

Relazione Reti StudiSync

Luca Sito 0124002612 Vittorio Picone 0124002584

June 20, 2024

1 Indice

1. Traccia
2. Introduzione
3. Struttura
4. Comunicazione
 - (a) Request
 - (b) Give
 - (c) Funzioni
 - i. Client
 - A. Full Read
 - B. Read Scket
 - C. Client Echo
 - D. Launch Method
 - ii. Server
 - A. Class Myhandler
 - B. Class ThreadedTCPServer
 - C. Server Main
5. Manuale Utente
 - (a) Come Installare
 - (b) Login
 - (c) Segreteria
 - i. Aggiungi Corso Di Laurea
 - ii. Aggiungi appello
 - iii. Aggiungi Corso
 - iv. Gestisci richieste Date
 - v. Gestisci prenotazioni Appelli

- (d) Studente
 - i. Richiedi Date
 - ii. Richiedi Prenotazione
 - iii. Visualizza Storico Prenotazioni
- 6. Database

2 Traccia

Scrivere un'applicazione client/server parallelo per gestire gli esami universitari
Segreteria:

- Inserisce gli esami sul server dell'università (salvare in un file o conservare in memoria il dato)
- Inoltra la richiesta di prenotazione degli studenti al server universitario
- Fornisce allo studente le date degli esami disponibili per l'esame scelto dallo studente

Studente:

- Chiede alla segreteria se ci siano esami disponibili per un corso
- Invia una richiesta di prenotazione di un esame alla segreteria

Server universitario:

- Riceve l'aggiunta di nuovi esami
- Riceve la prenotazione di un esame

Il server universitario ad ogni richiesta di prenotazione invia alla segreteria il numero di prenotazione progressivo assegnato allo studente e la segreteria a sua volta lo inoltra allo studente.

3 Introduzione

Il progetto è sviluppato nel linguaggio di programmazione **python** con supporto grafico mediante la libreria **pyqt5**. Sono previsti 2 livelli di accesso: segreteria e studente; l'autenticazione avviene tramite apposita interfaccia di login. Il terzo attore è il server universitario che gestisce le operazioni di lettura e scrittura sul DB rispondendo in maniera parallela le richieste dai diversi Client mediante l'utilizzo dei Socket.

4 Struttura

Di seguito viene mostrata la struttura dei file nel progetto

- **Cversion:** contiene gli algoritmi in c che sono stati tradotti in python
- **Common:** contiene gli script comuni, come il protocollo di comunicazione
- **Logic:** contiene le logiche dietro le interfacce
 - **Segreteria:** Contiene le logiche dietro le interfacce della segreteria
 - **Students:** Contiene le logiche dietro le interfacce degli studenti
- **db:** contiene i file CSV con i dati
 - **Esami:** contiene i file che riguardano gli esami, come appelli o corsi
 - **Prenotazioni:** contiene i file che riguardano le richieste degli utenti
 - **Users;** Contiene il file con le credenziali degli utenti
- **deprecated:** contiene file non utilizzati nel codice
- **gui:** contiene i file riguardanti l' interfaccia grafica
 - **Segreteria:** Contiene i file che generano le interfacce della segreteria
 - **Students:** Contiene i file che generano le interfacce degli studenti
 - **Designer:** contiene i file .ui generati tramite qt designer

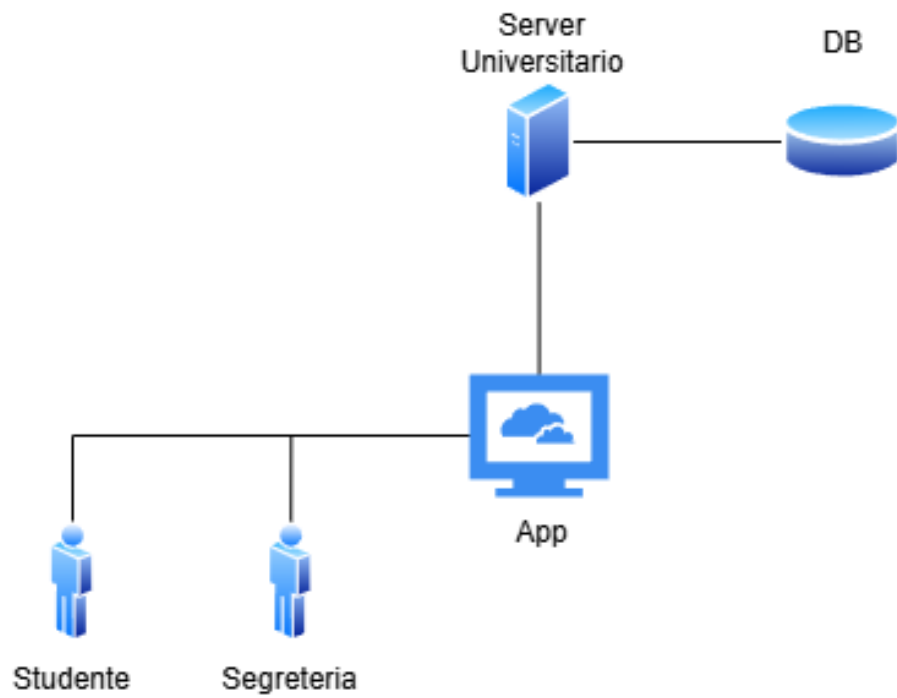


Figure 1: Connessioni

Connessioni

Figure 1 Descrive come avvengono le connessioni. I 2 diversi tipi di utenti effettuano richieste al server attraverso l' applicazione. Il server è l' unica entità che legge e scrive sul DB

