Università degli Studi di Messina
Dipartimento di Scienze Matematiche e Informatiche, Scienze Fisiche e Scienze della Terra

Corso di Laurea Triennale in Informatica

Gestione da remoto di un sistema bibliotecario

Francesco Montecucco M. 531620
Francesco Bartolomeo M. 528405

Professore: Roberto Marino

Anno Accademico 2023/2024

Contents

Stato dell'arte

Nell'era digitale moderna, la gestione efficiente delle risorse è diventata un elemento cruciale per molte istituzioni, tra cui le biblioteche. Con l'avvento di Internet e delle tecnologie di rete, è ora possibile amministrare sistemi complessi da remoto, rendendo i processi più fluidi e accessibili.

Questo documento descriverà i dettagli del progetto, inclusi i requisiti, l'architettura del sistema, le tecnologie utilizzate, le implementazioni specifiche del codice e i risultati ottenuti. L'intento è di fornire una panoramica completa del processo di sviluppo e delle soluzioni adottate per affrontare le sfide incontrate durante il progetto.

In questo capitolo, analizziamo le tecnologie e le metodologie attualmente utilizzate per la gestione da remoto di sistemi bibliotecari.

1.1 Sistemi Bibliotecari

Questo sistema bibliotecario è progettato per gestire in modo efficiente la catalogazione, il prestito e la restituzione di libri. Utilizzando un database relazionale per memorizzare informazioni sugli utenti, sui libri e sulle transazioni.

1.2 Paradigma REST

Il paradigma REST (Representational State Transfer) è uno stile architettonico utilizzato per la progettazione di sistemi distribuiti. In un'architettura RESTful, le risorse sono identificate da URI e le operazioni su queste risorse sono eseguite tramite metodi HTTP standard come GET, POST, PUT e DELETE. Questo approccio facilita l'integrazione e la comunicazione tra sistemi eterogenei.

1.3 Multithreading in Python

Il multithreading è una tecnica che consente l'esecuzione simultanea di più thread (flussi di esecuzione) all'interno di un singolo processo. In Python, il modulo `threading` fornisce strumenti per creare e gestire thread. Questo è particolarmente utile per le applicazioni client/server, dove è necessario gestire più connessioni simultanee.

1.4 Database MySQL

MySQL è un sistema di gestione di database relazionali (RDBMS) open source, ampiamente utilizzato in applicazioni web. Supporta il linguaggio SQL (Structured Query Language) per la gestione dei dati e offre funzionalità avanzate come transazioni, integrità referenziale e procedure memorizzate.

Descrizione del problema

Realizzare in Python un programma multithreading client/server per la gestione da remoto di un sistema bibliotecario secondo il paradigma REST e usando MYSQL.

2.1 Obiettivo

L'obiettivo principale di questo progetto è creare un sistema che permetta agli utenti di interagire con la biblioteca da remoto, consentendo operazioni come:

- Ricerca di libri
- Verifica della disponibilità
- Prestito di un libro
- Restituzione di un libro

Implementazione

3.1 Client

3.1.1 Descrizione del Codice

Il codice Python è strutturato in diverse sezioni chiare:

3.1.2 URL base e Variabili di Stato

Il file inizia definendo l'URL base del server locale a cui il client farà riferimento per le richieste HTTP e le variabili di stato per gestire lo stato del login dell'utente, il ruolo e l'id di quest'ultimo.

3.1.3 Funzioni per la Gestione della Biblioteca

Il codice include funzioni per gestire le operazioni principali sulla biblioteca, come:

- Mostrare a schermo di tutti i libri (`get_libri()`)
- Mostrare a schermo i dettagli di un singolo libro (`get_libro()`)
- Aggiunta di un nuovo libro al Db (`add_libro()`)
- Aggiornamento informazioni di un libro presente nel Db (`update_libro()`)
- Eliminazione di un libro dal Db (`delete_libro()`)
- Registrazione di un nuovo prestito (`new_prestito()`)
- Restituzione di un libro (`restituisci_libro()`)
- Recupero della disponibilità di un libro (`get_disponibilita()`)
- Recupero dei prestiti (passati/correnti) di un utente specificato (`get_user_prestiti()`)

Nello specifico:

Mostrare a schermo tutti i libri (`get_libri()`)

```
def get_libri():
response = requests.get(f"{base_url}/libri")
if response.status_code == 200:
    libri = response.json()
    print("Libri presenti nella biblioteca:")
```

```
    for libro in libri:
        disponibilita = 'SI' if libro['disponibilita'] == 'YES' else 'NO'
        print(f"ID: {libro['id_libro']}, Titolo: {libro['titolo']}, Autore:
        {libro['autore']}, Anno: {libro['anno']}, Disponibilità: {disponibilita}")
else:
    print("Errore nella richiesta GET")
```

Mostrare a schermo i dettagli di uno specifico libro (`get_libro()`)

```
def get_libro(libro_id):
response = requests.get(f"{base_url}/libri/{libro_id}")
if response.status_code == 200:
    libro = response.json()
    disponibilita = 'SI' if libro['disponibilita'] == 'YES' else 'NO'
    print(f"ID: {libro['id_libro']}, Titolo: {libro['titolo']}, Autore:
    {libro['autore']}, Anno: {libro['anno']}, Disponibilità: {disponibilita}")
else:
    print("Errore nella richiesta GET")
```

Aggiunta di un nuovo libro al Db (`add_libro()`)

```
def add_libro(titolo, autore, anno):
if user_role != 'Amministratore':
    print("\n---------------------------------------------------------------")
    print("Accesso negato: solo gli amministratori possono aggiungere libri.")
    print("---------------------------------------------------------------")
    return
nuovo_libro = {'titolo': titolo, 'autore': autore, 'anno': anno}
response = requests.post(f"{base_url}/libri", json=nuovo_libro)
if response.status_code == 201:
    libro = response.json()
    print("\n-------------------------")
    print("Libro aggiunto con successo:")
    print("-------------------------")
    print(f"ID: {libro['id_libro']}, Titolo: {libro['titolo']}, Autore:
    {libro['autore']}, Anno: {libro['anno']}")
else:
    print("Errore nella richiesta POST")
```

