Università degli studi di Napoli Pathenope

Progetto di Reti di Calcolatori 2023/24

Traccia – Università

Codice gruppo: 7ao3774ibdj

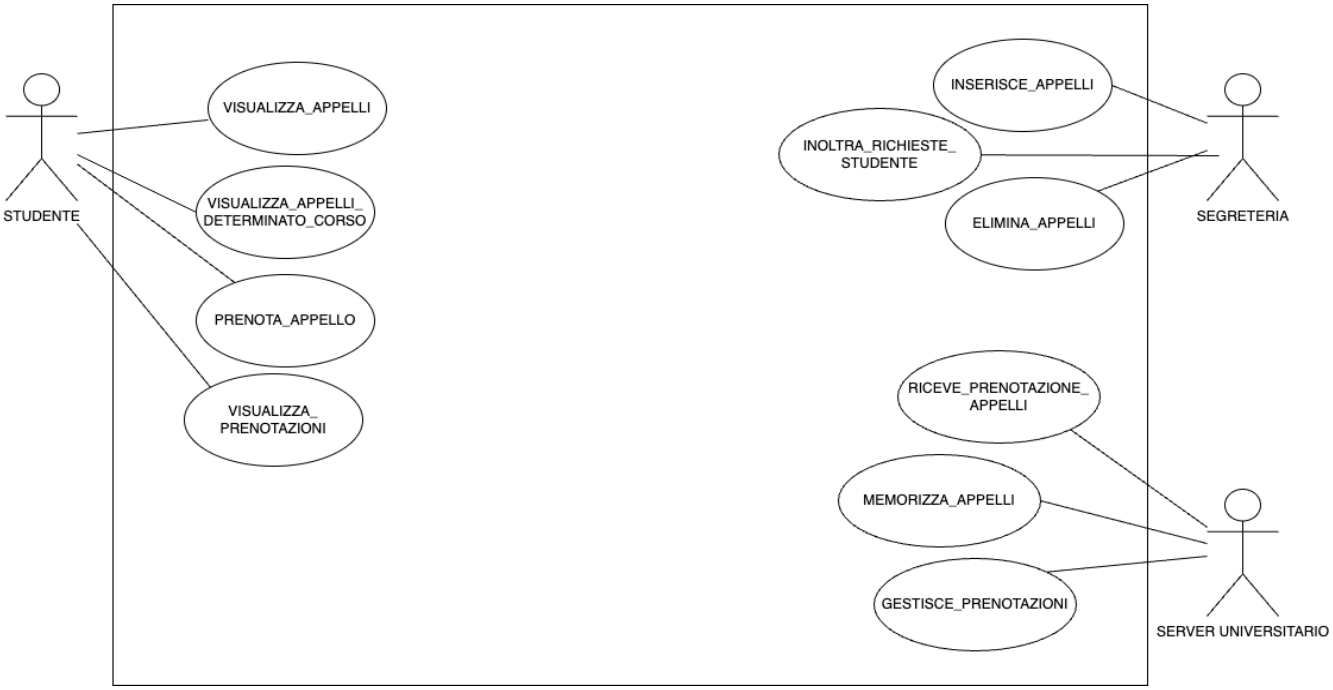
Alessandro Di Stefano 0124002276

Sommario

Descrizione Progetto

L'obiettivo è sviluppare un'applicazione **client/server parallela** per la gestione degli esami universitari. Il sistema prevede **tre attori** principali: la segreteria, lo studente e il server universitario.
La segreteria si occupa di registrare gli appelli sul server, inoltrare al server universitario le richieste di prenotazione degli studenti e comunicare agli studenti le date disponibili per gli appelli richiesti.
Lo studente può verificare con la segreteria la disponibilità degli appelli per un determinato corso e inviare una richiesta di prenotazione, ricevendo successivamente il numero progressivo assegnato alla sua prenotazione.
Il server universitario, infine, gestisce l'aggiunta di nuovi appelli e l'elaborazione delle prenotazioni ricevute.

Diagramma dei casi d'uso



Il diagramma rappresenta i casi d'uso principali del sistema di gestione degli esami universitari, modellando le interazioni tra i tre attori principali: **Studente**, **Segreteria** e **Server Universitario**.
Ogni attore svolge specifici ruoli all'interno del sistema, evidenziati attraverso le relazioni con i rispettivi casi d'uso.

Attori e Casi d'Uso

Studente

Lo studente interagisce con il sistema tramite la segreteria per:

- **Visualizzare gli appelli disponibili**: Richiede un elenco generale degli appelli presenti.
- **Visualizzare appelli per un determinato corso**: Specifica un corso e ottiene le date degli appelli disponibili.
- **Prenotare un appello**: Invia una richiesta di prenotazione per un appello specifico.
- **Visualizzare le prenotazioni effettuate**: Controlla lo stato delle prenotazioni già registrate.

Segreteria

La segreteria funge da intermediario tra lo studente e il server universitario, gestendo:

- **Inserimento degli appelli**: Registra nuovi appelli nel sistema universitario.
- **Inoltro delle richieste degli studenti**: Trasmette al server le prenotazioni ricevute dagli studenti.
- **Eliminazione degli appelli**: Rimuove appelli dal sistema.

Server Universitario

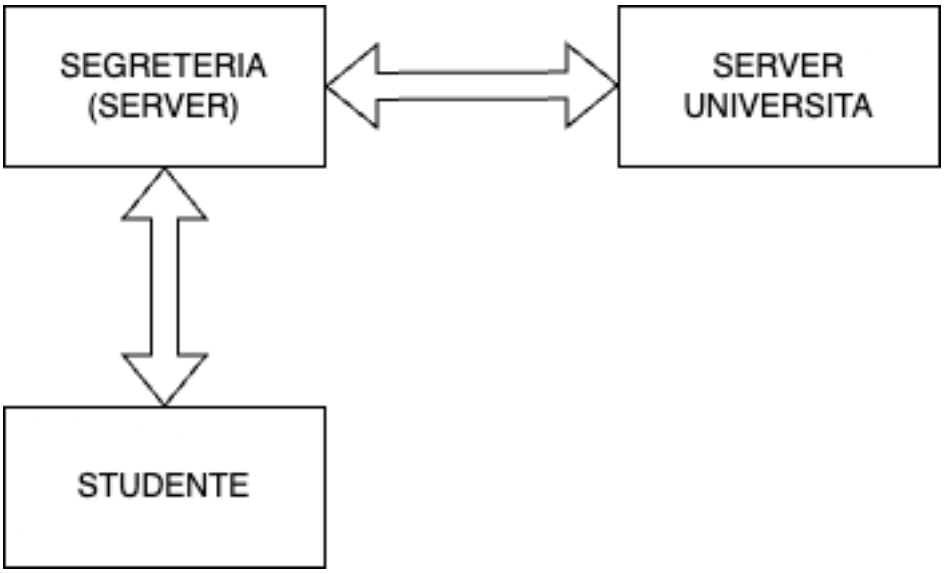
Il server rappresenta il nucleo centrale del sistema, con i seguenti compiti:

- **Ricevere le prenotazioni degli appelli**: Elabora le richieste di prenotazione inoltrate dalla segreteria.
- **Memorizzare gli appelli**: Gestisce l'archiviazione degli appelli nel sistema.
- **Gestire le prenotazioni**: Organizza le prenotazioni, assegnando un numero progressivo e garantendo la coerenza dei dati.
- **Memorizzare le prenotazioni**: Gestisce l'archiviazione delle prenotazioni agli appelli nel sistema.

Flusso delle Interazioni

Lo studente comunica esclusivamente con la segreteria per richiedere informazioni o inviare prenotazioni.
La segreteria opera come intermediario, inviando e ricevendo dati dal server universitario.
Il server universitario si occupa di elaborare i dati e mantenere la consistenza delle informazioni sugli appelli e sulle prenotazioni.

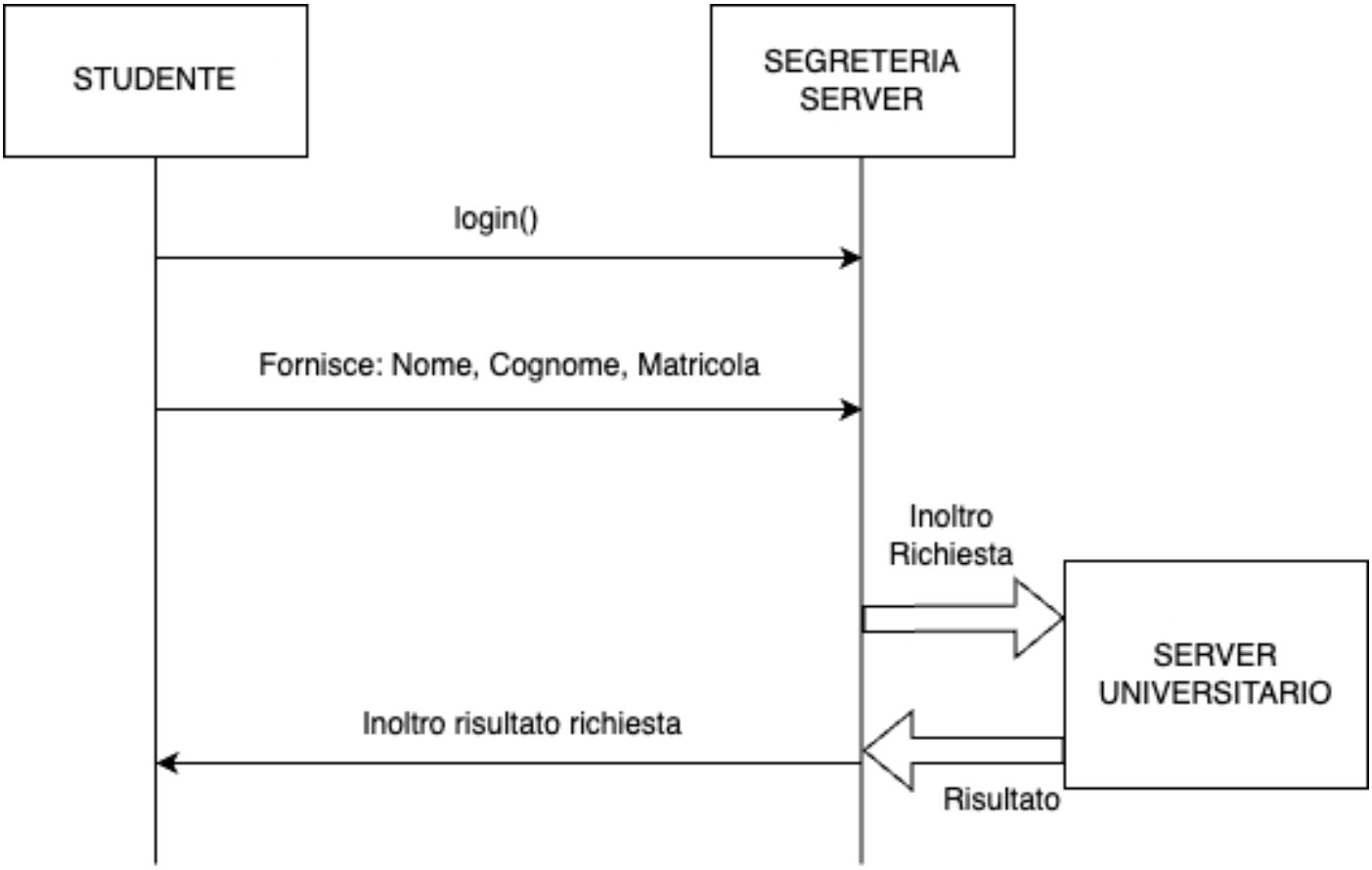
Schema Architetturale



L'applicazione segue **un'architettura client-server parallela** per gestire operazioni relative agli appelli d'esame, come l'inserimento, l'eliminazione e la prenotazione. La comunicazione tra client e server avviene attraverso il **protocollo TCP/IP**, sfruttando i socket per trasmettere dati in modalità bidirezionale.
Lo schema architetturale del sistema si basa su tre entità principali: lo Studente, la Segreteria e il Server Universitario. **Lo studente** non interagisce direttamente con il server universitario, ma si interfaccia esclusivamente tramite **la segreteria (server)**, che funge da intermediario per inoltrare le richieste dello studente al **server universitario** e restituire i dati elaborati. **La segreteria (client)** è inoltre responsabile dell'inserimento di nuovi appelli ed esami nel server universitario.
Per l'implementazione è stato utilizzato **il linguaggio Java**, sfruttando le sue API standard per la gestione delle connessioni di rete che garantisce un ambiente robusto per lo sviluppo del sistema. La persistenza dei dati è stata realizzata attraverso **la serializzazione su file**, assicurando un'archiviazione semplice e facilmente gestibile.