5 Comunicazione

Il protocollo di comunicazione tra client e server ha come suo fondamento il formato di scambio di dati JSON. Nell'infrastruttura possiamo trovare due tipologie di JSON, **request** e **give**:

5.1 Request

```
1 {  
2   "header": "StudentsLogin",  
3   "payload": {  
4     "Matricola": "0124002584",  
5     "Password": "test123"  
6   }  
7 }
```

Nell'esempio mostrato, si presenta un JSON **request**, formato da due attributi, **header** e **payload**, dove nell'header è presente l'endpoint richiesto, in questo caso il login per gli studenti, mentre nel payload sono presenti le informazioni necessarie al server per processare la request con successo, che in questo caso è matricola e password. La request viene decodifica e divisa nelle sue due componenti, per poi essere passata alla funzione `method_switch`, dove al suo interno è presente uno switch/case dove i case rappresentano i vari endpoint disponibili, richiamati all'interno del case con una funzione omonima all'endpoint.

5.2 Give

```
1 {  
2   "result": [  
3     "0124002584",  
4     "Vittorio",  
5     "Picone",  
6     "vittorio.picone001@studenti.uniparthenope.it",  
7     "1914752590",  
8     "0124",  
9     [  
10      "0124",  
11      "Informatica_Triennale"  
12    ]  
13   ]  
14 }
```

In quest altro esempio mostrato, si presenta la seconda tipologia di JSON, **give**, formato da un solo attributo, **result**, che contiene il risultato fornito dall'endpoint richiesto, che in questo caso è il risultato di un login studente andato a buon fine. I dati nell'attributo `result` vengono restituiti come un

array, e vengono interpretati direttamente lato client. La funzione per effettuare le chiamate agli endpoint è **launchMethod**, che come parametri di input ha input (str), server_address (str) e server_port(int). La funzione launchMethod è stata poi abbinata per un'usabilità maggiore alle due funzioni **request_constructor_str** e **request_constructor_obj**, dove request_constructor_obj restituisce un JSON in formato object a request_constructor_str, e quest ultimo ritorna il JSON come stringa a launchMethod, avendo quindi il seguente flow operativo:

```
1 def request_constructor_obj(input_object, header):
2     return {
3         "header": header,
4         "payload": input_object
5     }
6
7
8 def request_constructor_str(input_object, header):
9     return json.dumps(request_constructor_obj(input_object, header)
10 )
11
12 launchMethod(request_constructor_str(None, "GetCorsi"),
13               server_coords['address'], server_coords['port'])
```

5.3 Funzioni per la comunicazione

5.3.1 Client

```
1 def full_write(fd, buf):
2     nleft = len(buf)
3     while nleft > 0:
4         try:
5             nwritten = fd.send(buf)
6             nleft -= nwritten
7             buf = buf[nwritten:]
8         except socket.error as e:
9             if e.errno == errno.EINTR:
10                 continue
11             else:
12                 raise
13     return nleft
14
15 async def read_socket(sock):
16     received_data = ""
17     while True:
18         recvbuff = await asyncio.to_thread(sock.recv, MAXLINE)
19         if not recvbuff:
20             print("EOF on the socket")
21             break
22         if not recvbuff.strip():
23             print("Empty data received. Closing the socket.")
24             break
25         received_data += recvbuff.decode()
26     return received_data
27
```

```

28
29 async def client_echo(data, sock):
30     # Write the data using full_write
31     await asyncio.to_thread(full_write, sock, data.encode())
32     # Wait for the socket to be flushed (optional)
33     await asyncio.to_thread(sock.shutdown, socket.SHUT_WR)
34     return await read_socket(sock)
35
36
37 def launchMethod(input: str, server_address: str, server_port: int)
38 :
39     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
40
41     serv_add = (server_address, server_port)
42
43     try:
44         sock.connect(serv_add)
45         print(f"Connection to {serv_add} established!")
46     except Exception as e:
47         print(f"Connection error: {e}")
48         sys.exit(1)
49
50     result = asyncio.run(client_echo(input, sock))
51
52     # Close the socket after finishing the data stream
53     sock.close()
54
55     return result

```

full_write(fd, buf) Questa funzione ha il compito di inviare tutti i dati presenti nel buffer buf sul file descriptor fd (in questo caso, una socket). Ecco come funziona:

1. Calcolo di nleft: Viene calcolata la lunghezza dei dati da inviare.
2. Ciclo while: Continua finché ci sono dati da inviare (nleft > 0).
3. Tentativo di invio dati: Utilizza fd.send(buf) per inviare i dati.
4. Aggiornamento variabili: Sottrae il numero di byte inviati (nwritten) da nleft e aggiorna buf per rimuovere i dati già inviati.
5. Gestione errori: Se si verifica un'eccezione di tipo socket.error, controlla se l'errore è errno.EINTR (interruzione del sistema). In tal caso, il ciclo continua; altrimenti, l'eccezione viene rilanciata.

read_socket(sock) Questa funzione asincrona legge i dati da una socket finché non riceve EOF (End Of File) o dati vuoti:

1. Inizializzazione di received_data: Una stringa vuota per accumulare i dati ricevuti.
2. Ciclo while: Continua indefinitamente.