Aggiornamento delle informazioni di libro presente nel Db (`update_libro()`)

```
def update_libro(libro_id, titolo, autore, anno):
if user_role != 'Amministratore':
    print("\n---------------------------------------------------------------")
    print("Accesso negato: solo gli amministratori possono aggiornare libri.")
    print("---------------------------------------------------------------")
    return
aggiorna_libro = {'titolo': titolo, 'autore': autore, 'anno': anno}
response = requests.put(f"{base_url}/libri/{libro_id}", json=aggiorna_libro)
if response.status_code == 200:
```

```
    libro = response.json()
    print("\n----------------------------")
    print("Libro aggiornato con successo:")
    print("----------------------------")
    print(f"ID: {libro['id_libro']}, Titolo: {libro['titolo']}, Autore:
    {libro['autore']}, Anno: {libro['anno']}")
else:
    print("Errore nella richiesta PUT")
```

Eliminazione di un libro dal Db (`delete_libro()`)

```
def delete_libro(libro_id):
if user_role != 'Amministratore':
    print("\n-----------------------------------------------------------------")
    print("Accesso negato: solo gli amministratori possono eliminare libri.")
    print("-----------------------------------------------------------------")
    return
response = requests.delete(f"{base_url}/libri/{libro_id}")
if response.status_code == 200:
    print("\n----------------------------")
    print("Libro eliminato con successo")
    print("----------------------------")
else:
    print("Errore nella richiesta DELETE")
```

Registrazione di un nuovo prestito (`new_prestito()`)

```
def new_prestito(libro_id):
disponibile = get_disponibilita(libro_id)
if disponibile:
    data_prestito = datetime.now().strftime('%Y-%m-%d')
    nuovo_prestito = {'id_utente': user_id, 'id_libro': libro_id,
    'data_prestito': data_prestito}
    response = requests.post(f"{base_url}/prestiti", json=nuovo_prestito)
    if response.status_code == 201:
        print("\n-------------------------------------------------------")
        print("Prestito registrato con successo:")
        print(f"ID Utente: {user_id}, ID Libro: {libro_id}, Data Prestito:
        {data_prestito}")
        print("-------------------------------------------------------")
    else:
        print("Errore nella richiesta di prestito")
else:
    print("\n-----------------------------------")
    print("Il libro richiesto non è disponibile")
    print("-----------------------------------")
```

Restituzione di un libro (`restituisci_libro()`)

```
def restituisci_libro(libro_id):
```

```
data_restituzione = datetime.now().strftime('%Y-%m-%d')
restituzione = {'id_utente': user_id, 'id_libro': libro_id,
'data_restituzione': data_restituzione}
response = requests.post(f"{base_url}/restituzioni", json=restituzione)

if response.status_code == 200:
    print("\n---------------------------------------------------------------")
    print("Restituzione registrata con successo:")
    print(f"ID Utente: {user_id}, ID Libro: {libro_id}, Data Restituzione:
    {data_restituzione}")
    print("---------------------------------------------------------------")
else:
    print("Errore nella richiesta di restituzione")
```

Recupero della disponibilità di un libro al prestito (`get_disponibilita()`)

```
def get_disponibilita(libro_id):
response = requests.get(f"{base_url}/libri/{libro_id}")
if response.status_code == 200:
    libro = response.json()
is_disponibile = 1 if libro['disponibilita'] == 'YES' else 0
return is_disponibile
```

Recupero di tutti i prestiti (passati/correnti) di un utente specificato (`get_user_prestiti()`)

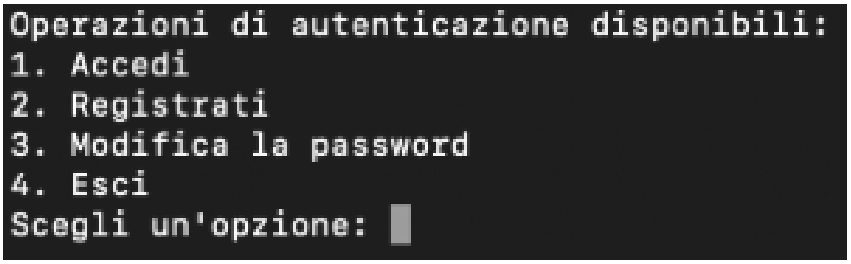
```
def get_user_prestiti(user_id):
response = requests.get(f"{base_url}/prestiti?id_utente={user_id}")
if response.status_code == 200:
    prestiti = response.json()
    print("Prestiti dell'utente:")
    for prestito in prestiti:
        print(f"ID Prestito: {prestito['id_prestito']}, ID Libro:
        {prestito['id_libro']}, Data Prestito: {prestito['data_prestito']},
        Data Restituzione: {prestito['data_restituzione']}")
else:
    print("Errore nella richiesta GET")
```

Ogni funzione gestisce le chiamate HTTP corrispondenti per interagire con il server.

