

Encrypted File Transfer

Documentazione

1	Introduzione.....	3
1.1	Informazioni sul progetto.....	3
1.2	Abstract.....	3
1.3	Scopo.....	3
2	Analisi.....	4
2.1	Analisi del dominio.....	4
2.2	Analisi e specifica <u>dei</u> requisiti.....	4
2.3	Use case.....	6
2.4	Pianificazione.....	6
2.5	Analisi dei mezzi.....	6
2.5.1	Software.....	6
2.5.2	Hardware.....	6
3	Progettazione.....	7
3.1	Design dell'architettura del sistema.....	7
3.2	Design dei dati e database.....	7
3.3	Design delle interfacce.....	7
3.4	Design procedurale.....	7
4	Implementazione.....	8
4.1		
4.2		
4.3		
4.4		
5	Conclusioni.....	9
5.1	Sviluppi futuri.....	9
5.2	Considerazioni personali.....	9
6	Bibliografia.....	9
6.1	Sitografia.....	10

Titolo del progetto: Encrypted File Transfer
Alunno: Giacinto Di Santis
Classe: I3BB
Anno scolastico: 2021/2022
Docente responsabile: Luca Muggiasca

1 Introduzione

1.1 Informazioni sul progetto

Questo progetto è stato assegnatomi dalla Scuola Arti e Mestieri di Trevano (SAMT) con queste relative informazioni:

- Anno: Terzo
- Classe: I3BB
- Modulo: 306
- Mandante: Luca Muggiasca
- Titolo: File Transfer with Encryption
- Data di Inizio: 21.11.2021
- Data di fine: 23.12.2021
- Documentazione: documentazione del lavoro svolto
- Codice: codice del progetto

1.2 Abstract

Il trasferimento di file è un servizio sempre più richiesto nel mondo moderno, un mondo in cui i dati da salvare e trasferire sono in aumento anno dopo anno: Le persone oltre a dover salvare i propri file necessitano anche di un servizio che riesca a trasferire file grandi e poterlo fare in modo sicuro in modo che i dati non vengano usati dalle aziende per scopi illeciti. Garantendo quindi agli utenti un trasferimento veloce, sicuro e che possa essere usato da tutti per trasferire magari quei file che con una semplice mail non possono essere inviati, piuttosto che poi doversi scambiare delle USB/HDD/SSD o affidare a delle grandi aziende di cloud storage o trasferimenti dati non sicuri.

Un esempio di questo tipo di servizio potrebbe essere Swiss Transfer, che è molto usato soprattutto in svizzera e che però non garantisce che i file vengano letti/usati solo dal mittente e/o destinatario (potrebbe anche essere solo usato per trasferire un file da un proprio device ad un altro).

1.3 Scopo

Lo scopo di questo progetto è creare una piattaforma online che permetta il trasferimento di file, con lo storage dei dati su un server in modo sicuro, quindi criptando i file prima che arrivino al server e decriptarli solo quando vengono trasferiti al client destinatario. In modo da tenere all'oscuro il server dei dati personali del mittente proteggendo la sua privacy. Inoltre l'interfaccia dovrebbe essere user friendly e avere la possibilità di supportare file / cartelle compresse di grandi dimensioni.

2 Analisi

2.1 Analisi del dominio

- Brave Browser V1.33.106
- GitHub come repository del progetto
- Visual Studio Code come IDE
- Libre Office Writer per la documentazione
- Markdown per i diari
- TeamGantt per la stesura del Gantt

2.2 Analisi e specifica dei requisiti

ID: REQ-01	
Nome	Caricare il file tramite la GUI
Priorità	2
Versione	1.0
Note	nessuna
Sotto requisiti	
001	Il file deve essere suddiviso in chunk da 1000 byte
002	Inserire una password per criptare il file

ID: REQ-02	
Nome	Criptare il file caricato dall'utente
Priorità	2
Versione	1.0
Note	Vengono criptati i chunk da 1000byte (che poi saranno inviati nel server)

ID: REQ-03	
Nome	Inviare i chunk di byte (del file) al server
Priorità	1
Versione	1.0
Note	nessuna

ID: REQ-04	
Nome	Ricevere e salvare il file
Priorità	1
Versione	1.0
Note	Il file viene passato in chunk da 1000 byte criptati e deve essere salvato senza che il server possa conoscere il contenuto del file.
Sotto requisiti	
001	Il server deve restituire all'utente un codice (generato automaticamente in modo randomico) identificativo dell'upload.