3. Lettura dati: Usa `asyncio.to_thread` per eseguire `sock.recv` in un thread separato e attendere il risultato.
4. Controllo EOF: Se non vengono ricevuti dati (`not recvbuff`), stampa un messaggio e interrompe il ciclo.
5. Controllo dati vuoti: Se i dati ricevuti sono vuoti, stampa un messaggio e interrompe il ciclo.
6. Accumulo dati: Aggiunge i dati decodificati a `received_data`.
7. Ritorno dei dati ricevuti: Al termine del ciclo, restituisce `received_data`.

`client_echo(data, sock)` Questa funzione asincrona invia dei dati ad un server e legge la risposta:

1. Invio dati: Utilizza `asyncio.to_thread` per eseguire `full_write` in un thread separato, inviando i dati codificati (`data.encode()`) tramite la socket.
2. Chiusura socket: Chiude il canale di invio della socket (`sock.shutdown(socket.SHUT_WR)`).
3. Lettura risposta: Attende e ritorna la risposta dal server tramite `read_socket(sock)`.

`launchMethod(input: str, server_address: str, server_port: int)` Questa funzione avvia la comunicazione con il server:

1. Creazione della socket: Crea una socket TCP/IP.
2. Costruzione dell'indirizzo del server: Crea una tupla con l'indirizzo e la porta del server.
3. Connessione al server: Tenta di connettersi al server e stampa un messaggio di conferma.
4. Gestione errori di connessione: In caso di errore durante la connessione, stampa un messaggio e termina il programma.
5. Usa `asyncio.run` per eseguire la funzione `client_echo` e attende il risultato.
6. Chiude la socket.
7. Restituisce il risultato ottenuto da `client_echo`.

5.3.2 Server

```
1 class MyHandler(socketserver.StreamRequestHandler):
2     def handle(self):
3
4         host, port = self.client_address
5         print(f"Request from host {host}, port {port}")
6
7         while True:
8             try:
9                 data = self.rfile.readline(MAXLINE)
10                if not data:
11                    print(f"Connection closed by {self.
12                        client_address}")
13                    break
14
15                print(f"Received from {self.client_address}: {data}
16                    ")
17                data_decoded = data.decode().replace('\n', '')
18                data_decoded = json.loads(data_decoded)
19
20                print(f"Parsed Data: {data_decoded}")
21
22                result = method_switch(data_decoded["header"],
23                    data_decoded["payload"])
24                response = f"{json.dumps(result)}.encode('utf-8")
25                print(f"Response to send: {response}")
26
27                # Use blocking send to ensure the response is fully
28                sent before closing
29                sent = full_write(self.request, response)
30                print(f"Sent {sent}")
31
32            except socket.error as e:
33                if e.errno == 10054:
34                    print(f"Connection forcibly closed by the
35                        remote host {self.client_address}")
36                else:
37                    print(f"{bcolors.FAIL} Socket Error: {e}{
38                        bcolors.ENDC}")
39                    break
40            except Exception as e:
41                print(f"{bcolors.FAIL} Generic Exception Error: {e
42                    }{bcolors.ENDC}")
43                break
44
45 class ThreadedTCPServer(socketserver.ThreadingMixIn, socketserver.
46     TCPServer):
47     pass
48
49 def server_main(server_address, server_port):
50     server = ThreadedTCPServer((server_address, server_port),
51         MyHandler)
52     server_thread = threading.Thread(target=server.serve_forever)
53     server_thread.start()
```

```

47     print(f"Server listening on port {server_port}...")
48
49     try:
50         server_thread.join()
51     except KeyboardInterrupt:
52         print("Server terminated by user")

```

Class MyHandler

1. Identificazione del client: Stampa l'indirizzo del client connesso.
2. Ciclo di lettura dei dati: Continua a leggere dati dal socket finché non viene chiusa la connessione.
 - (a) Lettura dei dati: Legge una linea di dati dalla socket.
 - (b) Chiusura della connessione: Se non ci sono dati, la connessione è chiusa.
 - (c) Decodifica e parsing dei dati: I dati ricevuti sono decodificati da bytes a stringa e poi convertiti in un oggetto JSON.
 - (d) Elaborazione della richiesta: Usa `method_switch` per elaborare i dati e ottenere un risultato.
 - (e) Invio della risposta: Codifica il risultato in bytes e lo invia al client usando `full_write`.
 - (f) Gestione degli errori: Gestisce eventuali errori di socket e generici, stampando messaggi di errore colorati.

