **memory space**, mitigating data conflicts with other operations.

### 11.6.2 Encoding/decoding history

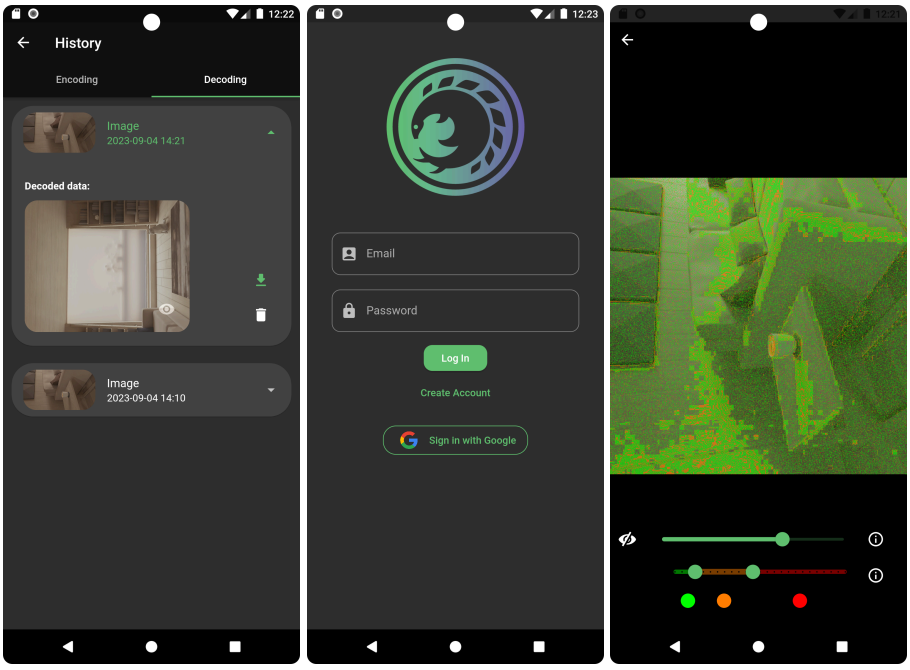
The next task was creating a new screen within **Steg'o**, designed to facilitate user access to *encoding* and *decoding* operations performed lately. This screen serves the purpose of presenting users with detailed information regarding these operations, including configuration parameters and data type employed in the encoding process. Furthermore, users can **download** the encoded/decoded data to their personal devices and even **delete** it from the database.

To facilitate these functionalities, two distinct collections, namely "**enc_history**" and "**dec_history**", were created within Firebase Storage. These collections are intended to **encapsulate** the encoding and decoding activities conducted by individual users. Within these collections reside documents containing essential data, including **encoding/decoding settings**, the **file path** for the data subject to encoding, and the resulting output of the encoding/decoding process.

When an encoding/decoding operation is initiated, an HTTP request is transmitted to the *backend* API. This requests holds the configuration parameters and the data path marked for processing. Next, the API undertakes a validation procedure to ensure the integrity of the configuration and proceeds to retrieve the data, which is then stored as a variable within the *Obfuscator/Deobfuscator* object. Following this, the data undergoes processing. In the event of a **successful** processing **operation**, the resultant data is uploaded to *Firebase Storage*, and the associated history document in *Firestore* is updated to include the path leading to the processed data. Upon confirmation that the operation has ended adequately, a notification is dispatched to the user through *Firebase Push Notification* system.




(a) Chat screen (b) Encoding screen (c) Steganography config




(d) Decoding history screen (e) Login screen (f) Heatmap image analysis

Figure 11.3: **Steg'o** screenshots

Chapter 12

# Conclusións

Upon the project finalization, a complete analysis can be conducted to evaluate the achieved objectives and identify potential paths for future enhancements. The key conclusions can be summarized as follows:

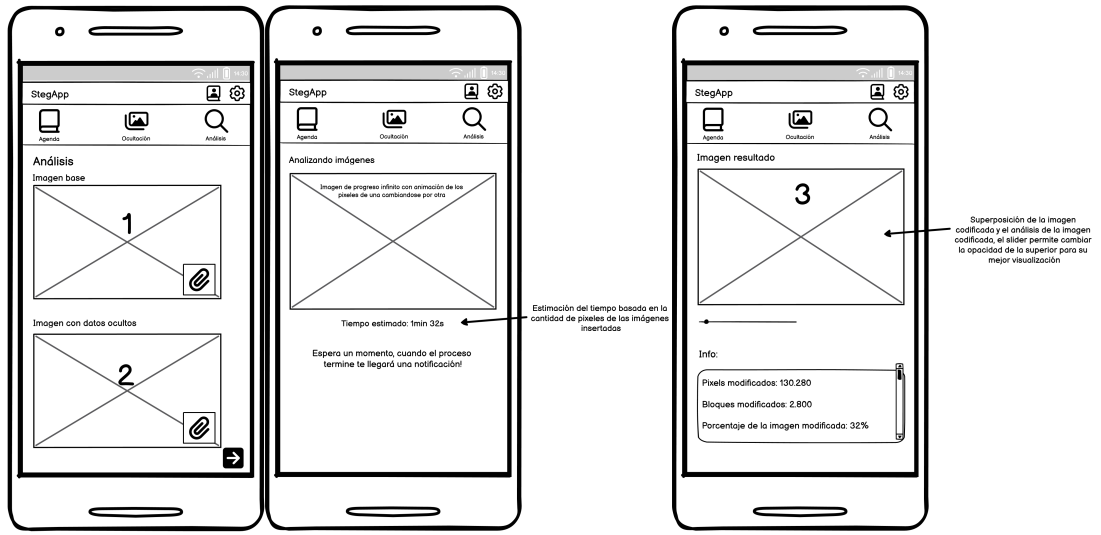
- The inception of this project stemmed from a personal initiative driven by a profound interest in the practical application of steganography in everyday scenarios. Consequently, a mobile app was developed to empower users to unleash their creativity and resourcefulness through the utilization of this practice.
- The project successfully accomplished its initially established objectives, resulting in the creation of a **fully functional mobile application** capable of executing scripts to **perform steganographic processes** on diverse file formats.
- **Steg'o**, as the resultant product, demonstrates impressive capabilities by **combining** a lot of **features** found in existing market solutions. This blend of characteristics makes **Steg'o** an adaptable and remarkably efficient application.
- We seamlessly integrated a comprehensive array of steganographic functionalities into an intuitive user-oriented interface. This interface **facilitates message transmission and reception**, **image analysis**, and various **customization options**, all presented within a contemporary and visually appealing design that adheres to the latest User Experience (UX) and User Interface (UI) standards.
- The study of the Information Technologies (IT) branch played a pivotal role in the project's development, enabling a comprehensive grasp of the fundamental aspects underlying steganographic processes.

## 12.1 Future development

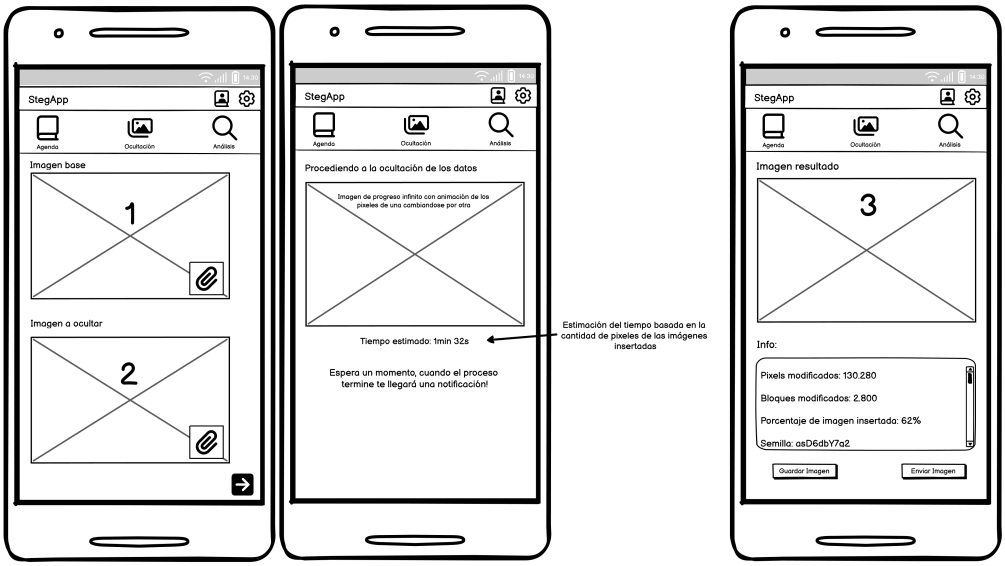
In this section, we propose several future enhancements for **Steg'o**:

- **Internationalization:** Incorporating internationalization efforts into the development process of the application, enabling its accessibility to a broader user base. This involves the addition of alternative language options in menus and textual content, facilitating user comprehension.
- **Expanded steganography capabilities:** Enhancing **Steg'o**steganography functionality by introducing support for embedding data into more file formats. For instance, enabling the insertion of data into PDF documents or the insertion of **RAR** or **ZIP** compressed files within images.
- **Group chat functionality:** Introducing a group chat system to facilitate efficient file sharing among multiple participants. Currently, **Steg'o** only supports individual chats, and the inclusion of group chats would facilitate collaborative interactions.
- **Enhanced customization options:** Providing users with greater flexibility to personalize the UI elements of the application. This could include options to customize wallpapers, primary color schemes, or other visual aspects according to individual preferences.

# Appendices

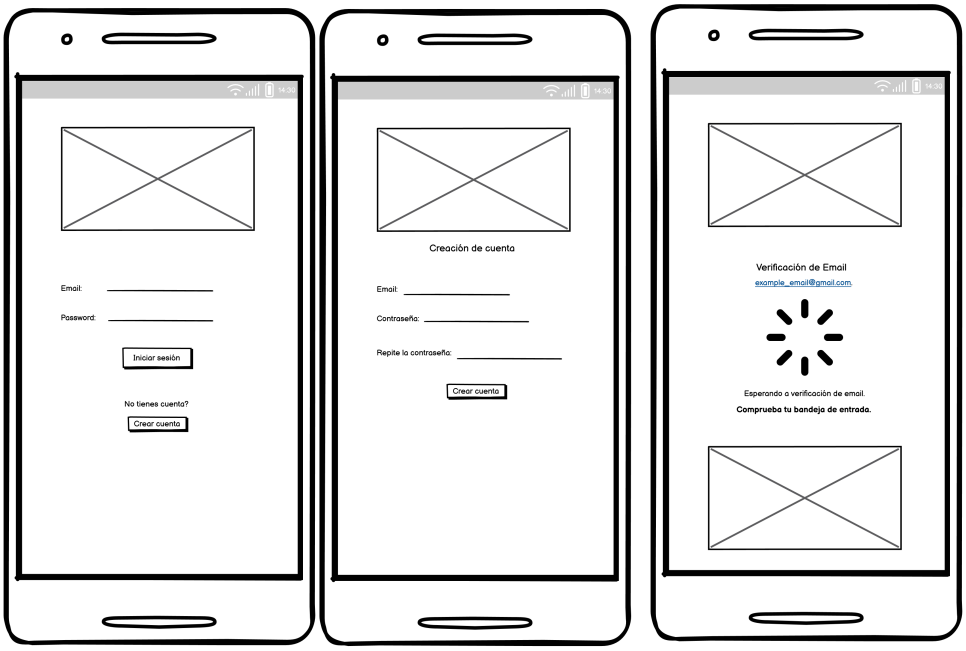



(a) Analysis wireframes

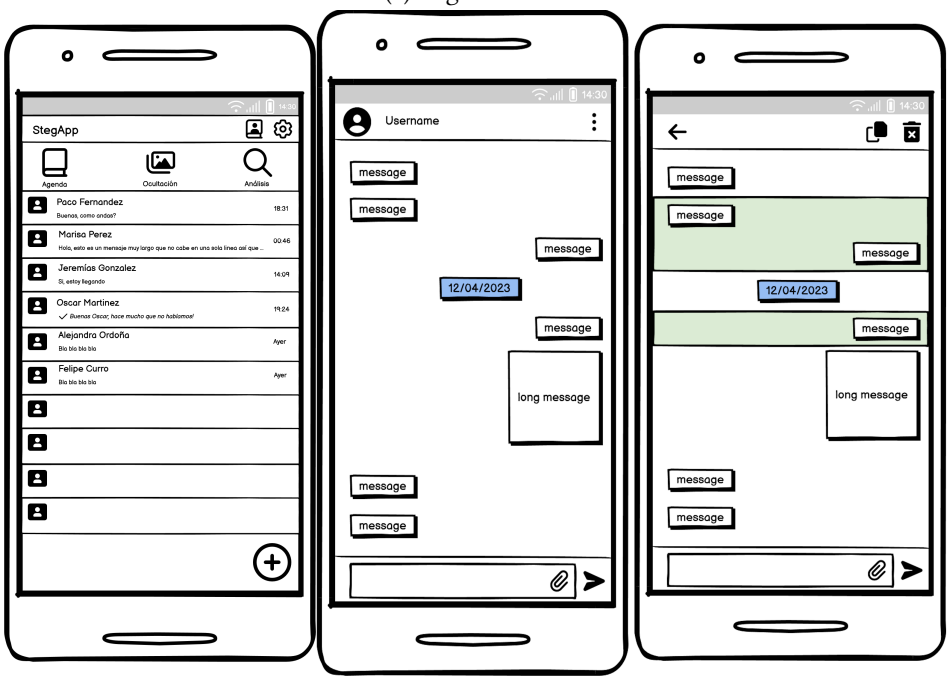



(b) Encoding wireframes

Figure 1: **Steg'o** mockups

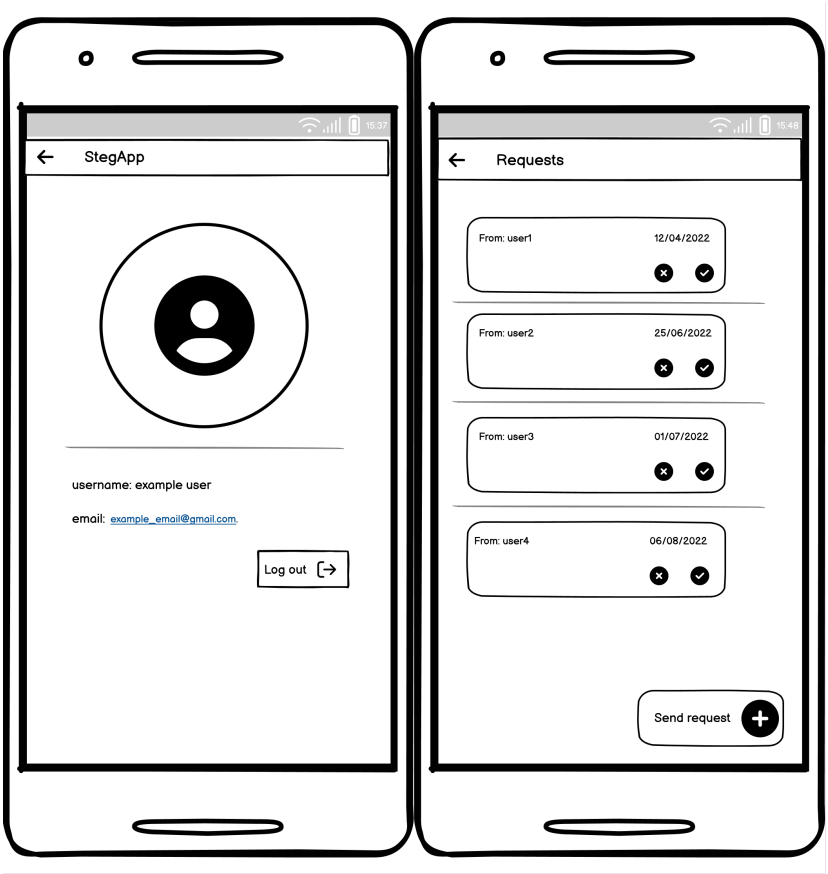



(a) Login wireframes




(b) Chat wireframes

Figure 2: **Steg'o** mockups




(a) Configuration wireframes

Figure 3: **Steg'o** mockups

```
chats (collection)
    $chatID (document)
        members (array): [
            users/hENA5o6EagOFrmzabqBjFjaFyGM2 (reference)
            users/LrRAeq5L0Leq4dIzGXPFBYNlGR42 (reference)
        ]
        messages (collection)
            $messageID (document)
            message (string): "hola"
            sendBy (reference): users/hENA5o6EagOFrmzabqBjFjaFyGM2
            time (timestamp): 10 de marzo de 2023, 00:21:13 UTC+1
            type (string): "text"

users (collection)
    $userID (document)
        email (string): "user@email.com"
