Relazione Reti Di Calcolatori 2023/2024

Leonardo Campana 0124002568
Adriano Guastaferro 0124002690

Codice Gruppo: naif3kibr3

June 19, 2024

Indice

1 Descrizione Del Progetto

Il nostro progetto consiste nello sviluppare un'applicazione client/server parallela per la gestione degli esami universitari. Per realizzare questa task abbiamo deciso di utilizzare Python, il quale offre una serie di moduli che ci hanno guidato e fornito funzionalità che che ci hanno permesso di sviluppare il progetto; qui di seguito una breve descrizione di quest'ultimi.

1.1 Moduli Utilizzati

- **Modulo Socket**
  Il modulo socket in Python fornisce l'accesso alle interfacce di rete di basso livello. Viene utilizzato per implementare la comunicazione tra computer attraverso una rete. Questo modulo consente di creare socket server e client, gestire connessioni, inviare e ricevere dati su TCP (Transmission Control Protocol) o UDP (User Datagram Protocol). È fondamentale per lo sviluppo di applicazioni di rete che richiedono l'interazione tra nodi diversi.

- **Modulo Tkinter**
  Tkinter è il modulo standard di Python per la creazione di interfacce grafiche (GUI). È una libreria versatile che consente di sviluppare applicazioni con finestre, pulsanti, etichette, campi di testo e altri widget GUI. Tkinter è incluso nella distribuzione standard di Python, rendendolo facilmente accessibile per creare interfacce utente interattive.

- **Modulo Threading**
  Il modulo thread, ora sostituito da threading nelle versioni moderne di Python, permette la creazione e la gestione di thread separati all'interno di un'applicazione. I thread consentono l'esecuzione parallela di codice, migliorando la reattività e le prestazioni dell'applicazione, soprattutto quando si gestiscono operazioni di I/O o compiti che richiedono tempo. Nel nostro contesto , quest'ultimi sono stati utilizzati per gestire più connessioni simultanee senza bloccare l'intero programma.

- **Modulo Json**
  Il modulo json in Python permette di lavorare con i dati nel formato JSON (JavaScript Object Notation), che è uno standard aperto per la trasmissione di dati strutturati tra un server e un client. Il modulo fornisce metodi per convertire dati Python in stringhe JSON e viceversa.L'utilizzo di questo modulo ci ha consentito una facile lettura/scrittura dei dati inerenti a studenti, prenotazioni ed esami.

- **Modulo Datetime**
  Il modulo datetime in Python fornisce classi per la gestione di date, orari e operazioni correlate. È utile per la manipolazione, l'analisi e la visualizzazione di informazioni temporali all'interno dei programmi Python.

- **Modulo Functools**
  Il modulo functools in Python offre vari strumenti per la programmazione funzionale. Questo modulo fornisce funzionalità che permettono di lavorare con funzioni in modo più flessibile e potente.Uno degli strumenti più utili di functools è la funzione partial. La funzione partial consente di "parzializzare" un'altra funzione, cioè di fissare alcuni dei suoi argomenti, creando una nuova funzione con questi argomenti già predefiniti.

2 Descrizione e schema dell'architettura

Il progetto si basa su un sistema distribuito composto da quattro componenti principali, ciascuno implementato in Python.Per la comunicazione tra i vari nodi, abbiamo utilizzato le socket tramite il modulo socket di Python. Abbiamo creato le socket impiegando i parametri di default AF-INET per la famiglia di indirizzi (IPv4) e SOCK-STREAM per il tipo di socket (TCP). Questa configurazione ha permesso di stabilire una connessione affidabile e orientata alla connessione tra i dispositivi, facilitando così lo scambio di dati in modo efficiente.

- **Studente**
  Lo studente accede tramite un'interfaccia di login al sistema. Qui può visualizzare gli esami per i quali è possibile effettuare una prenotazione. Per fare ciò egli comunica con il server della segreteria.

- **Server Segreteria**
  Il server della segreteria offre tutte le funzionalità necessarie affinchè l'utente possa:

  1. Visualizzare gli esami
  2. Visualizzare le date disponibili per gli esami
  3. Inoltare la richiesta di prenotazione al server dell'università instaurando una connessione con quest'ultimo.
  4. Fornire all'utente il numero progressivo associato alla sua prenotazione

- **Client Segreteria**
  Il client della segreteria permette, tramite interfaccia grafica, di poter inserire dei nuovi esami sul server dell'università, i quali saranno subito disponibili per la prenotazione dell'utente.