Classe ThreadedTCPServer Questa classe estende `TCPServer` e `ThreadingMixIn` per permettere la gestione multithreading delle richieste

Funzione server_main

1. Creazione del server: Crea un'istanza di `ThreadedTCPServer` con l'indirizzo e la porta specificati.
2. Avvio del thread del server: Avvia un thread separato per eseguire il server in modo che possa gestire le richieste in arrivo.
3. Messaggio di ascolto: Stampa un messaggio indicando che il server è in ascolto sulla porta specificata.
4. Attesa del thread: Usa `join` per mantenere il programma in esecuzione finché il thread del server è attivo.
5. Gestione dell'interruzione: Gestisce l'interruzione da tastiera (`KeyboardInterrupt`) per permettere all'utente di terminare il server pulitamente.

6 Manuale Utente

6.1 Come Installare

Linux Distrubution: Ubuntu

1. Aggiorna sudo: `sudo update` poi `sudo upgrade`
2. Aggiorna le librerie: `sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev libffi-dev libsqlite3-dev wget libbz2-dev`
3. Scarica python 3.11: `wget https://www.python.org/ftp/python/3.11.0/Python-3.11.0.tgz`
4. Estrai il file scaricato: `tar -xf Python-3.11.0.tgz`
5. Vai nella cartella: `cd Python-3.11.0`
6. Ottimizza la build `./configure --enable-optimizations`
7. Assegnamo più processori per velocizzare la build: `make -j$(4)`
8. installa: `sudo make altinstall`
9. per sicurezza vediamo se l'installazione è avvenuta con successo: `python3.11 --version`
10. installiamo git: se sei su ubuntu: `sudo apt install git-all` altrimenti: `sudo dnf install git-all`
11. clona il progetto: `git clone https://github.com/VikSn0w/StudiSync.git`
(verranno richieste le credenziali o un access token)
12. Spostati nell directory del progetto: `cd ;dirProgetto;`
13. Creiamo un venv: `python3 -m venv .venv`
14. Entiamo nel venv: `source .venv/bin/activate`
15. Installa le librerie: `pip install -r requirements.txt`
16. Installiamo dipendenze: `sudo apt-get install libxcb-xinerama0`
`sudo apt-get install libxkbcommon-x11-0 libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-randr0 libxcb-render-util0 libxcb-xinerama0 libxcb-xfixes0 libegl1-mesa`
17. Avvia il server: `python combined_multiplex_concurrent_server.py`
18. Avvia il main: `python main.py`

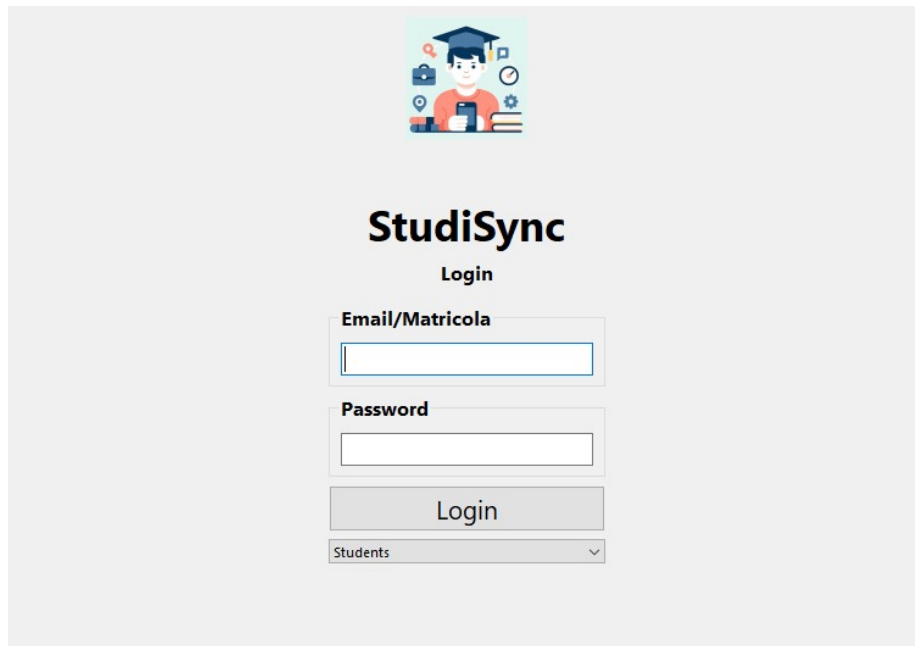


Figure 2: Interfaccia Login

6.2 Login

Figura 2 rappresenta l'interfaccia di login, si necessita di inserire email/matricola in caso di studente e password personale, gli utenti vengono inseriti manualmente nei file CSV con le password crittografate tramite un algoritmo custom. Bisogna inoltre specificare la tipologia di utente che sta eseguendo l'accesso

Se le credenziali inserite sono corrette si apre l'interfaccia Home dell'utente selezionato, altrimenti si ottiene un messaggio di errore "email, password or user type incorrect. Check your info and retry"

6.3 Segreteria

Quando un utente della segreteria effettua il login, viene mostrata questa interfaccia dove è possibile accedere a tutte le funzionalità