        name (string): "user0"
        profilePhoto (string): "https://firebasestorage/image"
        status (string): "Unavaliable"
        userToken (string): "covwblCISDWTsIaoHBfkk8..."
        chats (collection)
            $chatID (document)
                chatRef (reference): chats/03002e00-bea8-11ed
                unreadMessages (number): 0
        requests (collection)
            $requestID (document)
                requestRef (reference): requests/YHImN9D2c5ncUluoXbZ6
                isActive (boolean): true
        enc_history (collection)
            $encodingID (document)
                images_url (array): ["path1", "path2"]
                parameters (map): {bits: 2, mode: PNG}
                result (string): "result_path"
                time (timestamp): 10 de marzo de 2023, 00:21:13 UTC+1
        dec_history (collection)
            $decodingID (document)
                decoded_data_type (string): image
                images_url (array): ["path1", "path2"]
                parameters (map): {is_video: false}
                result (string): "result_path"
                time (timestamp): 10 de marzo de 2023, 00:22:15 UTC+1

requests (collection)
    $requestID (document)
        sentByRef (reference): users/LrRAeq5L0Leq4dIzGXPFBYNlGR42
        sentToRef (reference): users/hENA5o6EagOFrmzabqBjFjaFyGM2
        state (string): "pending"
        time (timestamp): 9 de marzo de 2023, 11:12:13 UTC+1
```

Listing 1: Example of structure and contents of collections and documents in **Steg'o** database

# List of Acronyms

**API** Application Programming Interface. iii, 2, 9, 10, 22, 24, 37–40, 66, 69, 70, 77

**CLI** Command Line Interface. 14

**CRF** Constant Rate Factor. 60, 61, 63

**DCT** Discrete Cosine Transform. v, 51, 52, 55, 69

**DPCM** Differential Pulse-Code Modulation. 53

**GOP** Group of Pictures. 57, 59, 61–63

**HEVC** High Efficient Video Coding. iii, v, vii, 1, 24, 56, 60–64, 76

**HTTP** Hypertext Transfer Protocol. 