3.1.4 Funzioni per la Gestione dell'Autenticazione

Sono definite anche funzioni per gestire l'autenticazione degli utenti:

- Login (`login()`)
- Registrazione (`register()`)
- Cambio password (`change_password()`)



Nello specifico:

Login (`login()`)

```
def login(username, password):
global logged_in, user_role, user_id
response = requests.post(f"{base_url}/login", json={'username': username, 'password':
if response.status_code == 200:
    data = response.json()
    logged_in = True
    user_role = data.get('ruolo', 'utente')
    user_id = data.get('id_utente')
    print("\n----------------------------")
    print("Login effettuato con successo")
    print(f"Ruolo dell'utente: {user_role}")
    print("----------------------------")
else:
    print("Errore nel login: credenziali non valide")
```

Register (`Register()`)

```
def register(username, password, confirm_password, pin):
global logged_in, user_role, user_id
if password != confirm_password:
    print("Errore: le password non coincidono")
    return
# Verifica del pin per il ruolo di amministratore
ruolo = 'utente'
if pin == '0000':  # Sostituire 'PIN_AMMINISTRATORE' con il pin reale
    ruolo = 'Amministratore'
try:
    response = requests.post(f"{base_url}/register", json={'username': username,
    'password': password, 'ruolo': ruolo, 'nome': nome, 'cognome': cognome})
    response.raise_for_status()
    if response.status_code == 201:
        data = response.json()
        logged_in = True
        user_role = data.get('ruolo', ruolo)
        user_id = data.get('id_utente')
        print("\n-----------------------------------------------------")
        print("Registrazione effettuata con successo e login automatico")
        print(f"Ruolo dell'utente: {user_role}")
```

```
        print("-------------------------------------------------------")
    else:
        print("Errore nella registrazione")
except requests.exceptions.HTTPError as e:
    if e.response.status_code == 409:
        print("Errore: Username già utilizzato")
    else:
        print(f"Errore HTTP {e.response.status_code}: {e.response.text}")
except requests.exceptions.ConnectionError as errc:
    print(f"Errore di connessione: {errc}")
except requests.exceptions.RequestException as err:
    print(f"Errore durante la richiesta: {err}")
```

Change_password (`change_password()`)

```
def change_password(username, old_password, new_password, confirm_new_password):
if new_password != confirm_new_password:
    print("Errore: le nuove password non coincidono")
    return
response = requests.post(f"{base_url}/change_password", json={'username':
username, 'old_password': old_password, 'new_password': new_password})
if response.status_code == 200:
    print("\n-------------------------------")
    print("Password modificata con successo")
    print("-------------------------------")
elif response.status_code == 404:
    print("Errore: Utente non trovato")
elif response.status_code == 400:
    print("Errore: La vecchia password non è corretta")
else:
    print("Errore nella modifica della password")
```

Queste funzioni gestiscono le chiamate per il login, la registrazione e il cambio password tramite il server.

3.1.5 Funzioni Ausiliarie

Vi sono anche funzioni ausiliarie come `clear_screen()` per pulire la console.

Clear_screen (`clear_screen()`)

```
    def clear_screen():
    os.system('cls' if os.name == 'nt' else 'clear')
```

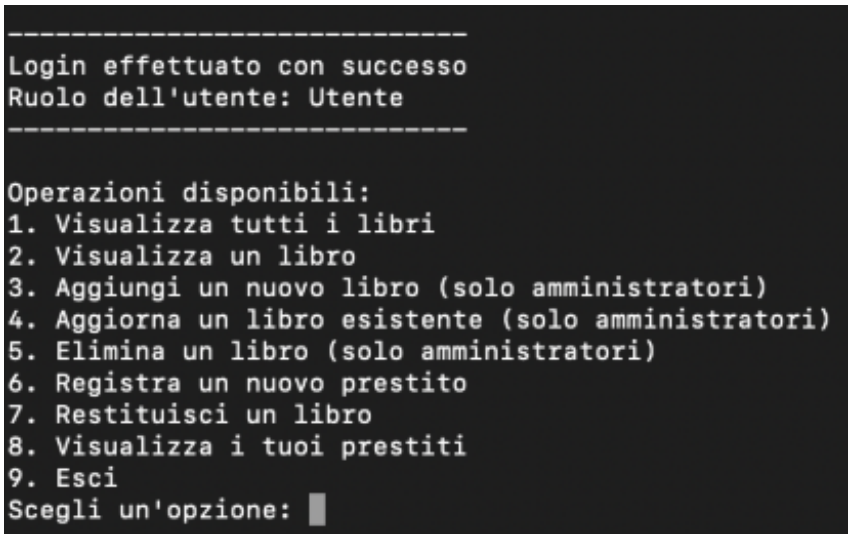
3.1.6 Gestione del Menu Principale

Il codice include un loop principale che gestisce le operazioni dell'applicazione a seconda dello stato di login dell'utente. Offre un menu interattivo che permette di selezionare le diverse operazioni disponibili a seconda del ruolo dell'utente.

Prima del login:

```
if not logged_in
print("\nOperazioni di autenticazione disponibili:")
print("1. Accedi")
print("2. Registrati")
print("3. Modifica la password")
print("4. Esci")

scelta = input("Scegli un'opzione: ")

if scelta == '1':
    username = input("Inserisci il nome utente: ")
    password = getpass.getpass("Inserisci la password: ")
    clear_screen()
    login(username, password)
elif scelta == '2':
    nome = input("Inserisci il tuo nome: ")
    cognome = input("Inserisci il tuo cognome: ")
    username = input("Inserisci il nome utente: ")
    password = getpass.getpass("Inserisci la password: ")
    confirm_password = getpass.getpass("Conferma la password: ")
    pin = input("Inserisci il pin per il ruolo di amministratore
    (lascia vuoto se utente): ")
    clear_screen()
    register(username, password, confirm_password, pin)
elif scelta == '3':
    username = input("Inserisci il nome utente: ")
    old_password = getpass.getpass("Inserisci la vecchia password: ")
    new_password = getpass.getpass("Inserisci la nuova password: ")
    confirm_new_password = getpass.getpass("Conferma la nuova password: ")
    clear_screen()
    change_password(username, old_password, new_password, confirm_new_password)
elif scelta == '4':
    clear_screen()
    print("Uscita dal programma.")
    break
else:
    clear_screen()
    print("Scelta non valida. Riprova.")
```