ID: REQ-05	
Nome	Inserire la password del mittente e il codice nella GUI del destinatario
Priorità	3
Versione	2.0
Note	La password deve essere la stessa del mittente e il codice deve essere quello che il mittente ha ricevuto quando ha fatto l'upload del suo file

ID: REQ-06	
Nome	Controllare che il codice e la password forniti <u>dall'utente</u> siano corretti
Priorità	3
Versione	2.0
Note	Il server non deve salvare le password o i codici

ID: REQ-07	
Nome	Il server deve inviare i chunk di dati criptati al destinatario
Priorità	1
Versione	1.0
Note	I chunk devono essere gli stessi inviati dal mittente (1000 byte)

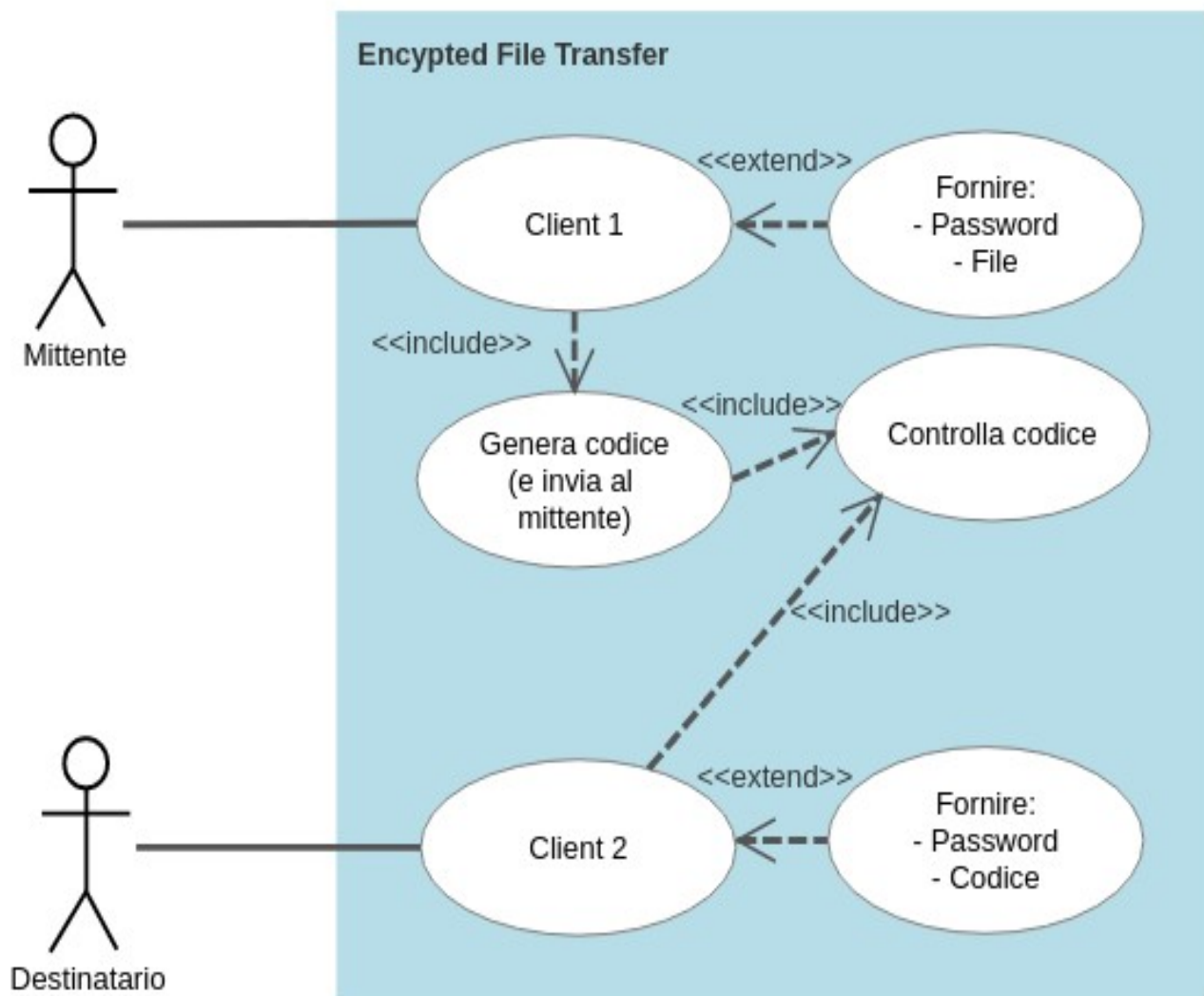
ID: REQ-08	
Nome	Il client destinatario deve decriptare i chunk inviati dal server e ricomporre il file
Priorità	1
Versione	2.0
Note	nessuna

ID: REQ-09	
Nome	Il destinatario deve poter scaricare il file richiesto (tramite codice e password precedentemente inseriti)
Priorità	2
Versione	1.0
Note	nessuna

ID: REQ-10	
Nome	Il mittente deve poter scegliere il tempo di scadenza del file (sul server) e/o il numero massimo di download
Priorità	4
Versione	2.0
Note	features aggiuntiva

ID: REQ-11	
Nome	Il sito dovrebbe essere responsive
Priorità	5
Versione	1.0
Note	Entrambi i client (upload e download)