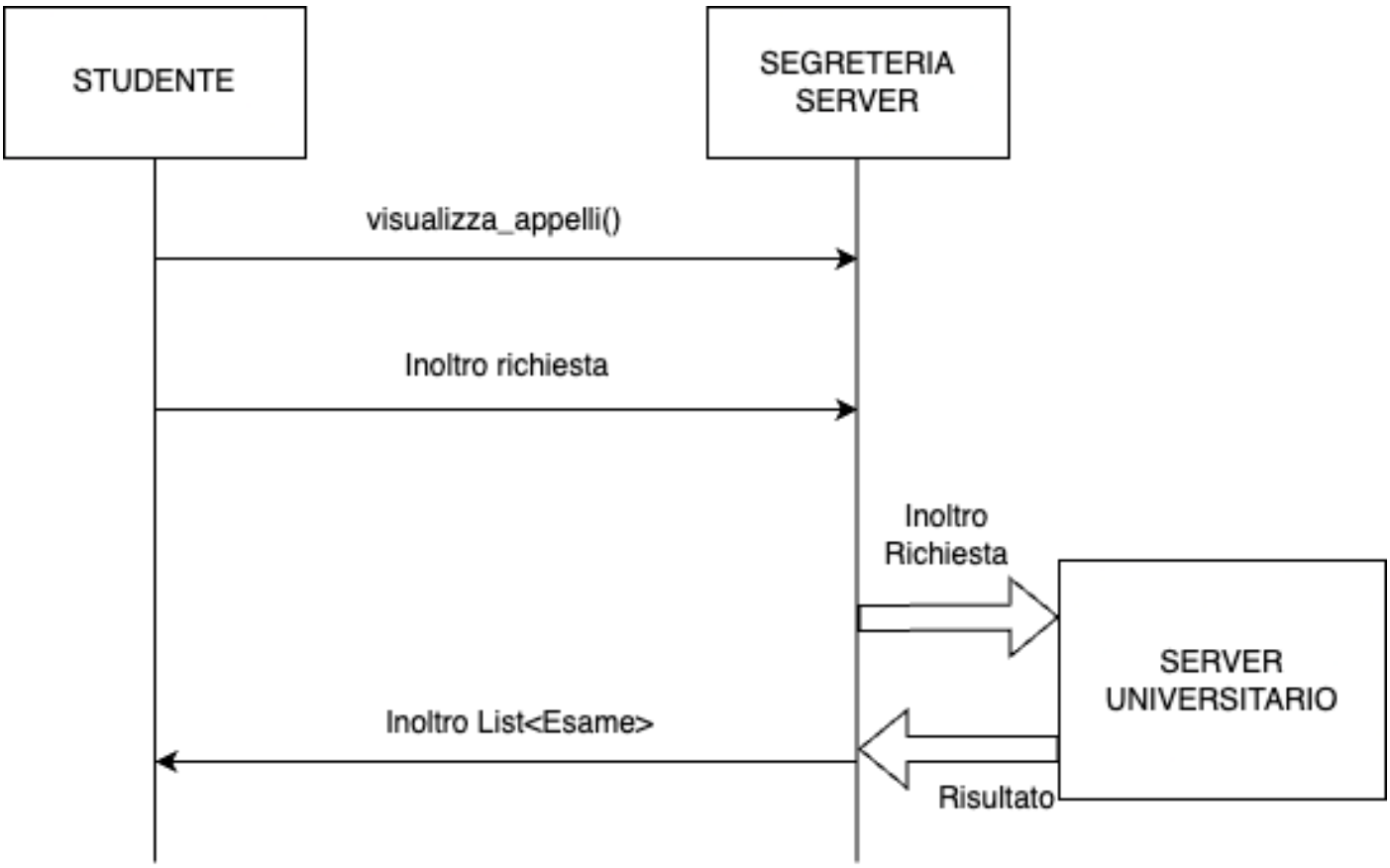
Operazioni Studente

Autenticazione Studente



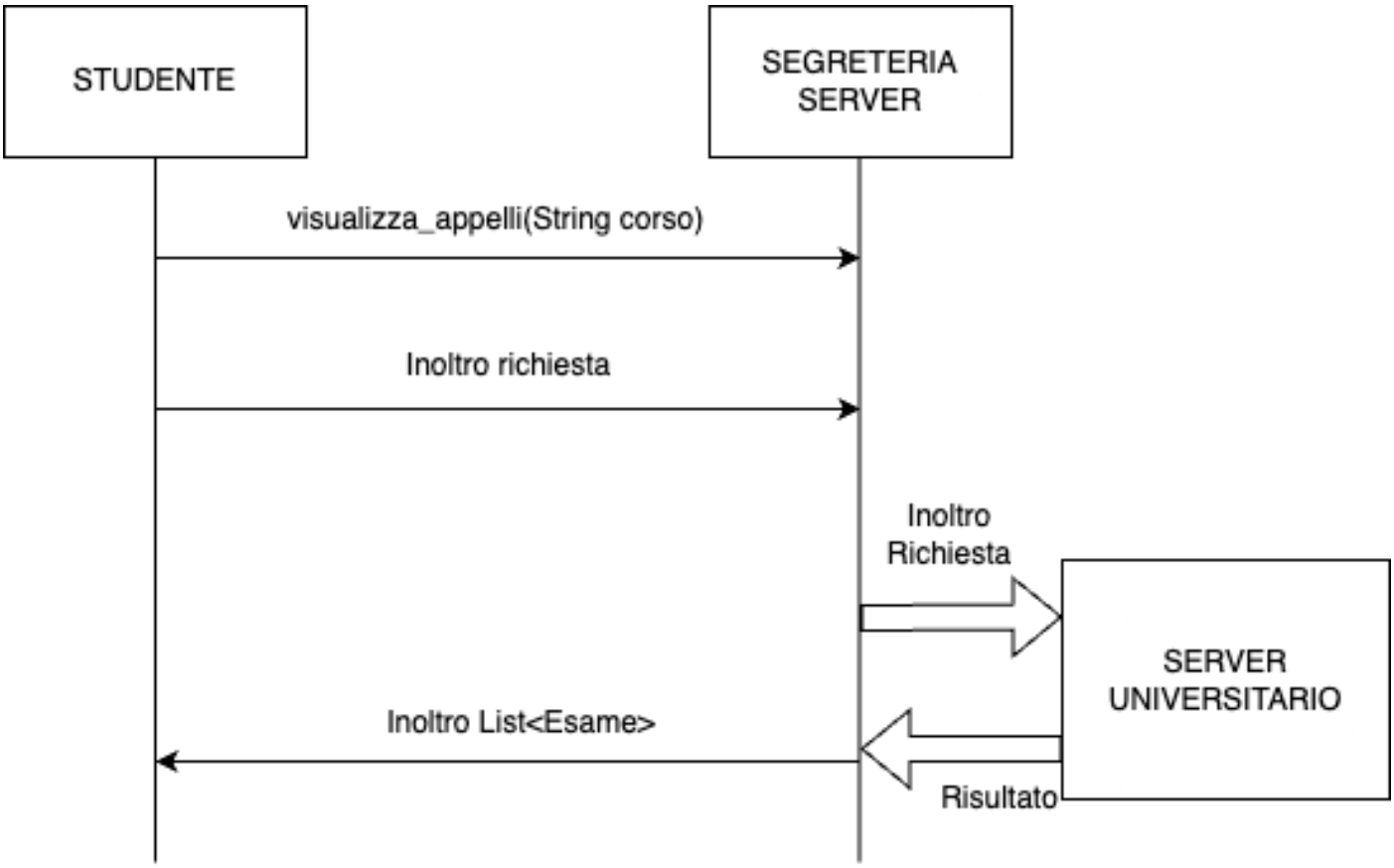
La sequenza inizia con l'invio di una richiesta di accesso (login) da parte dello studente al server della segreteria. Lo studente fornisce le proprie credenziali di identificazione, che includono nome, cognome e numero di matricola.
Il server della segreteria inoltra quindi queste informazioni al server universitario per la verifica e l'elaborazione della richiesta. Il server universitario processa i dati ricevuti e restituisce l'esito dell'operazione al server della segreteria. Infine, quest'ultimo comunica il risultato dell'autenticazione allo studente, completando il processo.