- **Server Università (uni.py)**
  Il server dell'università riceve sia le richieste per aggiungere nuovi esami da parte del client della segreteria sia le richieste di prenotazione dello studente tramite il server della segreteria. Quest'ultime vengono appositamente gestite tramite una serie di metodi che verranno visti più avanti.

Per dare una visione d'insieme riassuntiva, qui di seguito viene riportato un diagramma Use-Case affinchè siano più chiare le operazioni che i vari attori possono svolgere all'interno del sistema.

Nota:All'interno del diagramma l'attore Segreteria non presenta la divisione in parte client e parte server.

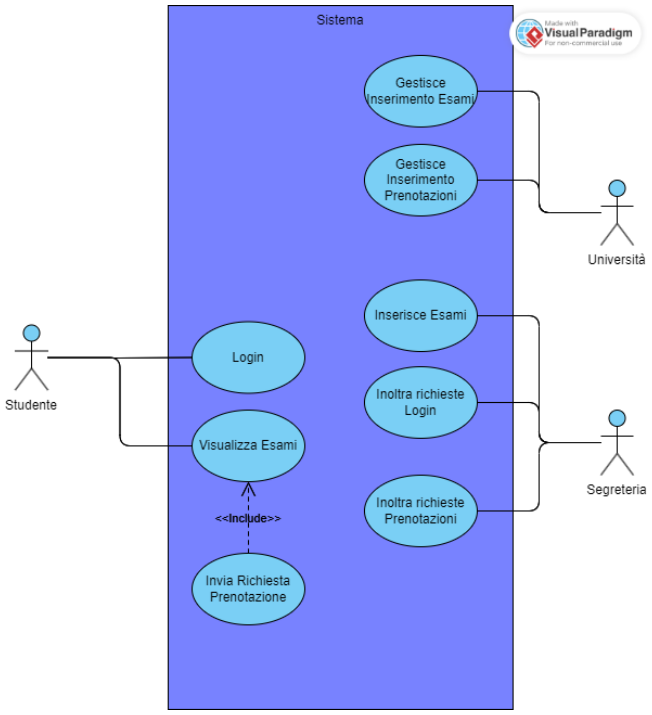

Figure 1: Diagramma Use-Case

3 Dettagli implementativi del client/server

- **Studente**
  Lo studente tramite una socket si connette al server della segreteria. Al momento della connessione viene visualizzata la finestra di Login, tramite la quale l'utente invia le proprie credenziali e può accedere ad una interfaccia personale.Le credenziali arrivano al server della segreteria per poi essere inoltrare al server dell'università che effettua un controllo per verificare l'esistenza dell'utente.
  Se l'esito della richiesta è positivo allora l'utente accede alla propria pagina altrimenti viene visualizzata una finestra d'errore.
  NOTA: Le credenziali per accedere sono presenti all'interno del file **studenti.json**.

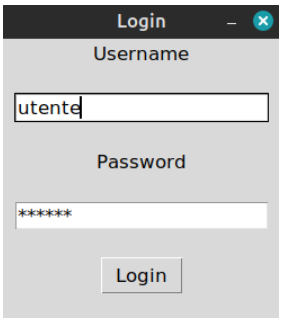

Figure 2: Schermata di Login

- **Server-Segreteria**
  La **parte server della segreteria** è il fulcro dell'intero progetto. Essa funge da **intermediario tra lo studente e il server dell'università**. Le richieste di prenotazioni e di login che giungono a questo nodo vengono poi inoltrate al server dell'università tramite una Socket. Un'altra funzione principale è quella di recuperare i dati inerenti agli esami e di mostrarli allo studente all'interno della sua interfaccia personale.