Dopo il login:




```
else:
print("\nOperazioni disponibili:")
print("1. Visualizza tutti i libri")
print("2. Visualizza un libro")
print("3. Aggiungi un nuovo libro (solo amministratori)")
print("4. Aggiorna un libro esistente (solo amministratori)")
print("5. Elimina un libro (solo amministratori)")
print("6. Registra un nuovo prestito")
print("7. Restituisci un libro")
print("8. Visualizza i tuoi prestiti")
print("9. Esci")

scelta = input("Scegli un'opzione: ")

if scelta == '1':
    clear_screen()
    get_libri()
elif scelta == '2':
    clear_screen()
    libro_id = input("Inserisci l'ID del libro: ")
    get_libro(libro_id)
elif scelta == '3':
    clear_screen()
    if user_role == 'Amministratore':
        titolo = input("Inserisci il titolo del libro: ")
        autore = input("Inserisci l'autore del libro: ")
        anno = input("Inserisci l'anno del libro: ")
        add_libro(titolo, autore, anno)
    else:
        print("\n-----------------------------------------------------------------")
        print("Accesso negato: solo gli amministratori possono aggiungere libri.")
        print("-----------------------------------------------------------------")
elif scelta == '4':
```

```
    clear_screen()
    if user_role == 'Amministratore':
        libro_id = input("Inserisci l'ID del libro: ")
        titolo = input("Inserisci il nuovo titolo del libro: ")
        autore = input("Inserisci il nuovo autore del libro: ")
        anno = input("Inserisci il nuovo anno del libro: ")
        update_libro(libro_id, titolo, autore, anno)
    else:
        print("\n---------------------------------------------------------------")
        print("Accesso negato: solo gli amministratori possono aggiornare libri.")
        print("---------------------------------------------------------------")
elif scelta == '5':
    clear_screen()
    if user_role == 'Amministratore':
        libro_id = input("Inserisci l'ID del libro da eliminare: ")
        delete_libro(libro_id)
    else:
        print("\n---------------------------------------------------------------")
        print("Accesso negato: solo gli amministratori possono eliminare libri.")
        print("---------------------------------------------------------------")
elif scelta == '6':
    clear_screen()
    libro_id = input("Inserisci l'ID del libro da prendere in prestito: ")
    new_prestito(libro_id)
elif scelta == '7':
    clear_screen()
    libro_id = input("Inserisci l'ID del libro da restituire: ")
    restituisci_libro(libro_id)
elif scelta == '8':
    clear_screen()
    get_user_prestiti(user_id)
elif scelta == '9':
    clear_screen()
    print("Uscita dal programma.")
    break
else:
    clear_screen()
    print("Scelta non valida. Riprova.")
```

3.2 Server

3.2.1 Panoramica dell'Implementazione

Il server è implementato utilizzando il modulo 'http.server' di Python per gestire le richieste HTTP. Utilizza 'socketserver.ThreadingMixIn' per supportare connessioni concorrenti. I componenti principali includono:

- **Configurazione e Connessione al Database**:

  Il database MySQL ('biblioteca') è configurato con dettagli di connessione come

host, username e password. Le connessioni al database sono gestite utilizzando ‘mysql.connector‘.

```
import mysql.connector

db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': 'password',
    'database': 'biblioteca'
}
def get_db_connection():
conn = mysql.connector.connect(**db_config)
return conn
```

- **Meccanismo di Threading e Locking**:

  Il server utilizza il threading di Python con un ‘Lock‘ (‘threading.Lock()‘) per garantire l'accesso al database in modo thread-safe (‘db_lock‘).

- **Gestione delle Richieste**:

  Il trattamento delle richieste HTTP è implementato nella classe ‘RequestHandler‘, che sottoclassa ‘BaseHTTPRequestHandler‘. Supporta metodi come ‘GET‘, ‘POST‘, ‘PUT‘, ‘DELETE‘ per gestire libri (‘libri‘), registrazione utenti, login, cambio password, prestiti (‘prestiti‘) e restituzioni libri (‘restituzioni‘).

```
class RequestHandler(BaseHTTPRequestHandler):
def _set_headers(self, status_code=200):
    self.send_response(status_code)
    self.send_header('Content-Type', 'application/json')
    self.end_headers()

def do_GET(self):
    start_time = time.time()
    parsed_path = urllib.parse.urlparse(self.path)
    path = parsed_path.path
    query = urllib.parse.parse_qs(parsed_path.query)
    status_code = 200

    if path == '/libri':
        self.get_libri()
    elif path.startswith('/libri/'):
        libro_id = int(path.split('/')[-1])
        self.get_libro(libro_id)
    elif path == '/prestiti':
        self.get_user_prestiti()
    else:
        status_code = 404
        self._set_headers(status_code)
```

```
            self.wfile.write(json.dumps({'error': 'Not found'}).encode('utf-8'))

        end_time = time.time()
        log_response_time(start_time, end_time, "GET", path, status_code)

    def do_POST(self):
        start_time = time.time()
        parsed_path = urllib.parse.urlparse(self.path)
        path = parsed_path.path
        status_code = 200

        if path == '/libri':
            self.add_libro()
        elif path == '/register':
            self.register_user()
        elif path == '/login':
            self.login_user()
        elif path == '/change_password':
            self.change_password()
        elif path == '/prestiti':
            self.new_prestito()
        elif path == '/restituzioni':
            self.restituisci_libro()
        else:
            status_code = 404
            self._set_headers(status_code)
            self.wfile.write(json.dumps({'error': 'Not found'}).encode('utf-8'))

        end_time = time.time()
        log_response_time(start_time, end_time, "POST", path, status_code)

    def do_PUT(self):
        start_time = time.time()
        parsed_path = urllib.parse.urlparse(self.path)
        path = parsed_path.path
        status_code = 200

        if path.startswith('/libri/'):
            libro_id = int(path.split('/')[-1])
            self.update_libro(libro_id)
        else:
            status_code = 404
            self._set_headers(status_code)
            self.wfile.write(json.dumps({'error': 'Not found'}).encode('utf-8'))

        end_time = time.time()
        log_response_time(start_time, end_time, "PUT", path, status_code)

    def do_DELETE(self):
```

```
start_time = time.time()
parsed_path = urllib.parse.urlparse(self.path)
path = parsed_path.path
status_code = 200

if path.startswith('/libri/'):
    libro_id = int(path.split('/')[-1])
    self.delete_libro(libro_id)
else:
    status_code = 404
    self._set_headers(status_code)
    self.wfile.write(json.dumps({'error': 'Not found'}).encode('utf-8'))

end_time = time.time()
log_response_time(start_time, end_time, "DELETE", path, status_code)
```