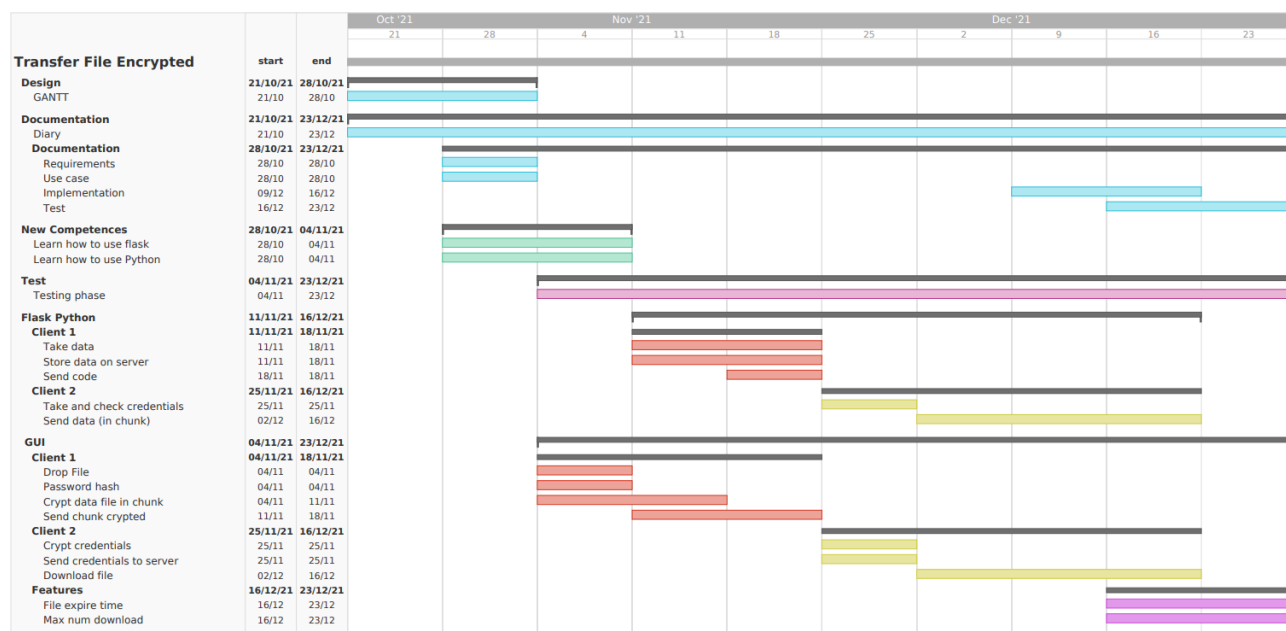
2.3 Use case



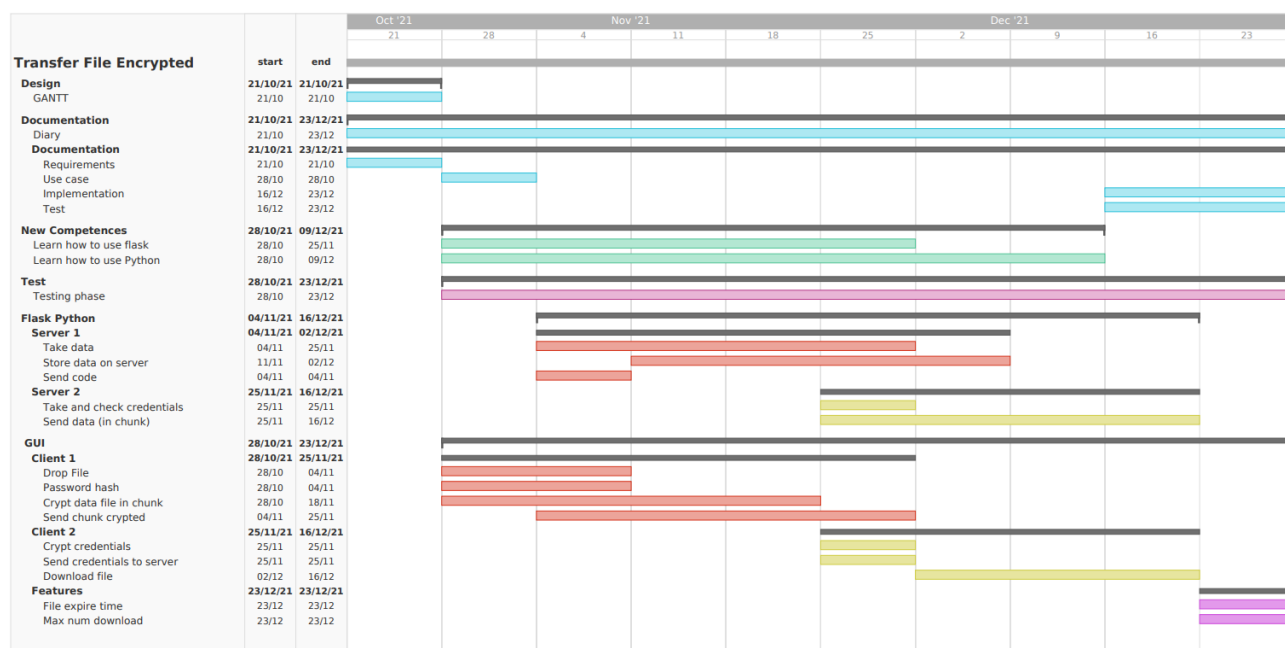
2.4 Pianificazione

I seguenti grafici sono stati fatti con l'ausilio di TeamGantt:

2.4.1 Gantt Iniziale



2.4.2 Gantt Consuntivo



La pianificazione iniziale non è stata molto rispettata siccome alcune attività ho preferito anticiparle siccome non ero sicuro di riuscire a finire in tempo, infatti molte attività sono durate più del previsto, per via di continui cambiamenti, aggiornamenti e correzioni man mano che andavo avanti con il codice (oltre agli svariati errori dovuti all'inesperienza con i linguaggi).