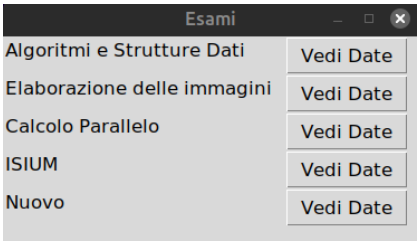

Figure 3: Tabella Degli Esami

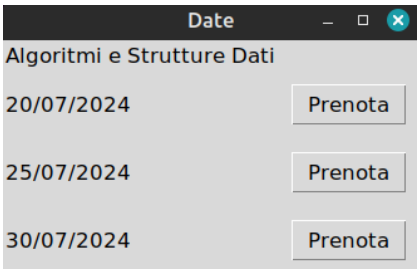

Figure 4: Data Selezionabile

- **Client Segreteria**
  La parte client della segreteria è connesso anch'esso al server dell'università. La scelta di dividere la segreteria in 2 file separati è stata dettata da una necessità di migliore organizzazione del lavoro e di una migliore leggibilità del codice.
  Il client della segreteria presenta una GUI che permette di aggiungere dei nuovi esami con le relative date.

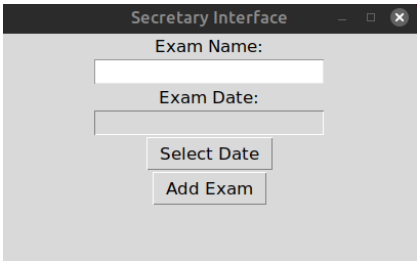

Figure 5: Tabella Degli Esami

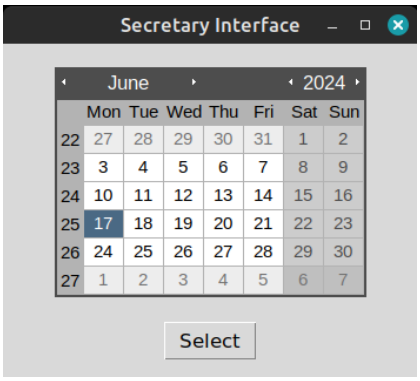

Figure 6: Date Disponibili

- **Università**
  Il Server università è l'ultimo endpoint.Il suo scopo principale, come descritto in precedenza, è quello di gestire sia l'inserimento di prenotazione all'interno del file prenotazioni.json sia l'inserimento di nuovi esami all'interno di un altro file denominato esami.json. Altra funzionalità del server università è quella di fornire all'utente il numero progressivo della prenotazione a lui associata.

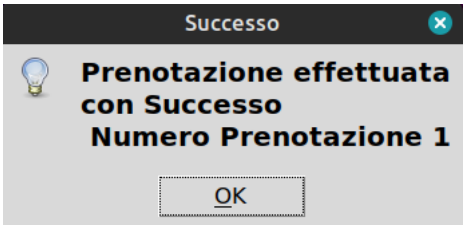

Figure 7: Numero progressivo prenotazione

4 Parti Rilevanti Codice Sviluppato

All'interno della seguente immagine mostriamo le parti rilevanti di ciascuna entità del progetto.

4.1 Studente

Qui di seguito possiamo notare i metodi principali della classe StudentClient, le quali definiscono una serie di oeprazioni base che l'utente può compiere:

```python
class StudentClient:

    def __init__(self, host = "localhost", port = 10001):
        # Inizializzazione del client
        self.host = host
        self.port = port
        #creazione socket: AF_INET specifica la tipologia di indirizzi (in questo caso IPV4), sock_Stream indica il protocollo TCP
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Crea un socket TCP

    def connect_to_server(self):
        # Connessione al server
        try:
            self.client_socket.connect((self.host,self.port))
            print("Connessione Stabilita con Server Segreteria")
        except socket.error as errore:
            print(f"Connessione non riuscita\nErrore:{errore}")
            exit()

    def send_login_request(self,username,password):
        # Invia la richiesta di login al server
        credentials = {"username":username, "password":password,"type":"login"} # Dati delle credenziali in formato dizionario
        self.client_socket.send(json.dumps(credentials).encode('utf-8')) # Invia i dati al server codificati in JSON

        response = json.loads(self.client_socket.recv(4096).decode('utf8')) # Riceve e decodifica la risposta del server
        return response


    def send_exam_table_request(self):
        # Invia la richiesta per visualizzare gli esami e le date ad essi associati
        request = {"type":"viewExams"} # Tipo di richiesta
        self.client_socket.send(json.dumps(request).encode('utf-8')) # Invia la richiesta al server
        response = json.loads(self.client_socket.recv(4096).decode('utf-8')) # Riceve e decodifica la risposta del server
        return response

    def send_exam_booking_request(self,esame,matricola,data):
        # Invia la richiesta per prenotare un esame
        booking_data = {"matricola":matricola,"esame":esame,"data":data,"type":"bookExam"} # Dati di prenotazione
        self.client_socket.send(json.dumps(booking_data).encode('utf-8')) # Invia i dati al server
        response = json.loads(self.client_socket.recv(4096).decode('utf-8')) # Riceve e decodifica la risposta del server
        return response


    def disconnect_from_server(self):
        # Disconnessione dal server
        self.client_socket.close()
        print("Connessione con il server chiusa")
```