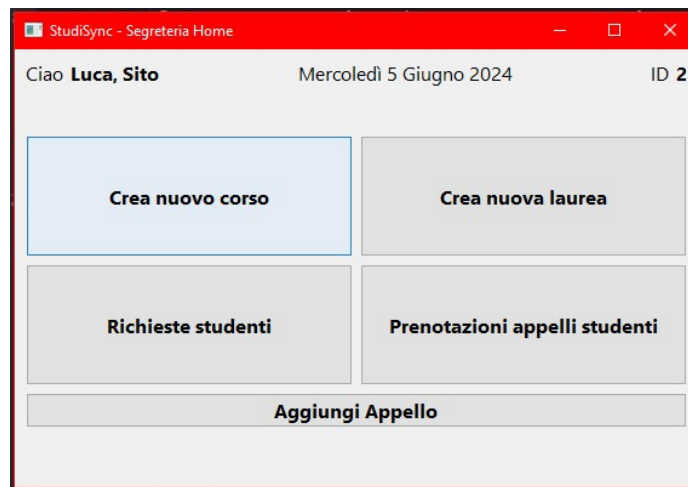
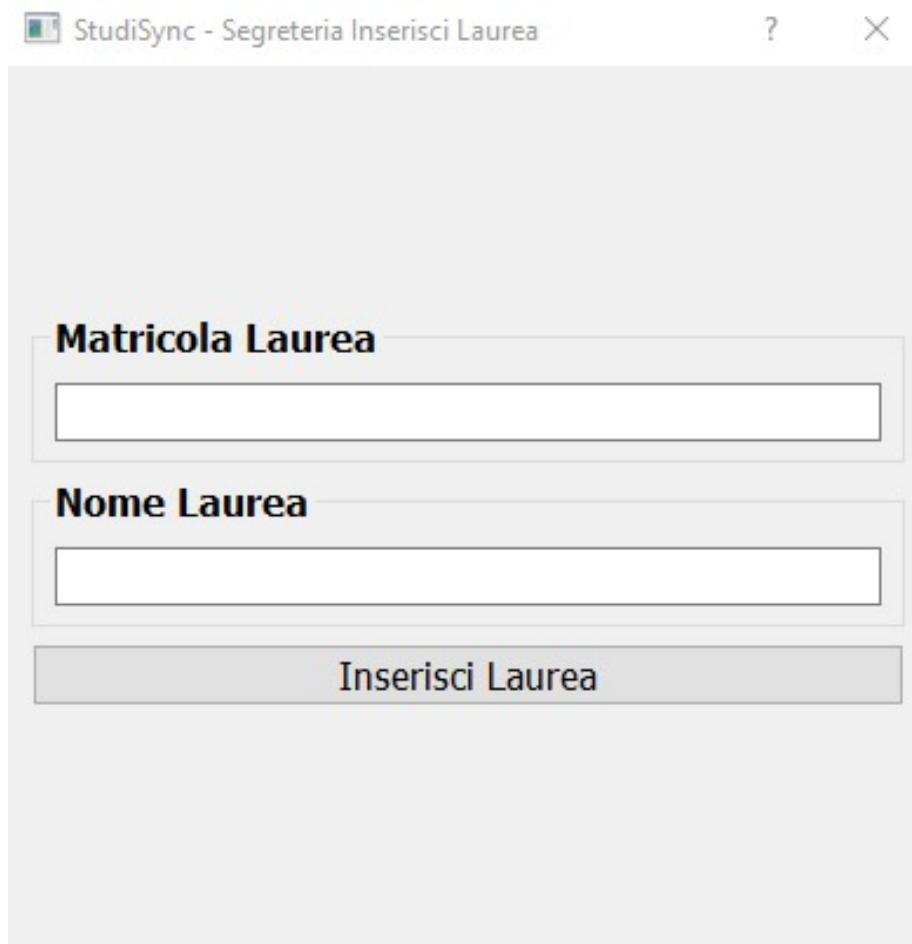


Figure 3: Home segreteria

6.3.1 Aggiungi Corso Di Laurea



The image shows a software window titled "StudiSync - Segreteria Inserisci Laurea". Inside the window, there are two text input fields. The first field is labeled "Matricola Laurea" and the second is labeled "Nome Laurea". Below these fields is a button labeled "Inserisci Laurea".

Figure 4: Aggiungi Laurea

La figura 4 mostra l' interfaccia aggiungi corso di laurea, i campi richiesti sono la matricola e il nome di riconoscimento della laurea. Una volta inseriti i dati e premuto sul pulsante "Inserisci laurea", verrà mostrato un messaggio di avvenuto inserimento del corso di laurea se non si verifica nessun errore.

6.3.2 Aggiungi Appello

StudiSync - Segreteria Inserisci Appello

Corso

[MAT1] Matematica 1 - 6 CFU

Data Appello

01/01/2000 00:00

Luogo

Inserisci Appello

Figure 5: Inserisci Appello

La figura 5 mostra l' interfaccia inserisci appello, i campi richiesti sono

- Il corso del quale si vuole aggiungere un appello, selezionabile tramite menù a tendina, verranno mostrati i corsi aggiunti in precedenza tramite le funzione "Aggiungi Corso"
- Data e ora dell' appello
- Luogo in cui si effettuerà la prova

Una volta inseriti i dati e premuto sul pulsante "Inserisci appello", verrà mostrato un messaggio di avvenuto inserimento dell'appello se non si verifica nessun errore.