2.5 Analisi dei mezzi

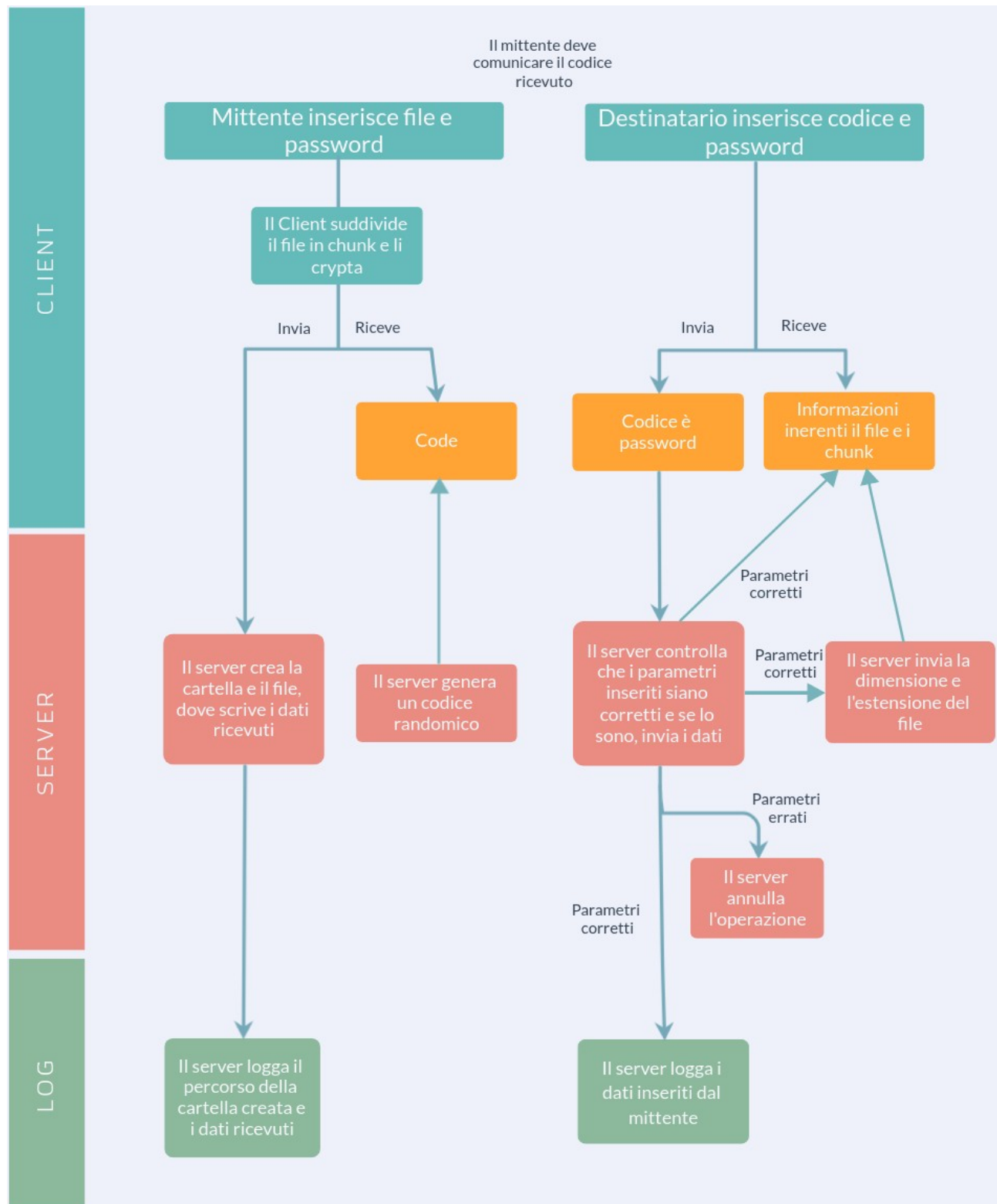
- PC scuola:
 - Intel i5-7360U
 - 16GB RAM
 - APPLE SSD SM0256L
 - Intel Iris Plus Graphics 640
- PC casa:
 - AMD Ryzen 3900X
 - 32GB RAM Corsair
 - 1TB SSD WD
 - AMD Radeon 5700XT
- Storage per i file: 1TB SSD Sandisk