Figure 8: Codice relativo al client dello Studente

4.2 Segreteria

4.2.1 Parte Server della Segreteria

Le parti fondamentali della parte server della segreteria sono quelle che getiscono le richieste degli studenti e il loro eventuale inoltro. Se la richiesta dello studente è quella di visualizzare esami e date allora la richiesta viene gestita dal metodo handle-student-connection tramite verifica con l'if, in caso contrario la richiesta è inoltrata all'università.

```python
def handle_student_connection(self, client_socket):
    # Gestisce la connessione con uno studente
    try:
        while True:
            request = client_socket.recv(4096).decode('utf-8')  # Riceve i dati dal client
            if not request:
                break  # Esce dal ciclo se non ci sono più dati
            request_data = json.loads(request)  # Decodifica i dati ricevuti in formato JSON

            # Inoltra i dati al server dell'università se necessario
            #se la richiesta dello studente è quella di visualizzare gli esami allora viene gestita direttamente da questo server
            if request_data["type"] == "viewExams":
                #print("Me ne occupo io") frase per debug
                seg_management = self.get_exams_data()  # Ottiene i dati degli esami
                seg_reponse = json.dumps(seg_management)  # Codifica i dati degli esami in JSON
                client_socket.sendall(seg_reponse.encode('utf-8'))  # Invia i dati al client
            else:
                uni_response = self.forward_request_to_university_server(request_data)  # Inoltra la richiesta al server dell'università
                rispostaUni = json.dumps(uni_response)  # Codifica la risposta del server dell'università in JSON
                print(rispostaUni)
                client_socket.sendall(rispostaUni.encode('utf-8'))  # Invia la risposta al client
    finally:
        client_socket.close()  # Chiude la connessione con il client
```

Figure 9: Metodo che si occupa della gestione delle richieste del client.

```python
def forward_request_to_university_server(self, request_data):
    # Inoltra la richiesta al server dell'università e ottiene la risposta
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as uni_socket:
        uni_socket.connect((self.uni_server_host, self.uni_server_port))  # Connette al server dell'università
        uni_socket.sendall(json.dumps(request_data).encode('utf-8'))  # Invia i dati al server dell'università
        response = uni_socket.recv(4096).decode('utf-8')  # Riceve la risposta dal server dell'università
        return json.loads(response)  # Decodifica la risposta in formato JSON
```

Figure 10: Metodo che inoltra la richiesta al server dell'Università.

4.2.2 Parte Client della Segreteria

L'inserimento degli esami è gestito tramite i seguenti metodi:

```
def add_exam(self, exam_name, date):
    # Invia una richiesta al server per aggiungere un nuovo esame con nome e data specificati
    request = { 'exam_name': exam_name, 'dates': date,'type': 'addExam'}
    self.client_socket.sendall(json.dumps(request).encode('utf-8'))
    response = self.client_socket.recv(1024).decode('utf-8')
    return json.loads(response)
```

Figure 11: Metodo per aggiungere esami.

```
def add_exam(self):
    # Invia una richiesta al server per aggiungere un nuovo esame
    exam_name = self.exam_name_entry.get()  # Ottiene il nome dell'esame dall'entry
    exam_date = self.exam_date_entry.get()  # Ottiene la data dell'esame dall'entry

    # Controlla se la data è vuota
    if not exam_date.strip():
        messagebox.showerror("Error", "The exam date cannot be empty.")  # Mostra un messaggio di errore
        return  # Interrompe l'esecuzione del metodo

    # Procede con l'invio della richiesta al server se la data non è vuota
    response = self.secretary_client.add_exam(exam_name, exam_date)  # Invia la richiesta e ottiene la risposta

    if response["status"] == "success":
        messagebox.showinfo("Fatto","Esame aggiunto correttamente")
    else:
        messagebox.showerror("Errore","Esame non Aggiunto")
```

Figure 12: Gestione response esami.