Visualizza Appelli



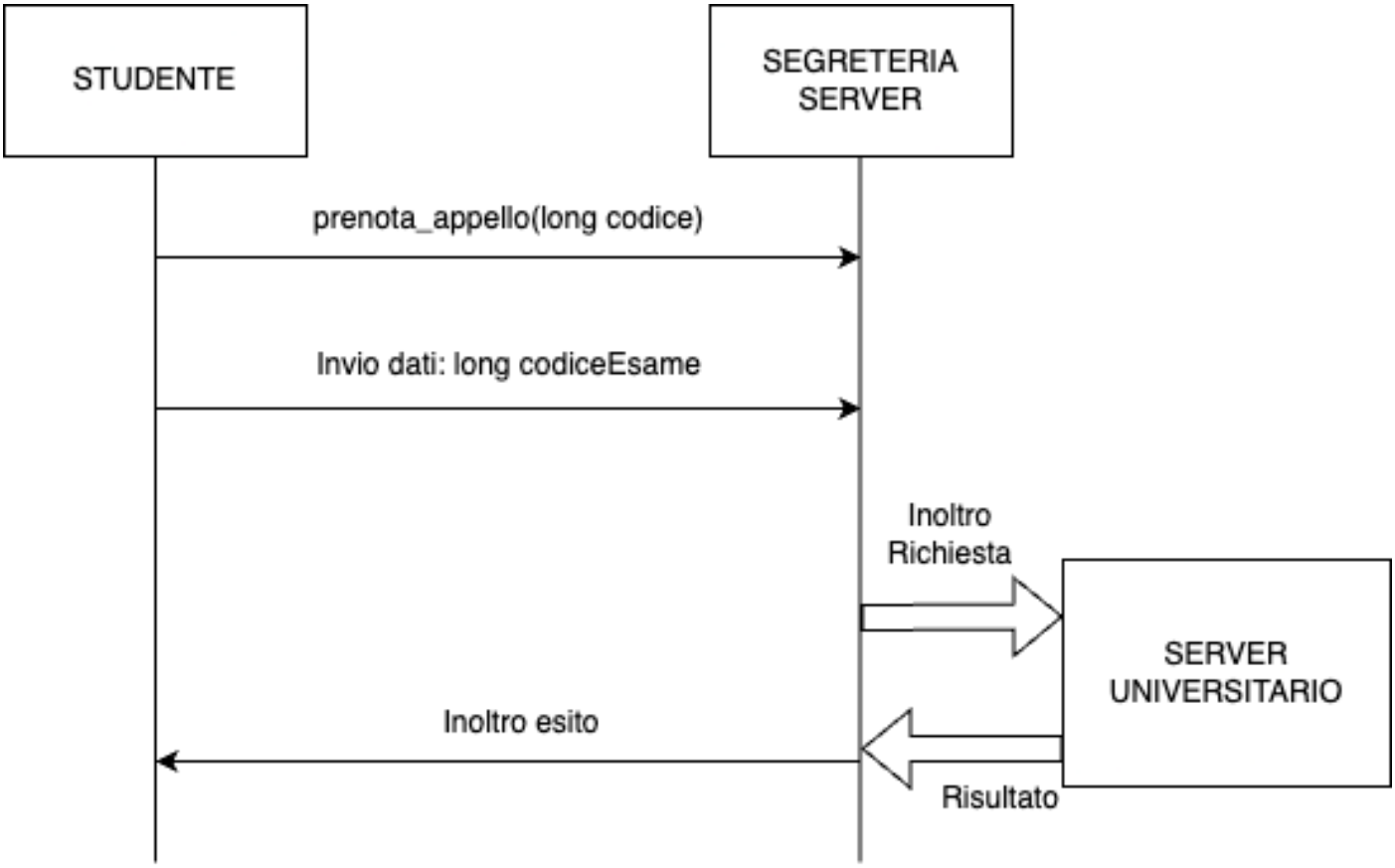
La sequenza inizia con la richiesta "visualizza_appelli()" inviata dallo studente al server della segreteria. Quest'ultimo, ricevuta la richiesta, provvede a inoltrarla al server universitario per l'elaborazione.
Il server universitario processa la richiesta verificando i dati relativi agli appelli disponibili e invia il risultato al server della segreteria sotto forma di una lista contenente le informazioni sugli appelli. Il server della segreteria, a sua volta, inoltra la lista degli appelli allo studente, completando così il processo.