2.5.1 Software

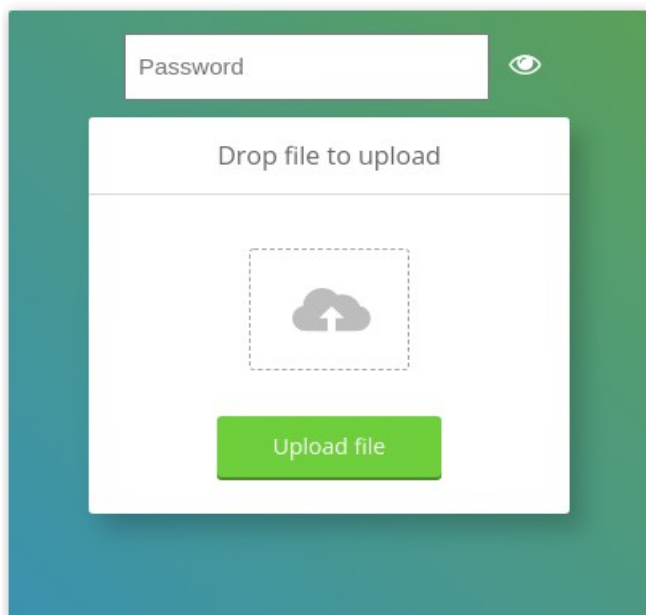
- Flask
- Requirements: render_template, os, random, SocketIO, emit, logging, base64, eventlet, eventlet.wsgi
- Python 3.10
- Brave V1.33.106

3 Progettazione

Il programma funziona con la comunicazione tra il lato fronthand e backhand, con la trasmissione di dati e le eventuali verifiche necessarie per inviare le risposte. Di seguito nella sezione sottostante c'è il diagramma di flusso (disegnato con Visme) che spiega più dettagliatamente il funzionamento del programma.