- **Registrazione dei Tempi di Risposta**:

  Il server registra i tempi di risposta in un file CSV (‘response_times.csv‘) utilizzando la funzione ‘log_response_time()‘.

```
def log_response_time(start_time, end_time, method, path, status_code):
    duration = (end_time - start_time)*1000
    log_message = {
        'method': method,
        'path': path,
        'status': status_code,
        'duration (ms)': f"{duration:.4f}"
    }

    with open('response_times.csv', mode='a', newline='') as file:
        writer = csv.DictWriter(file, fieldnames=log_message.keys())
        if file.tell() == 0:
            writer.writeheader()
        writer.writerow(log_message)
```

- **Libri (‘/libri‘)**:
  - **GET**: Recupera tutti i libri o un libro specifico tramite ID.
  - **POST**: Aggiunge un nuovo libro al database.
  - **PUT**: Aggiorna i dettagli di un libro esistente.
  - **DELETE**: Elimina un libro dal database.
- **Utenti (‘/register‘, ‘/login‘, ‘/change_password‘)**:
  - **POST ‘/register‘**: Registra un nuovo utente nel sistema.
  - **POST ‘/login‘**: Gestisce l’autenticazione dell’utente.

  - **POST ‘/change_password‘**: Permette all’utente di cambiare la propria password.

- **Prestiti (‘/prestiti‘, ‘/restituzioni‘)**:
  - **POST ‘/prestiti‘**: Avvia un nuovo prestito di un libro a un utente.
  - **POST ‘/restituzioni‘**: Gestisce la restituzione di un libro precedentemente prestato.
  - **GET ‘/prestiti‘**: Recupera i prestiti attivi di un utente.

3.2.2 Endpoint e Operazioni

- **Operazioni GET**: Recupero di tutti i libri (‘/libri‘), di un libro specifico (‘/libri/id‘), prestiti dell’utente (‘/prestiti‘).
- **Operazioni POST**: Aggiunta di un libro (‘/libri‘), registrazione di un nuovo utente (‘/register‘), login dell’utente (‘/login‘), cambio password (‘/change_password‘), avvio di un prestito (‘/prestiti‘), restituzione di un libro (‘/restituzioni‘).
- **Operazioni PUT**: Aggiornamento dei dettagli di un libro (‘/libri/id‘).
- **Operazioni DELETE**: Eliminazione di un libro (‘/libri/id‘).

3.2.3 Considerazioni di Sicurezza

Le password degli utenti sono hashate sia in scrittura nel Database utilizzando ‘bcrypt‘ prima di essere memorizzate nel database, sia in lettura durante il login, garantendo una sicurezza maggiore dei dati.

3.2.4 Conclusioni

Il server implementato offre funzionalità robuste per gestire un sistema bibliotecario, gestendo efficientemente le richieste concorrenti con operazioni sicure sul database.

3.3 Database

MySQL è un sistema di gestione di database relazionali open-source molto popolare. È ampiamente utilizzato per la gestione di grandi quantità di dati e applicazioni web. MySQL supporta linguaggi come SQL (Structured Query Language) per interrogare e gestire i dati all’interno dei database.

MySQL Connector for Python è un’interfaccia Python che consente di connettersi a un database MySQL utilizzando Python. È una libreria che facilita l’interazione tra un’applicazione Python e un server MySQL, permettendo di eseguire query SQL, creare e gestire tabelle, inserire dati, e altro ancora.

3.3.1 Principali funzionalità di MySQL Connector for Python

- **Connessione al database**: MySQL Connector for Python permette di stabilire facilmente una connessione a un server MySQL specificato utilizzando le credenziali appropriate (utente, password, host).

- **Esecuzione di query SQL**: La libreria supporta l'esecuzione di query SQL direttamente da Python, consentendo di recuperare, aggiornare o eliminare dati all'interno delle tabelle MySQL.

- **Gestione dei dati**: Permette di manipolare dati all'interno delle tabelle del database, inclusi l'inserimento di nuove righe, l'aggiornamento delle righe esistenti e la gestione delle relazioni tra le tabelle tramite chiavi esterne.

- **Creazione e gestione delle tabelle**: È possibile creare nuove tabelle all'interno del database direttamente da Python, specificando campi, tipi di dati e vincoli di chiave primaria o esterna.

- **Transazioni e sicurezza**: Supporta l'utilizzo di transazioni per garantire l'integrità dei dati, consentendo di eseguire operazioni atomiche sul database.