Visualizza Appelli di uno Specifico Corso



La sequenza si avvia con la chiamata della funzione "visualizza_appelli_corso(String corso)" da parte dello studente verso il server della segreteria. Lo studente fornisce come parametro una stringa contenente il nome del corso di interesse.
Il server della segreteria riceve il dato e lo inoltra al server universitario. Il server universitario elabora la richiesta e restituisce al server della segreteria una lista degli appelli disponibili per il corso specificato. Infine, il server della segreteria trasmette la lista degli appelli allo studente.

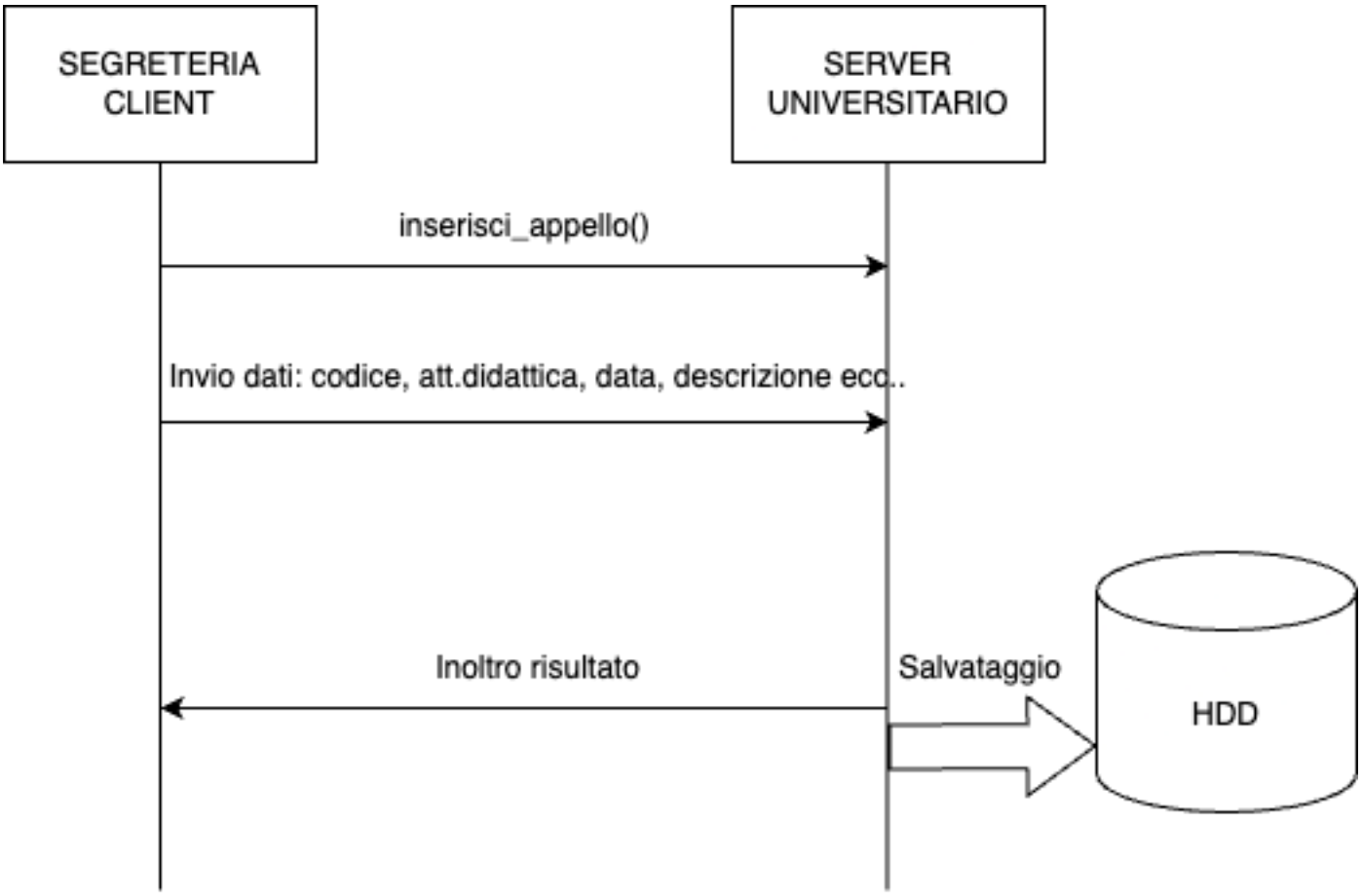
Prenotazione Appello



La sequenza inizia con la chiamata alla funzione "prenota_appello(long codice)" da parte dello studente verso il server della segreteria. Lo studente trasmette un parametro identificativo, il codice dell'esame, che permette di individuare con precisione l'appello a cui intende iscriversi.
Il server della segreteria riceve la richiesta e provvede a inoltrarla al server universitario, che è responsabile dell'elaborazione della prenotazione. Quest'ultimo verifica la richiesta e restituisce un esito, indicante il successo o il fallimento della prenotazione, al server della segreteria. Infine, il server della segreteria invia lo stesso esito allo studente, concludendo il processo.