StudiSync - Segreteria Inserisci Esame

ID Esame

Nome Esame

Nome Professore

Corso di Laurea

0124 - Informatica Triennale

CFU

1

Inserisci Esami

Figure 6: Aggiungi Corso

6.3.3 Aggiungi Corso

La figura 6 mostra l' interfaccia aggiungi Corso, i campi richiesti sono:

- Id esame, un codice univoco che identifica l' esame
- Nome esame, il nome del' esame
- Nome proefessore che tiene in corso
- Corso di leurea, selezionabile tramite il menù a tendina, si possono scegliere i corsi precedentemente inseriti
- Numero di CFU riconosciuti per quell' esame

StudiSync - Segreteria Inoltra Prenotazione

?

×

Richieste date

Aggiorna

Matricola	Esame	Nome	Cognome	Azione
0124002584	MAT1	Vittorio	Picone	Fornisci Date
				Rifiuta
0124002584	PROG1	Vittorio	Picone	Fornisci Date
				Rifiuta
0124002584	MAT1	Vittorio	Picone	Fornisci Date
				Rifiuta
a	RET	Luca	Sito	Fornisci Date
				Rifiuta
a	PROG2	Luca	Sito	Fornisci Date
				Rifiuta

Figure 7: Richiesrte date

6.3.4 Gestisci Richieste Date

la figura 7 mostra l' interfaccia "richieste date" Qui sono riportate tutte le richieste di date effettuate dagli studenti, per ogni richiesta è possibile:

- Fornire le date: Il server si occuperà di fornire le date dell' esame richiesto allo studente, che potrà visualizzarle dalla sua interfaccia
- Rifiutare: lo studente non otterrà le date e potrà visualizzare la sua proposta come "Rifiutata"

A prescindere dalla scelta, la riga con la richiesta verrà eliminata dalla tabella e verrà mostrato un messaggio in base all' esito dell' elaborazione

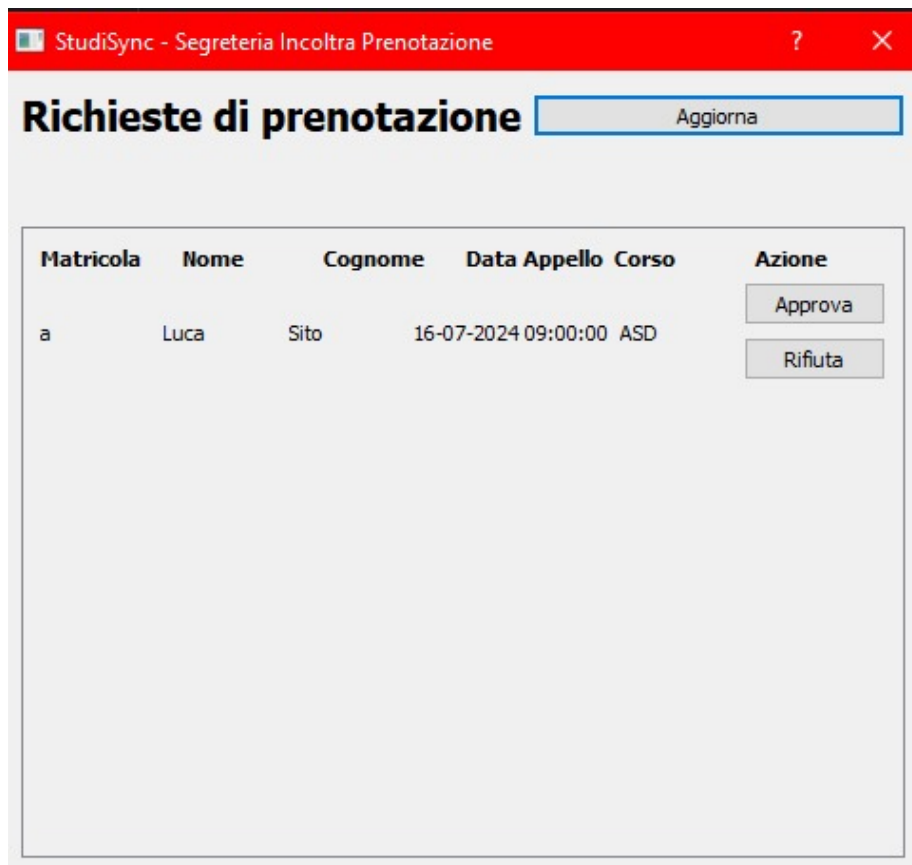


Figure 8: Richieste Prenotazioni

6.3.5 Gestisci Prenotazioni Appelli

la figura 8 mostra l' interfaccia "richieste prenotazione" Qui sono riportate tutte le richieste di prenotazione effettuate dagli studenti, per ogni richiesta è possibile:

- Approvare: Lo studente visualizzerà lo stato della sua richiesta come "evasa" e quindi è a tutti gli effetti prenotato all' esame
- Rifiutare: Lo studente vedrà dalla sue interfaccia "Respinta" e dovrà effettuare un'altra richiesta per essere accettato.