3.3.2 Creazione del Database

```
#Configurazione Db
db_config = {
    'user': 'root',
    'password': '',
    'host': '127.0.0.1'
}

#Connessione al Db
conn = mysql.connector.connect(**db_config)
cursor = conn.cursor()

#Creazione del Db
cursor.execute("CREATE DATABASE IF NOT EXISTS biblioteca")
cursor.execute("USE biblioteca");
```

3.3.3 Tabella Libri

```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS libri (
        id_libro INT AUTO_INCREMENT PRIMARY KEY,
        titolo VARCHAR(255) NOT NULL,
        autore VARCHAR(255) NOT NULL,
        anno INT NOT NULL,
        disponibilita CHAR(3) DEFAULT 'YES'
    )
""")
```

3.3.4 Tabella Ruoli

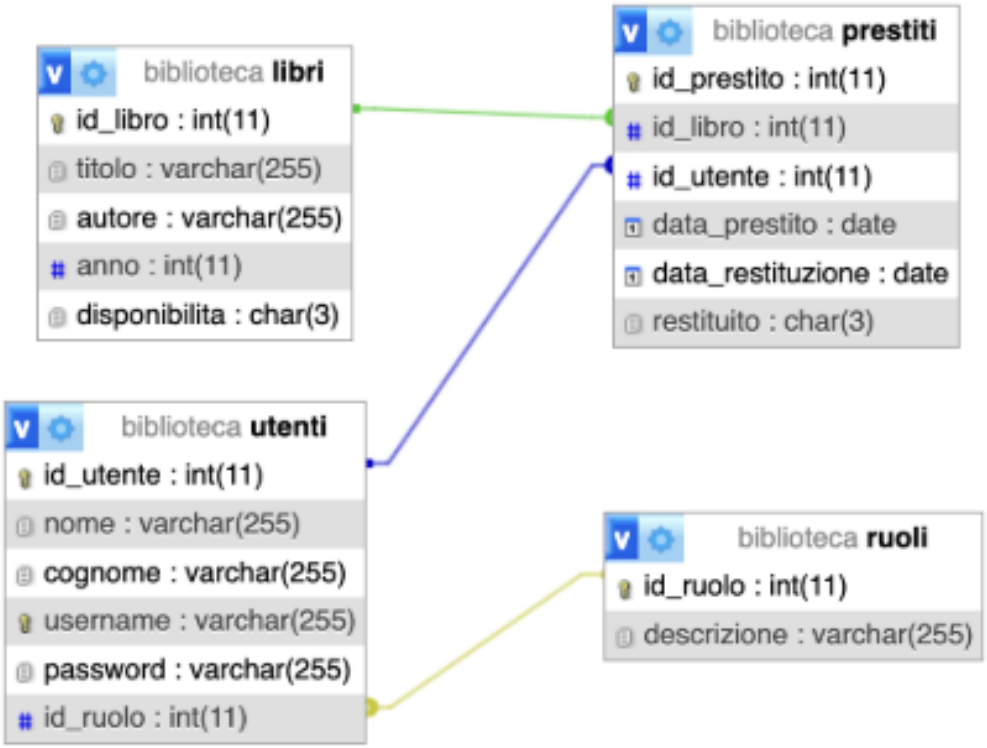
```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS ruoli (
        id_ruolo INT AUTO_INCREMENT PRIMARY KEY,
        descrizione VARCHAR(255) NOT NULL
    )
""")
```

3.3.5 Tabella Utenti

```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS utenti (
        id_utente INT AUTO_INCREMENT PRIMARY KEY,
        nome VARCHAR(255) NOT NULL,
        cognome VARCHAR(255) NOT NULL,
        username VARCHAR(255) NOT NULL UNIQUE,
        password VARCHAR(255) NOT NULL,
        id_ruolo INT NOT NULL,
        FOREIGN KEY (id_ruolo) REFERENCES ruoli(id_ruolo)
    )
""")
```

3.3.6 Tabella Prestiti

```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS prestiti (
        id_prestito INT AUTO_INCREMENT PRIMARY KEY,
        id_libro INT NOT NULL,
        id_utente INT NOT NULL,
        data_prestito DATE NOT NULL,
        data_restituzione DATE DEFAULT NULL,
        restituito CHAR(3) DEFAULT 'NO',
        FOREIGN KEY (id_libro) REFERENCES libri(id_libro),
        FOREIGN KEY (id_utente) REFERENCES utenti(id_utente)
    )
""")
```



3.3.7 Inserimento Ruoli

```
ruoli = [
    ('Amministratore'),
    ('Utente')
]

cursor.executemany("INSERT INTO ruoli (descrizione) VALUES (%s)",
[(ruolo,) for ruolo in ruoli])
```

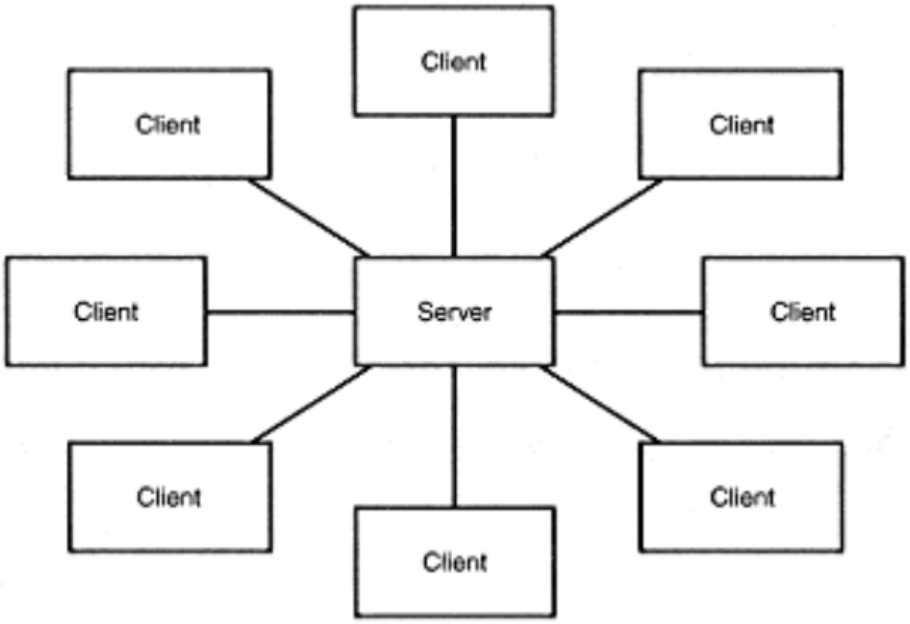
3.3.8 Inserimento Libri

```
libri = [
    ('Il nome della rosa', 'Umberto Eco', 1980),
    ('1984', 'George Orwell', 1949),
    ('Il piccolo principe', 'Antoine de Saint-Exupéry', 1943),
    ('La Divina Commedia', 'Dante Alighieri', 1320),
    ('Orgoglio e pregiudizio', 'Jane Austen', 1813),
    ('Moby Dick', 'Herman Melville', 1851),
    ('Guerra e pace', 'Lev Tolstoj', 1869),

    ..etc

cursor.executemany("INSERT INTO libri (titolo, autore, anno)
VALUES (%s, %s, %s)", libri)
```

Risultati sperimentali

4.1 Introduzione

Sono stati effettuati dei test per andare a verificare se il server è in grado di gestire grossi carichi, nel caso in cui una grande quantità di client invii richieste contemporaneamente.