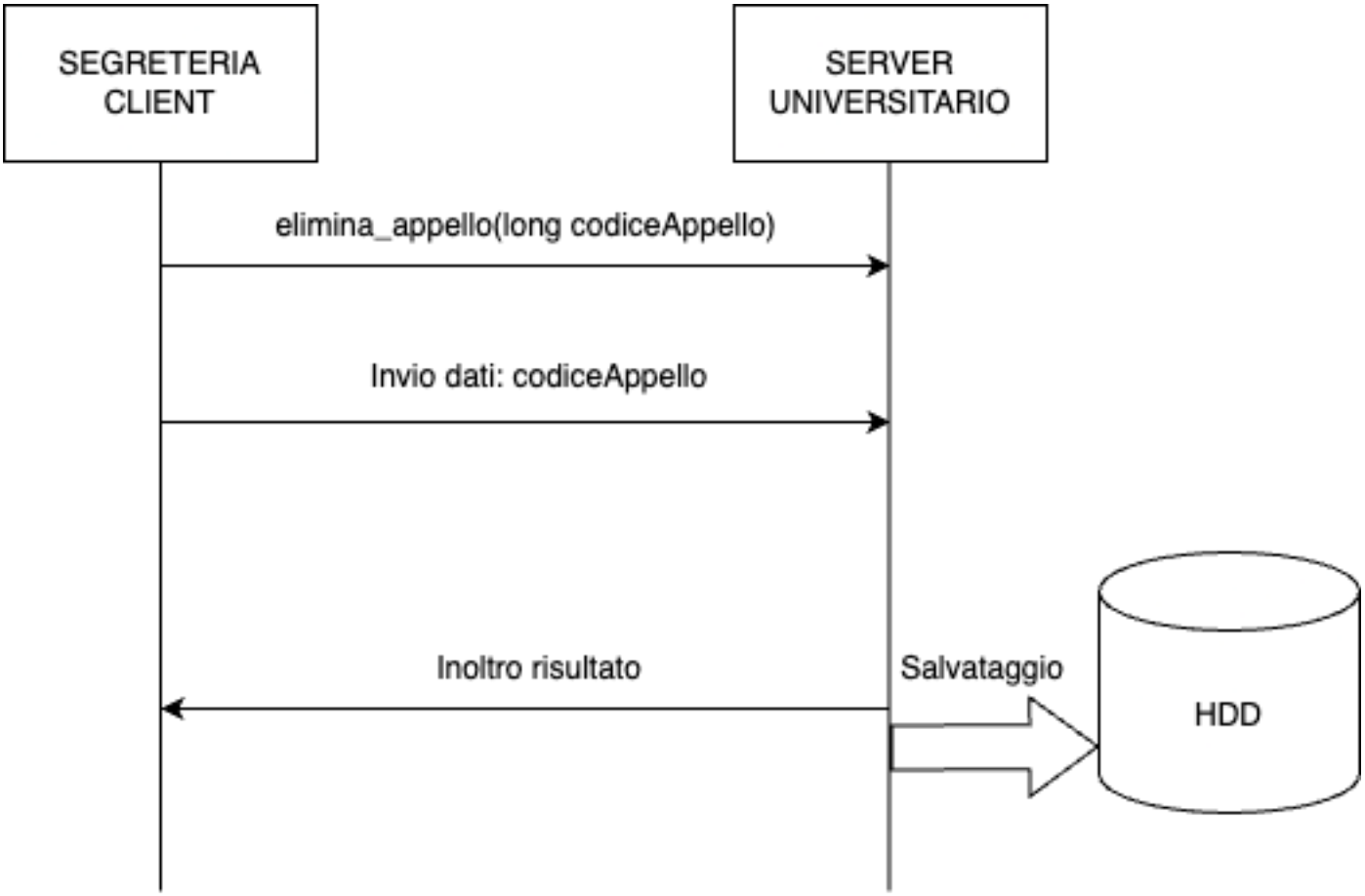
Operazioni Segreteria

Aggiungi Appello



La sequenza viene avviata con la chiamata della funzione "inserisci_appello()" dal client della segreteria verso il server universitario. In questa fase, vengono trasmessi i dati necessari per la creazione dell'appello, come il codice dell'appello, l'attività didattica, la data, una descrizione, il nome del Docente e il numero massimo di prenotazioni.
Il server universitario riceve le informazioni e procede con l'elaborazione e il salvataggio dei dati nel sistema di archiviazione (HDD). Dopo aver completato questa operazione, il server restituisce un esito al client della segreteria, che può confermare il successo o segnalare eventuali problemi.

Elimina Appello



La procedura inizia con la chiamata alla funzione "elimina_appello(long codiceAppello)" inviata dal client della segreteria al server universitario. Il client trasmette i dati relativi all'appello da eliminare, identificati tramite il codiceAppello.
Il server universitario, una volta ricevuti i dati, avvia il processo di eliminazione verificando e rimuovendo le informazioni relative all'appello specificato. Successivamente, registra le modifiche nel sistema di archiviazione rappresentato dall'HDD, assicurando che i dati vengano aggiornati in maniera persistente. Infine, il server universitario restituisce un risultato al client della segreteria, indicante l'esito dell'operazione (conferma di avvenuta eliminazione o notifica di eventuali errori).

Dettagli Implementativi Client/Server

Seguono i dettagli implementativi dei client e server.

Segreteria (Client) / Server Universitario

Dal lato server, la classe principale è `UniversityServer`, che avvia un `ServerSocket` sulla porta 12345, dove rimane in ascolto per connessioni in ingresso. All'avvio, il server carica i dati persistiti da un file di tipo serializzato chiamato university_data.ser. Questi dati includono informazioni sugli esami e sulle prenotazioni associate. Per ogni connessione accettata, il server crea un thread separato gestito dalla classe `ClientHandler`. Questo approccio consente di mantenere l'elaborazione delle richieste parallela e indipendente per ogni client connesso. La comunicazione tra client e server utilizza oggetti Java serializzati, inviati e ricevuti tramite `ObjectOutputStream` e `ObjectInputStream`.
Dal lato client, l'interazione con l'utente è gestita attraverso la console, con input acquisiti tramite la classe `Scanner`. Il programma utilizza la classe `SegreteriaClient`, che stabilisce connessioni con il server sulla stessa porta. Per ogni operazione richiesta dall'utente, come l'aggiunta o l'eliminazione di un esame, il client costruisce un oggetto di tipo `Packet`. Questo oggetto incapsula i dettagli della richiesta e viene inviato al server tramite una connessione socket. Dopo che il server elabora la richiesta, il client riceve un pacchetto di risposta che include un messaggio di successo o di errore.
Le operazioni sul server sono gestite attraverso un'architettura modulare, dove ogni tipo di richiesta è associata a una specifica classe che implementa un'azione lato server, come `InserisciEsameServerAction` per l'aggiunta di esami. Quando il server riceve una richiesta, il `ClientHandler` individua l'azione corrispondente utilizzando una mappa predefinita e invoca il metodo `execute` dell'azione. Questo garantisce una gestione organizzata e facilmente estensibile delle operazioni supportate dal server.
Per quanto riguarda **la persistenza**, il server salva automaticamente i dati su file quando viene arrestato. Durante il riavvio, i dati vengono caricati dal file, ripristinando lo stato precedente. Questa scelta, pur semplice, garantisce che il sistema non perda informazioni importanti, come gli appelli creati o le prenotazioni effettuate.
La robustezza dell'applicazione è assicurata attraverso una **gestione accurata delle eccezioni**, sia lato client che lato server. Ad esempio, il client gestisce situazioni come l'assenza di connessione con il server o errori di input da parte dell'utente, mostrando messaggi informativi appropriati. Sul server, richieste non valide o eccezioni durante l'elaborazione sono trattate inviando al client un pacchetto di errore con dettagli utili.
Dal punto di vista del flusso, il client riceve comandi dall'utente tramite un menù interattivo. Quando viene selezionata un'operazione, il programma raccoglie i dati necessari, come dettagli sull'esame, li invia al server e attende una risposta. Il server, una volta ricevuta la richiesta, identifica il tipo di operazione, elabora i dati e restituisce un messaggio di successo o di errore, che il client visualizza nella console.
Infine, il design del sistema è pensato per essere modulare e scalabile. Ogni funzionalità è incapsulata in componenti (classi) indipendenti, facilitando l'aggiunta di nuove operazioni.