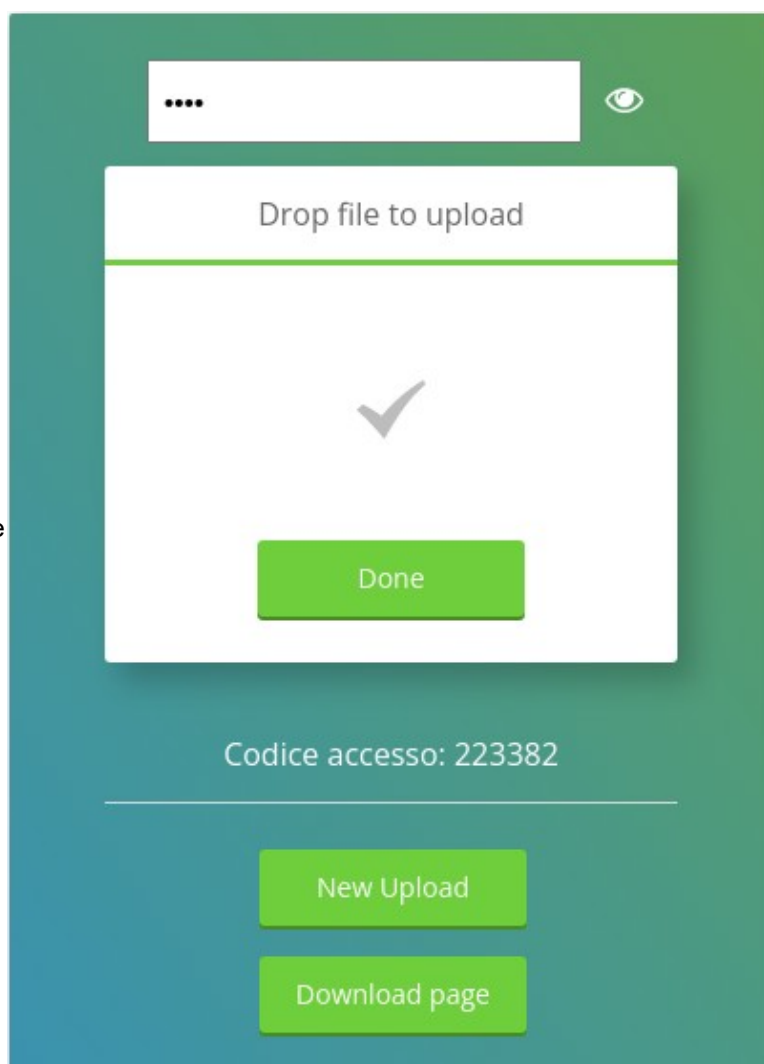


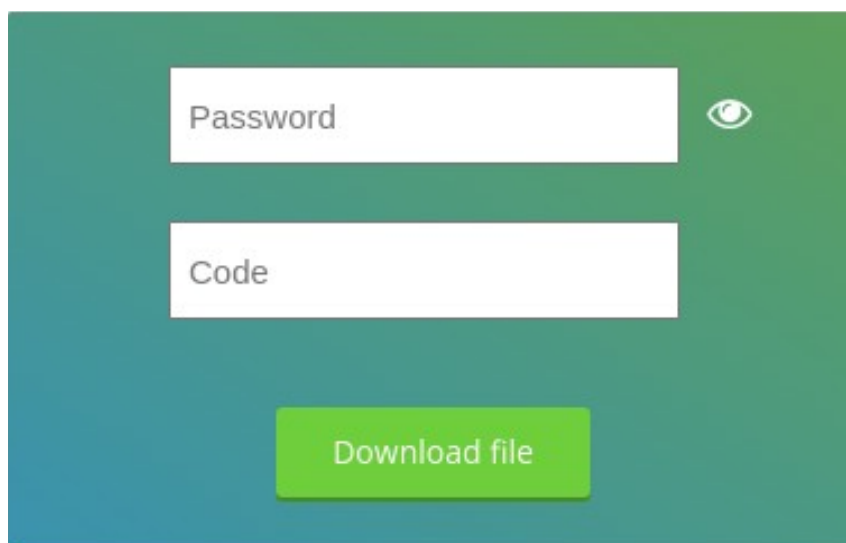
3.1 Design delle interfacce



L'interfaccia dell'upload è una semplice pagina HTML che permette all'utente che deve trasferire il file, di inserire la propria password e il file. La pagina funziona in modo che la password possa essere visualizzata dall'utente cliccando sull'icona dell'occhio. Inoltre permette nella zona di tratteggiata rettangolare di trascinare il file che si vuole trasferire. Si ha poi anche la possibilità di cliccare sulla nuvola e andare a selezionare il file desiderato tramite la finestra dell'Explorer, che si apre automaticamente (dopo il click).

Una volta premuto il bottone "Upload file" inizia il caricamento dell'immagine verso il server. Una volta che tutti i pacchetti sono arrivati il server segnala di aver terminato all'utente inviando il codice che ha generato. Quando arriva il codice, la pagina si aggiorna mostrando il codice e dando la possibilità all'utente di continuare ad eventualmente trasferire altri file tramite il bottone "New Upload". Viene infine anche mostrato il bottone per andare nella pagina di download dei file "Download page".





L'interfaccia per la pagina del download è una pagina basilare in cui al centro del sito c'è un container, come quello rappresentato nell'immagine sovrastante, che permette all'utente destinatario di inserire la password e il codice che il mittente deve avergli comunicato. Come per la pagina dell'upload l'utente può verificare la password tramite un click sull'icona dell'occhio di fianco al campo password. Una volta inseriti i dati, all'utente non basta far altro che premere il bottone "Download file", che poi farà partire il download (normale di ogni browser, tipicamente in basso a sinistra).

4 Implementazione

4.1 Client per Upload

```

90 $button.bind('click', function() {
91     var password = CryptoJS.SHA256($('#password').val())
92     var file = $('#file').prop('files')[0]
93     var sizeChunk = 1000
94     for (let i = 0; i < Math.ceil(file.size/sizeChunk); i++) {
95         var offset = i * sizeChunk
96         file.slice(offset, offset+sizeChunk).text().then(text => {
97             if(i == Math.ceil(file.size/sizeChunk)-1){
98                 socket.emit( 'take', {
99                     last: true,
100                     first: false,
101                     value: CryptoJS.enc.Base64.stringify(CryptoJS.AES.encrypt(text, password, { mode: CryptoJS.mode.ECB }).ciphertext)
102                 })
103             }else if(i == 0){
104                 socket.emit( 'take', {
105                     first: true,
106                     ext: file.name.split('.').pop(),
107                     hash: CryptoJS.enc.Base64.stringify(CryptoJS.SHA256(CryptoJS.enc.Utf8.parse(password))),
108                     value: CryptoJS.enc.Base64.stringify(CryptoJS.AES.encrypt(text, password, { mode: CryptoJS.mode.ECB }).ciphertext),
109                 })
110             }else{
111                 socket.emit( 'take', {
112                     last: false,
113                     first: false,
114                     value: CryptoJS.enc.Base64.stringify(CryptoJS.AES.encrypt(text, password, { mode: CryptoJS.mode.ECB }).ciphertext)
115                 })
116             }
117         })
118     }
119 })

```

Alla riga 90 vi è il bind del click del bottone per l'upload, ovvero se cliccato esegue la funzione sottostante.

Alla riga 91 viene salvata la password cifrata con la SHA256.

Alla riga 92 viene salvato il file.

Alla riga 94 inizia un ciclo for che viene ripetuto tante volte tante il numero di chunk.

Alla riga 95 viene settato l'offset, ovvero il numero dal quale deve iniziare a leggere i dati da inviare.

Alla riga 96 viene eseguito uno slice del file da offset fino ad altri 1000 byte, che ritorna una promise.

Dalla riga 97 alla riga 116 viene usato il text ritornato dalla promise data dallo slice del file per inviare i dati al server tramite un socket.emit(). A dipendenza di se è il primo o l'ultimo chunk vengono rispettivamente aggiunti l'hash della password con l'estensione e nel caso dell'ultimo chunk viene appunto segnalato che è l'ultimo, in ogni caso invia sempre i 1000 byte del file criptati con AES.

```

128 socket.on( 'code', function( code ){
129     uploading = true
130     $button.css("pointer-events", "none")
131     $button.html( 'Done' )
132     $('.frame').css("height", "555px")
133     $('.newfile-btn').css("display", "block")
134     $('.download-page-btn').css("display", "block")
135     $('.code').css("display", "block")
136     $('.code').text("Codice accesso: " + code)
137 })

```

Dalla riga 128 alla 137 il client del mittente si occupa di cambiare la visualizzazione della schermata in modo da mostrare il codice generato dal server e preso tramite il socket.on() nel client. Viene reso impossibile riutilizzare i dati già inseriti e vengono mostrate 2 opzioni, "New Upload" e "Download Page".