A prescindere dalla scelta, la riga con la richiesta verrà eliminata dalla tabella e verrà mostrato un messaggio in base all' esito dell' elaborazione

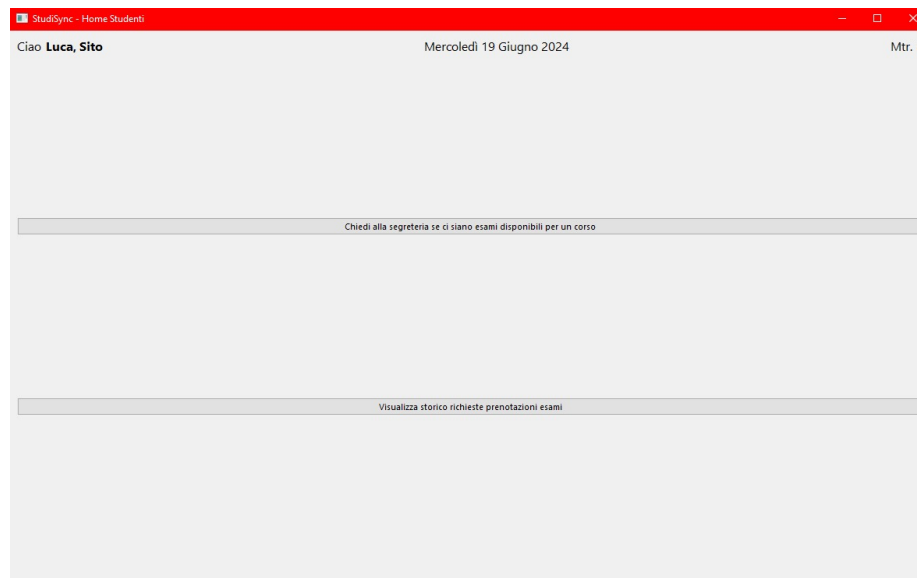


Figure 9: Enter Caption

6.4 Studente

Quando un utente studente effettua il login, viene mostrata questa interfaccia dove è possibile accedere a tutte le funzionalità

StudiSync - Students Richiesta Date Esami

Esami relativi al corso di laurea: **0124 - Informatica Triennale**

Seleziona Corso: [MAT1] Matematica 1 - 6 CFU

Invia Richiesta

Storico richieste

ID Richiesta	Corso	Stato	Azione
7	RET	In attesa	Mostra date
9	PROG2	In attesa	Mostra date
10	MAT1	Evasa	Mostra date
11	MAT1	Accetta	Mostra date
12	ASD	Evasa	Mostra date

☒ Tutte
 ☐ In attesa
 ☐ Accettate
 ☐ Rifiutate
 ☐ Evase

Aggiorna Storico

Figure 10: Richiedi date

6.4.1 Richiedi date

La figura 10 mostra l' interfaccia "richiedi date", dove è possibile fare richiesta alla segreteria per sapere le date disponibili per un determinato esame, per fare richiesta basta:

1. selezionare l' esame interessato tramite il menù a tendina
2. Fare click su invia richiesta

E' possibile inoltre visualizzare uno storico di tutte le richieste inviate in precedenza. Le richieste possono avere i seguenti stati:

- In attesa: la segreteria ha ricevuto la richiesta ma ancora deve approvarla/respingerla
- Accettata: la segreteria ha accettato la richiesta e sono state fornite le date di appello per quell' esame

- Rifiutata: la segreteria ha respinto la richiesta
- Evasa: la segreteria ha accettato non solo la richiesta di date ma anche la prenotazione ad un appello in una determinata data