L'uso di una **struttura thread-safe** sul server garantisce che più clienti possano interagire contemporaneamente senza causare problemi di consistenza dei dati.

Interazione nel Flusso Segreteria(Client) - Università

Segreteria (Client)

`Segreteria`: Classe principale del sistema della segreteria (main). Inizia il processo creando un'istanza di `SegreteriaClient` e `SegreteriaMenu`. Gestisce il ciclo di vita del client e avvia l'interazione con l'utente tramite il menu console.

`SegreteriaMenu`: Fornisce un'interfaccia console per interagire con l'utente. Consente di scegliere operazioni come aggiungere o eliminare un appello, raccoglie i dati necessari e invoca i metodi del `SegreteriaClient`.

`SegreteriaClient`: Implementa la logica di comunicazione con il server universitario. Per ogni operazione, costruisce un oggetto `Packet` con i dati della richiesta, lo invia al server tramite socket, e gestisce la risposta ricevuta.

Server Universitario

`UniversityServer`: È il server principale che gestisce le operazioni relative agli esami e alle prenotazioni. Rimane in ascolto sulla porta 12345 per richieste dalla segreteria. Ogni connessione è gestita in un thread separato tramite `ClientHandler`.

`ClientHandler`: Elabora le richieste ricevute dalla segreteria. Identifica l'azione richiesta tramite il campo request dell'oggetto `Packet` e invoca la classe di azione corrispondente.

Azioni Specifiche:

- **`InserisciEsameServerAction`**: Gestisce la logica per aggiungere un nuovo esame alla lista degli esami del server.
- **`EliminaEsameServerAction`**: Rimuove un esame dalla lista sulla base del codice fornito.

Flusso Completo

Aggiunta di un Appello

Interazione con l'Utente:
- `SegreteriaMenu` chiede all'utente i dettagli del nuovo appello e crea un oggetto `Esame` con i dati raccolti.
- Invoca il metodo `aggiungiEsame` di `SegreteriaClient`.

Invio della Richiesta:
- `SegreteriaClient` costruisce un `Packet` con richiesta INSERISCI_ESAME e l'oggetto `Esame` creato in precedenza.
- Apre un socket verso `UniversityServer` e invia il pacchetto tramite `ObjectOutputStream`.

Elaborazione sul Server:
- `UniversityServer` riceve la richiesta tramite `ClientHandler`.
- `InserisciEsameServerAction` aggiunge l'esame alla lista e restituisce un pacchetto di risposta con l'esito.

Risposta alla Segreteria:
- `SegreteriaClient` legge il pacchetto di risposta e notifica l'utente tramite messaggi sulla console.

Eliminazione di un Appello

Interazione con l'Utente:
- `SegreteriaMenu` chiede all'utente il codice dell'esame da eliminare e invoca il metodo `eliminaEsame` di `SegreteriaClient`.

Invio della Richiesta:
- `SegreteriaClient` costruisce un `Packet` con richiesta ELIMINA_ESAME e il codice dell'esame.
- Apre un socket verso `UniversityServer` e invia il pacchetto.

Elaborazione sul Server:
- `UniversityServer` riceve la richiesta tramite `ClientHandler`.
- `EliminaEsameServerAction` verifica se l'esame esiste, lo rimuove se presente, e restituisce un pacchetto con l'esito.

Risposta alla Segreteria:
- `SegreteriaClient` legge il pacchetto di risposta e mostra il risultato all'utente.

Studente / Segreteria (Server) / Server Universitario

L'architettura del sistema prevede un'interazione a tre livelli tra il **client studente**, il **server della segreteria**, e il **server universitario**. Lo studente utilizza un'interfaccia console per interagire con il sistema, invocando operazioni specifiche come visualizzare appelli, prenotare esami o accedere alle proprie prenotazioni. Queste richieste vengono trasmesse al server della segreteria, che funge da intermediario tra lo studente e il server universitario.
Quando il client (studente) **si avvia**, l'utente deve autenticarsi utilizzando nome, cognome e numero di matricola. L'applicazione invia un oggetto `Packet` contenente i dati dell'utente e la richiesta di autenticazione al server della segreteria `SegreteriaServer`. Questo pacchetto viene elaborato da un handler specifico per il login, `LoginAction` . Se l'autenticazione ha successo, il server universitario conferma l'identità dello studente, e il server della segreteria inoltra il risultato al client.
Dopo il login, il menu principale dello studente consente di selezionare diverse operazioni. Ad esempio, se lo studente sceglie di visualizzare tutti gli appelli disponibili, il client costruisce una richiesta di tipo VISUALIZZA_ESAME, che viene inviata al server della segreteria. Quest'ultimo inoltra la richiesta al server universitario. Il server universitario elabora la richiesta accedendo alla sua lista di esami, quindi restituisce al server della segreteria un oggetto `Packet` contenente la lista di appelli. Infine, il server della segreteria invia questa lista al client studente, che la visualizza nella console.
Nel caso in cui lo studente decida di prenotare un appello, il flusso è simile. Il client crea un oggetto `Packet` contenente la richiesta PRENOTA_ESAME e i dati necessari (matricola e codice esame). La richiesta è inviata al server della segreteria, che la inoltra al server universitario. Il server universitario verifica se l'appello esiste e se lo studente non è già iscritto, quindi aggiorna i dati delle prenotazioni. La risposta di conferma o errore è trasmessa al client attraverso il server della segreteria, con dettagli come il numero progressivo della prenotazione.
Le operazioni come visualizzare gli appelli di un corso specifico o le prenotazioni seguono una logica analoga. Per ogni richiesta, il client studente invia i dati pertinenti al server della segreteria, che li trasmette al server universitario. Il server universitario elabora la richiesta e invia i risultati al server della segreteria, che li restituisce al client.
Questo design consente di mantenere un'interazione modulare e separata. Il server della segreteria agisce come una barriera intermedia, isolando la logica del server universitario dal client studente. L'uso del protocollo TCP/IP garantisce una comunicazione affidabile tra i vari componenti del sistema. La **serializzazione** degli oggetti permette la trasmissione efficiente di dati complessi come oggetti `Packet` o liste di esami. La **gestione accurata delle eccezioni** assicura un funzionamento robusto, con messaggi di errore appropriati in caso di problemi di connessione o richieste non valide.