4.2 Server Upload

```

15 code = ""
16
17 @app.route("/", methods=["GET", "POST"])
18 def upload():
19     return render_template( 'upload.html' )
20
21 def generateCodeAndFolder( filename ):
22     global code
23     code = str(random.randint(000000, 999999))
24     while os.path.exists(os.path.join(os.getcwd(), f'uploads/{code}')):
25         code = str(random.randint(000000, 999999))
26     os.makedirs(str(os.path.join(os.getcwd(), f'uploads/{code}')))
27     logging.debug(f'path generate: {os.path.join(os.getcwd(), f"uploads/{code}/{filename}")}')
28     return os.path.join(os.getcwd(), f'uploads/{code}/{filename}')
29
30 @socketio.on( 'take' )
31 def taked( json ):
32     logging.debug(f'response taked: {str(json)}')
33     if json['first']:
34         with open(generateCodeAndFolder(((str(json['hash']).replace("/", "$")) + "." + json['ext'])), 'wb') as f:
35             f.write(base64.b64decode(json['value']))
36     if json['last']:
37         global code
38         pathfile = "".join(f'{os.getcwd()}/uploads/{code}/{x}' for x in os.listdir(f'{os.getcwd()}/uploads/{code}'))
39         with open(str(pathfile), 'wb') as f:
40             f.write(base64.b64decode(json['value']))
41         emit('code', (pathfile.split("/"))[7])
42     else:
43         pathfile = "".join(f'{os.getcwd()}/uploads/{code}/{x}' for x in os.listdir(f'{os.getcwd()}/uploads/{code}'))
44         with open(str(pathfile), 'wb') as f:
45             f.write(base64.b64decode(json['value']))

```

Alla riga 15 viene istanziata / dichiarata la variabile globale code, che essendo limitata nel tenere la memoria, serve solo per usarla tra 2 funzioni.

Alla riga 17 viene stabilito che chi contatta il server nella root «/» può usare i metodi get e post, oltre che visualizzare la pagina html del upload (tramite il render_template riga 18).

Dalla riga 21 alla 28 la funzione generateCodeAndFolder si occupa di generare un codice randomico compreso tra 000000-999999, che non sia già stato usato e una volta trovato crea la cartella con il nome del codice che ha generato. Inoltre grazie al parametro filename che gli viene passato crea anche il file dove poi verranno scritti i dati. Per verificare che il codice sia univoco e quindi mai usato, controllo che non esista già una cartella con il numero generato dal random (riga 24).

Dalla riga 30 alla 45 il server si occupa di prendere i dati inviati dal mittente (tramite la pagina di upload). Quando riceve questi dati controlla se i dati ricevuti siano i primi o gli ultimi, se sono i primi allora viene richiamata la funzione per generare il codice (cartella) e il file, se invece è l'ultimo invia il codice al client in modo da segnalargli di aver finito di salvare i file che esso ha inviato. In tutti i casi va a scrivere nel file il valore passato, prima decodificandolo dalla base 64.

4.3 Client Download

```

40      $('.download-btn').bind('click', function() {
41          password = CryptoJS.SHA256($(':password').val())
42          socket.emit( 'request', {
43              hash: CryptoJS.enc.Base64.stringify(CryptoJS.SHA256(CryptoJS.enc.Utf8.parse(password))),
44              code: $(':text').val()
45          })
46      })

```

Alla riga 40 viene definita la funzione da eseguire se viene cliccato il tasto di download.
Se cliccato viene criptata la password con la SHA256 e poi viene inviato il codice con l'hash al server.

```

47      socket.on( 'info', function( data ){
48          lengthFile = data['size']
49          ext = data['ext']
50          console.log("n. chunk: " + (Math.ceil(lengthFile/1000)-1) + "\nsize: " + lengthFile + "\next: " + ext)
51      })
52      socket.on( 'download', function( encrypted ){
53          if(i < Math.ceil(lengthFile/1000)){
54              file += CryptoJS.AES.decrypt(encrypted, password, { mode: CryptoJS.mode.ECB }).toString()
55              if((Math.ceil(lengthFile/1000)-1) == i){
56                  var arrayBuffer = new Uint8Array(file.match(/[\da-f]{2}/gi).map(function (h) {return parseInt(h, 16)}))
57                  var content = new Blob([arrayBuffer]);
58                  var a = document.createElement('a')
59                  a.href = window.URL.createObjectURL(content)
60                  a.download = ("prova." + ext)
61                  a.click()
62              }
63          }
64          i+=1
65      })

```

Dalla riga 47 alla 51 il client del download si occupa di prendere le informazioni dal server: lunghezza del file e estensione.