4.3 Università

Le parti essenziali del codice dell'università sono descritte da handle-student-client che agisce allo stesso modo di quello della segreteria e poi vi è il metodo process-request, il quale si occupa del recupero e scrittura dei dati all'interno dei vari file.json che contengono dati inerenti agli esami,studenti e prenotazioni.

```
def handle_client_connection(self, client_socket):
    while True:
        try:
            request = client_socket.recv(4096).decode('utf-8')  # Riceve i dati dal client
            if not request:
                break
            request_data = json.loads(request)  # Decodifica i dati JSON
            response_data = self.process_request(request_data)  # Processa la richiesta
            client_socket.sendall(json.dumps(response_data).encode('utf-8'))  # Invia la risposta al client
        except Exception as e:
            print(f"Errore {e}")
            break
    client_socket.close()  # Chiude la connessione del client
```

Figure 13: Gestione connessione Client.

```
#metodo all'interno del quale vengono gestite tutte le tipologie di richieste effettuate dal client
def process_request(self, request):
    try:
        if 'type' not in request:
            return {'error': "Request type is missing"}
        #login studente
        if request['type'] == 'login':
            print("richiesta login in verifica...\n")
            with open("studenti.json", 'r') as f:
                users = json.load(f)  # Carica i dati degli studenti
                response = {"status": "fail", "matricola": ""}
                for user in users:
                    if user["username"] == request["username"] and user["psw"] == request["password"]:
                        print("Login Corretto!\n")
                        response["status"] = "success"
                        response["matricola"] = user["matricola"]
                        break
                return response
        #aggiunta esami
        elif request['type'] == "addExam":
            response = self.add_Exam(request)  # Processa la richiesta di aggiunta esame
            return response
        #prenotazioni esami
        elif request['type'] == "bookExam":
            response = self.book_Exam(request)  # Processa la richiesta di prenotazione esame
            return response

    except Exception as e:
        print(f"Errore in gestione response: {e}")
```

Figure 14: Gestione tipolgie richieste.

5 Manuale Utente

Per avviare il progetto, è necessario eseguire i file Python nell'ordine corretto per garantire il corretto funzionamento del sistema. Di seguito sono riportati i passaggi dettagliati:

1. **Effettuare download Progetto**
   Scarica o clona il progetto dal seguente link:
   https://github.com/LeoGH02/SviluppoReti.

2. **Apertura del Terminale**
   Per eseguire i file è necessaria l'apertura di 4 terminali, uno per ogni file python.

3. **Verificare versione installata di Python**
   In caso in cui il progetto è eseguito in ambiente Unix è consigliato verificare la versione di Python.
   Nel caso di Python 3 allora:
   `python3 nomefile.py`

   Se invece è installata solo una versione di Python 2, il comando sarà:

   `python nomefile.py`
   Quest'ultimo comando vale anche per i sistemi Windows.

4. **Avvio dei file in ordine**
   Seguire questi passaggi per avviare i file nell'ordine corretto:

   - **uni.py**: Questo file deve essere eseguito per primo. Configura l'ambiente di base e inizializza le variabili e le risorse necessarie per l'intero progetto.

     `python3 uni.py`

   - **server_segreteria.py**: Una volta che il client della segreteria è in esecuzione, avviare `server_segreteria.py`. Questo script è responsabile della gestione delle richieste ricevute dai client e dell'inoltro delle risposte.

     `python3 server_segreteria.py`

   - **client_segreteria.py**: Dopo aver avviato `uni.py`, eseguire il file `client_segreteria.py`. Questo script rappresenta il client della segreteria che invia richieste al server.

     `python3 client_segreteria.py`

   - **studente.py**: Infine, eseguire il file `studente.py`, che rappresenta l'interfaccia del client studente. Questo script permette agli studenti di interagire con il sistema tramite il server della segreteria.

     `python3 studente.py`
     NOTE FINALI
     Per effettuare un login di prova inserire le seguenti credenziali:

username:leonardo
psw:asd123
Per vedere il numero progressivo di prenotazioni, essendo che vi sono dei controlli che impediscono ad uno studente di prenotarsi 2 volte per lo stesso esame(con la stessa data), è necessario effettuare il login tramite un altro account.

Seguendo questi passaggi nell'ordine indicato, il progetto sarà avviato correttamente e tutti i componenti della rete comunicheranno come previsto.