4.2 Run process linux (Avvio di Client Multipli Utilizzando Python)

Il codice seguente illustra come avviare più istanze di un client Python utilizzando il modulo subprocess e gestire il loro lancio asincrono utilizzando il terminale GNOME.

Il codice è stato modificato di volta in volta andando a cambiare il numero dei client aperti.

4.2.1 Descrizione del Codice

Il codice è progettato per avviare più istanze di un client Python (client.py) simultaneamente, utilizzando il terminale GNOME come ambiente di esecuzione separato per ciascun client. Ciò è particolarmente utile quando si desidera testare o simulare comportamenti multi-client senza dover avviare manualmente ogni istanza.

4.2.2 Implementazione

Il codice Python è strutturato come segue:

```
import subprocess
import time

def launch_client(clientid):
print(f"Lancio del client {clientid}")
subprocess.Popen(["gnome-terminal", "--", "python3", "client.py"])

def main():
num_clients = 5 # Numero di client da avviare
for i in range(num_clients):
    launch_client(i + 1)
    time.sleep(1)
if name == "main":
main()
```

4.2.3 Spiegazione

- **Funzione launch_client(clientid)**:
  - Questa funzione riceve come parametro clientid, che rappresenta l'identificatore univoco del client da avviare.
  - Utilizza subprocess.Popen per avviare un nuovo processo. Nel nostro caso, viene utilizzato il comando gnome-terminal per aprire un nuovo terminale GNOME.
  - Il terminale GNOME viene quindi utilizzato per eseguire il comando python3 client.py, che avvia il client Python specificato (client.py).
  - Il messaggio Lancio del client clientid viene stampato per indicare l'inizio del processo di avvio del client.
- **Funzione 'main()'**:
  - La funzione principale 'main()' determina il numero di client da avviare.
  - Utilizza un ciclo 'for' per iterare su ciascun client da avviare.
  - Chiama la funzione 'launch_client(i + 1)' per avviare ciascun client, aggiungendo un ritardo di 1 secondo tra ciascuna iterazione ('time.sleep(1)') per evitare conflitti durante l'avvio simultaneo.
- **Condizione di Avvio (if __name__ == "__main__":)**:
  - Questa condizione verifica se lo script è eseguito direttamente come programma principale.
  - In tal caso, viene chiamata la funzione 'main()' per avviare il processo di avvio dei client.