Dalla riga 52 alla 65 il client (dovrebbe) ricostruisce il file, quindi prende ogni chunk di dati che riceve lo decripta e lo aggiunge ad una variabile file, che poi conterrà tutti i byte decriptati. Quando arriva all'ultimo chunk converte il file in un array buffer, che poi mette in un blob che userà per scaricarlo come un file tramite un a (tag) che viene costruito e cliccato in automatico per far partire il download.

Il programma sa quando sta processando l'ultimo chunk siccome si basa sulla lunghezza del file che gli è stata precedentemente passata e usa un contatore per sapere quanti chunk ha processato fino a quel momento.

4.4 Download Server

```

61      @socketio.on( 'request' )
62      def check( json ):
63          logging.debug(f'response check: {str(json)}')
64          pathfile = "".join(f'{os.getcwd()}/uploads/{json["code"]}/{x}' for x in os.listdir(f'{os.getcwd()}/uploads/{json["code"]}'))
65          percorso = f"{os.getcwd()}/uploads/{json['code']}/{(str(json['hash']).replace('/', ' $'))}.{pathfile.split('.')[1]}"
66          if os.path.exists(pathfile) and pathfile == percorso:
67              logging.debug(f"size: {os.path.getsize(percorso)} ext: {pathfile.split('.')[1]}")
68              emit('info', { 'size': os.path.getsize(percorso), 'ext': pathfile.split('.')[1] })
69              with open(percorso, "rb") as f:
70                  while (byte := f.read(1000)):
71                      emit('download', str(base64.b64encode(byte).decode('ascii')))

```

Alla riga 51 il server è in ascolto di un request

Alla riga 54 viene cercato e scritto il percorso del file all'interno della cartella con il codice passato

Alla riga 55 viene scritta nella variabile il percorso che sarebbe generato dalle credenziali passate dal destinatario

Alla riga 56 viene controllato se il percorso del file dato dal codice esista (quindi il codice fornito è esistente) e se i 2 percorsi corrispondano (quindi i 2 file hanno lo stesso nome ovvero hash, quindi password)

Alla riga 58 vengono trasmesse al client destinatario le informazioni inerenti il file necessarie

Dalla riga 59 alla 61 vengono inviati al client i chunk (1000 byte alla volta) di dati presenti nel file sul server

5 Conclusioni

5.1 Sviluppi futuri

In futuro sarebbe interessante riuscire concretamente a far funzionare il programma, oltre che magari aggiungere le 2 funzionalità di limite di tempo sul server e limite di download. Inoltre si potrebbe provare a salvare i file in un database o trovare un sistema più efficiente per il salvataggio dei byte sul server-

5.2 Considerazioni personali

Il progetto è stato abbastanza impegnativo e stressante, avendo avuto poco tempo in ogni momento c'era la pressione e lo stress di essere in ritardo con il programma. Anche se tutto sommato è stato un'esperienza interessante, ho dovuto aprire la mente e impegnarmi per cercare risposte più velocemente, oltre che imparare da 0 a programmare in Python e ad usare Flask. Sono contento di aver imparato delle nuove cose che magari potranno servirmi in futuro e se pur difficoltoso l'essermi sotto posto a questi stress mi ha migliorato, aumentando la capacità di adattamento per quanto marginale. Immagino abbia aiutato anche il fatto che ero incuriosito dal programma stesso e avrei voluto farlo funzionare. Spero che per il prossimo progetto con più tempo possa fare un lavoro migliore e imparare nuove competenze come in questo caso.

6 Bibliografia

6.1 Sitografia

1. <https://flask.palletsprojects.com/en/2.0.x/quickstart/>, *Quick start*, 28-11-2021.
2. <https://socket.io/docs/v4/>, *Socket.IO*, 04-12-2021.
3. https://flask-socketio.readthedocs.io/en/latest/getting_started.html#sending-messages, *Sending Messages*, 04-12-2021
4. <https://stackabuse.com/creating-and-deleting-directories-with-python/>, *Python os*, 09-12-2021
5. https://www.tutorialspoint.com/python/os_listdir.htm, *listdir*, 09-12-2021
6. <https://www.guru99.com/reading-and-writing-files-in-python.html>, *file*, 09-12-2021
7. <https://docs.python.org/3/library/base64.html>, *base 64*, 10-12-2021
8. <https://thispointer.com/how-to-create-a-directory-in-python/>, *Directory*, 16-12-2021
9. <https://www.geeksforgeeks.org/python-os-path-isdir-method/?ref=lbp>, *path is dir*, 16-12-2021
10. <https://javascript.info/blob>, *Blob*, 16-12-2021
11. <https://github.com/eligrey/FileSaver.js/blob/master/src/FileSaver.js>, *File Saver*, 16-12-2021
12. <https://www.pdftron.com/documentation/web/guides/basics/open/arraybuffer/>, *Array Buffer*, 17-12-2021
13. <https://stackoverflow.com/questions/35038884/download-file-from-bytes-in-javascript>, *Download file*, 18-12-2021
14. <https://stackoverflow.com/questions/5104957/how-do-i-create-a-file-at-a-specific-path>, *open file*, 18-12-2021
15. <https://stackabuse.com/encoding-and-decoding-base64-strings-in-python/>, *base 64*, 18-12-2021