39, 41, 70, 77

**IDE** Integrated Development Enviroment. vi, 12, 21, 66

**IT** Information Technologies. 79

**JPEG** Joint Photographic Experts Group. 1, 21–23, 48, 51, 53, 57, 69, 71, 75

**JSON** JavaScript Object Notation. 41

**LSB** Least Significant Bit. 1, 7, 49, 56

**MCU** Minimum Coding Unit. 52–54, 71

**ML** Machine Learning. 9

**PNG** Portable Network Graphics. 49

**QP** Quantization Parameter. 60, 61, 63

**REST** REpresentational State Transfer. iii, 39, 40, 69

**RGB** Red Green Blue. v, 1, 47, 49–51, 56, 57, 59, 68, 71

**RLE** Run Length Encoding. 53

**UI** User Interface. 11, 43, 44, 79, 80

**UML** Unified Modeling Language. 42

**UX** User Experience. 8, 79

**WIP** Work in Progress. 17

# Bibliography


[1] "Client-server model." [Online]. Available: https://commons.wikimedia.org/wiki/File:Client-server_model.svg

[2] Google, "The histories of herodotus wikipedia page."

[3] E. Cole, *Hiding in plain sight*. Wiley Hoboken, 2002.

[4] G. Project, "Pixelknot app store page." [Online]. Available: https://play.google.com/store/apps/details?id=info.guardianproject.pixelknot&hl=es&gl=US&pli=1

[5] 9iNe, "Noclue app store page." [Online]. Available: https://play.google.com/store/apps/details?id=hamza.app.steganography&hl=en_US

[6] T. Medley, "Stegdroid apk page." [Online]. Available: https://apkpure.com/stegdroid-alpha/uk.ac.cam.tfmw2.stegdroid

[7] D. Trnka, "Steganography master apk page." [Online]. Available: https://m.apkpure.com/es/steganography-master/com.dinaga.photosecret

[8] A. Boyd, "Pictograph apple store page." [Online]. Available: https://apps.apple.com/us/app/pictograph-steganography/id1051879856

[9] P. S. Foundation, "Python web page." [Online]. Available: https://www.python.org/

[10] Flask, "Flask project page." [Online]. Available: https://flask.palletsprojects.com/en/2.3.x/

[11] Google, "Firestore docs web page." [Online]. Available: https://firebase.google.com/docs/firestore?hl=es-419

[12] Mozilla, "Javascript mozilla dev web page." [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript

[13] Flutter, "Flutter dev web page." [Online]. Available: https://flutter.dev/

[14] Dart, "Dart web page." [Online]. Available: https://dart.dev/

[15] Google, "Android studio web page." [Online]. Available: https://www.educative.io/answers/how-huffmans-algorithm-works

[16] Jetbrains, "Jetbrains pycharm web page." [Online]. Available: https://www.jetbrains.com/es-es/pycharm/

[17] Overleaf, "Overleaf web page." [Online]. Available: https://es.overleaf.com

[18] L. Balsamiq Studios, "Balsamiq wireframes web page." [Online]. Available: https://balsamiq.com/wireframes/

[19] Git, "Git web page." [Online]. Available: https://git-scm.com/

[20] Google, "Material design web page." [Online]. Available: https://m3.material.io/

[21] M. Rehkopf, "Kanbanize web page." [Online]. Available: https://github.com/dwgoon/

[22] M. L. Despa, "Comparative study on software development methodologies," *Database Systems Journal*, vol. 5, no. 3, pp. 37–56, 2014.

[23] M. Awad, "A comparison between agile and traditional software development methodologies," *University of Western Australia*, vol. 30, pp. 1–69, 2005.

[24] M. Fowler, "The new methodology." [Online]. Available: https://www.martinfowler.com/articles/newMethodology.html

[25] M. Rehkopf, "Kanbanize web page." [Online]. Available: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban

[26] ——, "Kanban vs scrum," last accessed September 11, 2023. [Online]. Available: https://www.atlassian.com/agile/kanban/kanban-vs-scrum

[27] ——, "Kanbanize web page." [Online]. Available: https://www.glassdoor.es/Sueldos/project-manager-sueldo-SRCH_KO0,15.htm

[28] ——, "Kanbanize web page." [Online]. Available: https://www.glassdoor.es/Sueldos/junior-scrum-master-sueldo-SRCH_KO0,19.htm#:~:text=En%20Espa%C3%B1a%2C%20el%20sueldo%20medio,master%20es%20de%20%E2%82%AC24.828%20.

[29] ——, "Kanbanize web page." [Online]. Available: https://www.glassdoor.es/Sueldos/desarrollador-full-stack-junior-sueldo-SRCH_KO0,31.htm#:~:text=El%20sueldo%20medio%20para%20el,450%20%E2%82%AC%20y%202507%20%E2%82%AC.

[30] ——, "Kanbanize web page." [Online]. Available: https://pub.dev/packages/provider

[31] Amazon, "Nosql databases." [Online]. Available: https://aws.amazon.com/nosql/

[32] Google, "Material design web page." [Online]. Available: https://m3.material.io/

[33] "Skygge twitter page." [Online]. Available: https://twitter.com/JustSkygge?t=Dafc6NaQGsXUj1kGibiJjw&s=08

[34] "Numpy web page." [Online]. Available: https://numpy.org/

[35] U. of Indiana, "Huffman coding explanation." [Online]. Available: https://cgi.luddy.indiana.edu/~yye/c343-2019/huffman.php

[36] D. R. Galiano, A. A. Del Barrio, G. Botella, and D. Cuesta, "Efficient embedding and retrieval of information for high-resolution videos coded with hevc," *Computers & Electrical Engineering*, vol. 81, p. 106541, 2020.

[37] Google, "Firebase core package." [Online]. Available: https://pub.dev/packages/firebase_core

[38] ——, "Firebase auth package." [Online]. Available: https://pub.dev/packages/firebase_auth

[39] ——, "Firebase storage package." [Online]. Available: https://pub.dev/packages/firebase_storage

[40] ——, "Firebase firestore package." [Online]. Available: https://pub.dev/packages/cloud_firestore

[41] fangwei123456, "Python jpeg encoder." [Online]. Available: https://github.com/fangwei123456/python-jpeg-encoder

[42] Y. Khalidd, "Understanding and writing a jpeg decoder using python." [Online]. Available: https://yasoob.me/posts/understanding-and-writing-jpeg-decoder-in-python/

[43] Meta, "Whatsapp web page." [Online]. Available: https://www.whatsapp.com/?lang=es_LA

[44] Telegram, "Telegram web page." [Online]. Available: https://weba.telegram.org/

[45] D. Inc., "Discord web page." [Online]. Available: https://discord.com/

[46] dwgoon, "Jpegio github page." [Online]. Available: https://github.com/dwgoon/jpegio