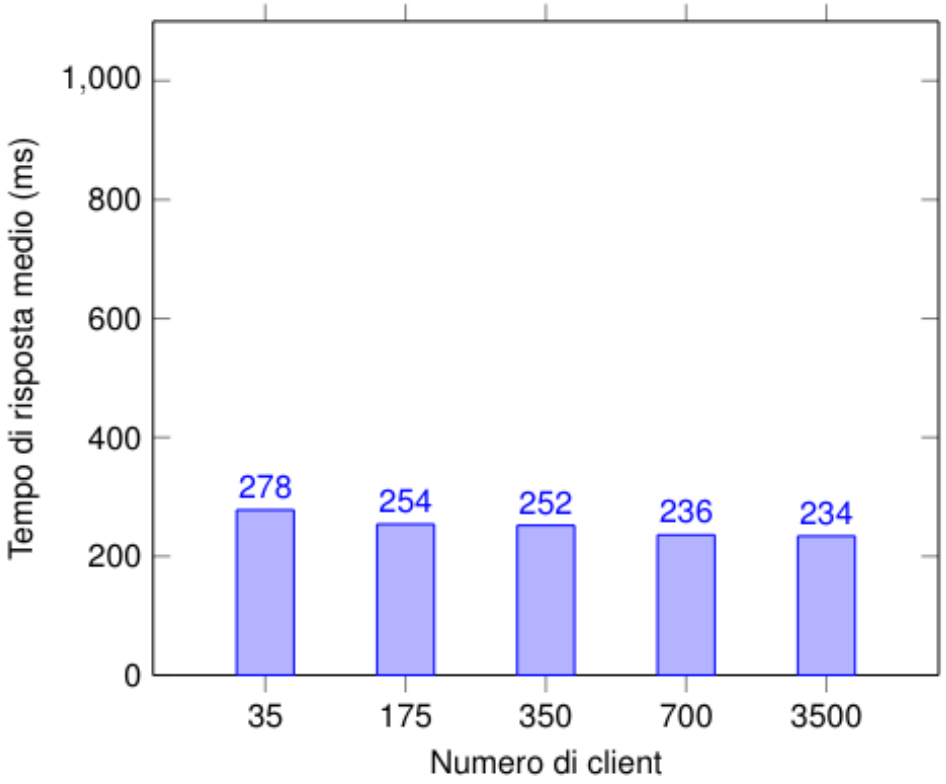
```
192.168.128.235 - - [26/Jun/2024 13:44:47] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:48] "GET /libri HTTP/1.1" 200 -
192.168.128.233 - - [26/Jun/2024 13:44:48] "POST /login HTTP/1.1" 200 -
192.168.128.233 - - [26/Jun/2024 13:44:48] "GET /libri HTTP/1.1" 200 -
192.168.128.235 - - [26/Jun/2024 13:44:49] "GET /libri HTTP/1.1" 200 -
192.168.128.237 - - [26/Jun/2024 13:44:49] "GET /libri HTTP/1.1" 200 -
192.168.128.231 - - [26/Jun/2024 13:44:49] "POST /login HTTP/1.1" 200 -
192.168.128.235 - - [26/Jun/2024 13:44:49] "GET /libri HTTP/1.1" 200 -
192.168.128.235 - - [26/Jun/2024 13:44:50] "POST /login HTTP/1.1" 200 -
192.168.128.233 - - [26/Jun/2024 13:44:50] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:50] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:50] "GET /libri HTTP/1.1" 200 -
192.168.128.237 - - [26/Jun/2024 13:44:51] "POST /login HTTP/1.1" 200 -
192.168.128.236 - - [26/Jun/2024 13:44:51] "POST /login HTTP/1.1" 200 -
192.168.128.233 - - [26/Jun/2024 13:44:52] "POST /login HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:52] "POST /login HTTP/1.1" 200 -
192.168.128.231 - - [26/Jun/2024 13:44:52] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:53] "POST /login HTTP/1.1" 200 -
192.168.128.233 - - [26/Jun/2024 13:44:53] "GET /libri HTTP/1.1" 200 -
192.168.128.238 - - [26/Jun/2024 13:44:54] "GET /libri HTTP/1.1" 200 -
192.168.128.238 - - [26/Jun/2024 13:44:54] "POST /login HTTP/1.1" 200 -
192.168.128.231 - - [26/Jun/2024 13:44:54] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:54] "GET /libri HTTP/1.1" 200 -
192.168.128.235 - - [26/Jun/2024 13:44:55] "POST /login HTTP/1.1" 200 -
192.168.128.235 - - [26/Jun/2024 13:44:55] "GET /libri HTTP/1.1" 200 -
192.168.128.238 - - [26/Jun/2024 13:44:56] "POST /login HTTP/1.1" 200 -
192.168.128.237 - - [26/Jun/2024 13:44:56] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:56] "GET /libri HTTP/1.1" 200 -
192.168.128.235 - - [26/Jun/2024 13:44:56] "GET /libri HTTP/1.1" 200 -
192.168.128.236 - - [26/Jun/2024 13:44:56] "GET /libri HTTP/1.1" 200 -
192.168.128.233 - - [26/Jun/2024 13:44:57] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:57] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:57] "POST /login HTTP/1.1" 200 -
192.168.128.231 - - [26/Jun/2024 13:44:57] "POST /login HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:44:58] "GET /libri HTTP/1.1" 200 -
192.168.128.236 - - [26/Jun/2024 13:44:59] "POST /login HTTP/1.1" 200 -
192.168.128.238 - - [26/Jun/2024 13:44:59] "GET /libri HTTP/1.1" 200 -
192.168.128.235 - - [26/Jun/2024 13:45:00] "GET /libri HTTP/1.1" 200 -
192.168.128.238 - - [26/Jun/2024 13:45:01] "GET /libri HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:45:02] "GET /libri HTTP/1.1" 200 -
192.168.128.231 - - [26/Jun/2024 13:45:02] "GET /libri HTTP/1.1" 200 -
192.168.128.238 - - [26/Jun/2024 13:45:04] "POST /login HTTP/1.1" 200 -
192.168.128.233 - - [26/Jun/2024 13:45:04] "POST /login HTTP/1.1" 200 -
192.168.128.230 - - [26/Jun/2024 13:45:04] "GET /libri HTTP/1.1" 200 -
192.168.128.236 - - [26/Jun/2024 13:45:04] "GET /libri HTTP/1.1" 200 -
192.168.128.231 - - [26/Jun/2024 13:45:08] "GET /libri HTTP/1.1" 200 -
192.168.128.231 - - [26/Jun/2024 13:45:08] "POST /login HTTP/1.1" 200 -
192.168.128.238 - - [26/Jun/2024 13:45:09] "GET /libri HTTP/1.1" 200 -
192.168.128.233 - - [26/Jun/2024 13:45:09] "GET /libri HTTP/1.1" 200 -
192.168.128.231 - - [26/Jun/2024 13:45:13] "GET /libri HTTP/1.1" 200 -
```

4.2.4 Considerazioni Finali

Questo approccio fornisce un metodo efficiente per testare e gestire più istanze di client Python contemporaneamente, sfruttando le capacità di gestione dei processi di Python e l'ambiente del terminale GNOME per l'esecuzione separata di ciascun client. È utile in scenari di sviluppo e test in cui è necessario simulare interazioni tra più client in modo simultaneo e indipendente.

4.3 Risultati

I risultati dei test mostrano come il server sia stato in grado di gestire numerose richieste da parte dei client senza subire rallentamenti di alcun tipo.



Conclusioni

Il progetto ha dimostrato come l'utilizzo della programmazione multithread possa migliorare significativamente l'efficienza e la reattività di un sistema complesso. L'implementazione ha permesso di gestire simultaneamente più richieste degli utenti, riducendo i tempi di attesa e migliorando l'esperienza complessiva dell'utente.

- Il sistema è stato progettato per essere scalabile, consentendo l'aggiunta di nuove funzionalità e la gestione di un numero crescente di utenti e libri senza comprometterne le prestazioni.

5.1 Sviluppi futuri

Ci sono diversi ambiti in cui il progetto potrebbe essere ulteriormente sviluppato e migliorato:

- Aggiungere funzionalità avanzate come la vendita di libri, o la possibilità di recensire i libri letti
- Implementare algoritmi di raccomandazione basati su machine learning per suggerire libri agli utenti in base alle loro letture passate
- Sviluppare un'interfaccia utente più intuitiva e user-friendly
- Implementare l'aggiunta di altri linguaggi per rendere il sistema accessibile ad un numero maggiore di utenti