E' possibile filtrare le richieste per status

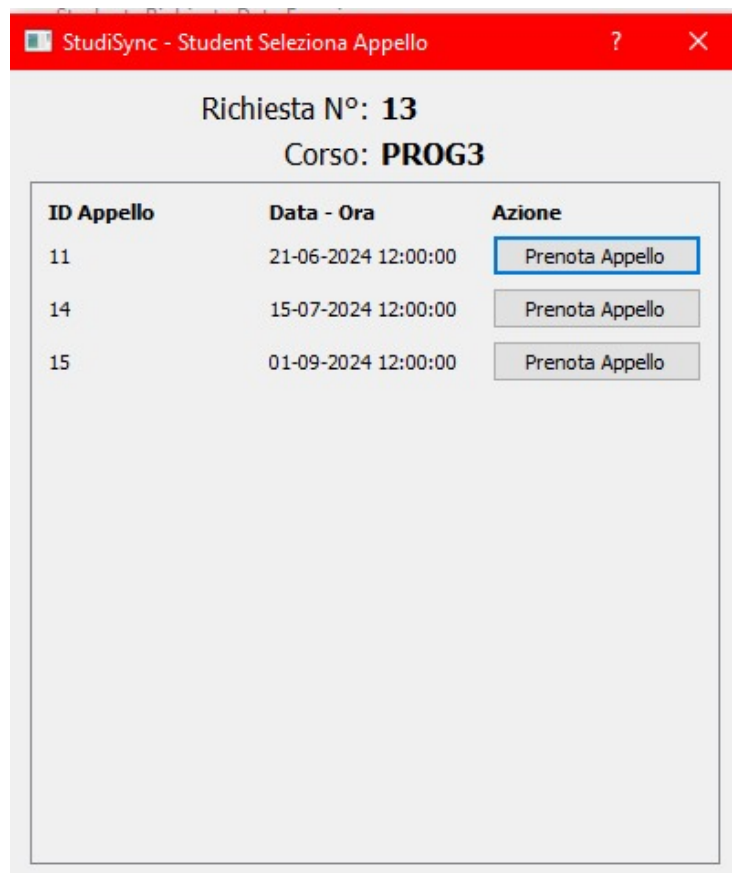


Figure 11: Richiedi Prenotazione

6.4.2 Richiedi prenotazione

In figura 11 viene mostrata l' interfaccia "Richiedi prenotazione", questa interfaccia può essere raggiunta cliccando su "mostra date" in figura 10, dopo che la segreteria ha fornito le date richieste per un determinato esame. Per prenotarsi all' appello basta cliccare su "Prenota appello" sulla riga che si desidera in base alla data dell' appello. In seguito la segreteria dovrà accettare/riufutare la richiesta di prenotazione

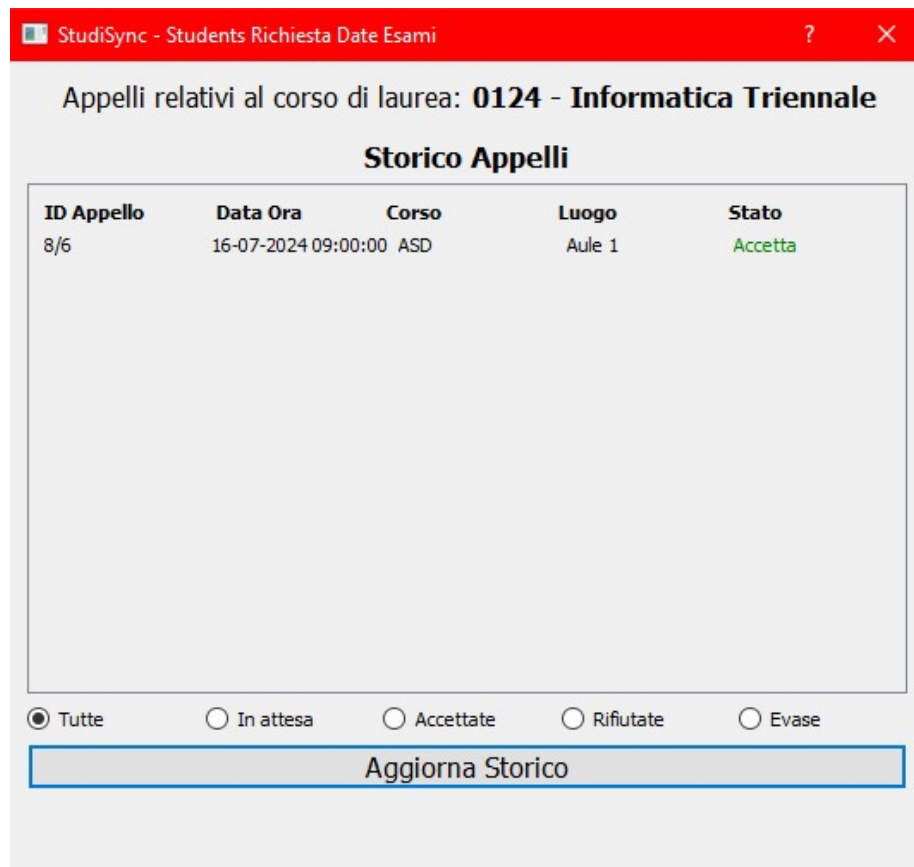


Figure 12: Storico Prenotazioni

6.4.3 Visualizza Storico Prenotazioni

In figura 12 viene mostrata l'interfaccia "Storico Prenotazioni" dove è possibile vedere lo storico delle prenotazioni passate con tanto di stato, è anche possibile filtrare per stato

7 Database

Il sistema utilizzato non rispecchia completamente la definizione di database, ma comunque vengono predisposte delle procedure per un'accesso rapido ai dati organizzati su dei file CSV, ai quali solamente il server può accedere sia in lettura che in scrittura.

Visto l'ambiente di programmazione concorrente l'accesso ai file avviene in una sezione critica tra i processi figli del server, solo le operazioni di scrittura avvengono in sezione critica.

La mutua esclusione è stata implementata tramite la classe `Lock()` presente nella libreria nativa di python 3.11 "multiprocessing"; per ottenere la mutua esclusione:

1. Si instancia la classe `Lock`: `locker = multiprocessing.Lock`
2. Si utilizza `acquire` all' inizio della sezione critica: `locker.acquire()`
3. Si scrive il codice nella sezione critica
4. Si rilascia: `locker.release()`

Ergo `Lock()` funge da mutex garantendo anche la mutua esclusione