Interazione nel Flusso Studente(Client) – Segreteria(Server) – Università(Server)

Studente

`MenuStudente`: Gestisce l'interazione con l'utente attraverso un menu console. Ogni operazione selezionata (visualizzazione di esami, prenotazioni, ecc.) invoca metodi della classe `StudenteClient`.

`StudenteClient`: Implementa la logica di comunicazione tra il client e il server della segreteria. Per ogni richiesta, costruisce un oggetto `Packet` contenente i dati della richiesta, lo invia al server della segreteria, e gestisce le risposte.

Server della Segreteria

`SegreteriaServer`: Accetta connessioni in ingresso dai client (studente) sulla porta 54321. Per ogni connessione, crea un thread separato utilizzando la classe `SegreteriaClientHandler`.

SegreteriaClientHandler: Elabora le richieste ricevute dai client (studente). Ogni richiesta è identificata dal campo `request` del pacchetto e gestita da un'azione specifica implementata da classi come `LoginAction`, `VisualizzaEsameAction`, ecc.

Azioni Specifiche:

- **`LoginAction`**: Gestisce l'autenticazione degli studenti, inoltrando la richiesta al server universitario.
- **`VisualizzaEsameAction`**: Recupera tutti gli esami disponibili dal server universitario.
- **`VisualizzaEsameCorsoAction`**: Recupera gli esami relativi a uno specifico corso dal server universitario.
- **`PrenotaEsameAction`**: Inoltra al server universitario la richiesta di prenotazione di un esame.
- **`PrenotazioniStudenteAction`**: Recupera le prenotazioni di uno studente dal server universitario.

Server Universitario

`UniversityServer`: Accetta richieste inoltrate dal server della segreteria sulla porta 12345. Ogni richiesta è elaborata da un thread separato gestito dalla classe `ClientHandler.`

`ClientHandler`: Elabora le richieste ricevute. Ogni richiesta è associata a una specifica azione server implementata da classi come `LoginServerAction`, `VisualizzaEsameServerAction`, ecc.

Azioni Specifiche:

- **`LoginServerAction`**: Verifica l'autenticazione dello studente confrontando i dati forniti con quelli salvati.
- **`VisualizzaEsameServerAction`**: Recupera la lista degli esami disponibili.

- **`VisualizzaEsameCorsoServerAction`**: Recupera gli esami relativi a un determinato corso.
- **`PrenotaEsameServerAction`**: Gestisce la logica per prenotare un esame. Aggiorna i dati delle prenotazioni verificando la disponibilità e restituisce un esito.
- **`PrenotazioniStudenteServerAction`**: Recupera gli esami prenotati da uno studente.

Flusso Completo

Autenticazione dello Studente:

- `MenuStudente` → `StudenteClient` invia una richiesta LOGIN al → `SegreteriaServer`.
- `SegreteriaServer` gestisce la richiesta tramite `LoginAction`, che la inoltra al → `UniversityServer`..

Visualizzazione Esami:

- `MenuStudente` → `StudenteClient` invia una richiesta VISUALIZZA_ESAME al → `SegreteriaServer`.
- `SegreteriaServer` gestisce la richiesta tramite `VisualizzaEsameAction`, che la inoltra al → `UniversityServer`.

Visualizzazione Esami di un Corso:

- `MenuStudente` → `StudenteClient` invia una richiesta VISUALIZZA_ESAME_CORSO al → `SegreteriaServer`.
- `SegreteriaServer` gestisce la richiesta tramite `VisualizzaEsameCorsoAction`, che la inoltra al → `UniversityServer`.

Prenotazione di un Esame:

- `MenuStudente` → `StudenteClient` invia una richiesta PRENOTA_ESAME al → `SegreteriaServer`.
- `SegreteriaServer` gestisce la richiesta tramite `PrenotaEsameAction`, che la inoltra al → `UniversityServer`.

Visualizzazione Prenotazioni:

- `MenuStudente` → `StudenteClient` invia una richiesta VISUALIZZA_PRENOTAZIONI_STUDENTE al → `SegreteriaServer`.
- `SegreteriaServer` gestisce la richiesta tramite `PrenotazioniStudenteAction`, che la inoltra al → `UniversityServer`.

Parti rilevanti del codice

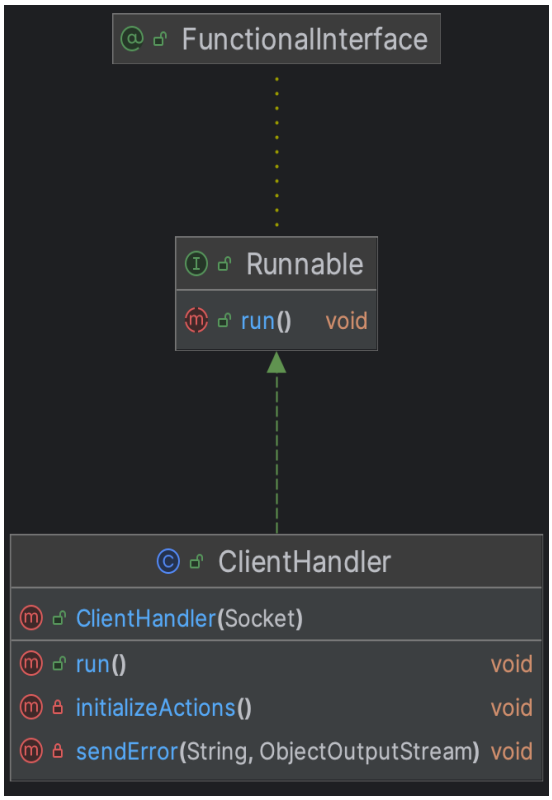
Server Universitario

UniversityServer.Java

La classe **`UniversityServer`** è il componente centrale del sistema che gestisce esami e prenotazioni di appelli. Implementata come singleton, garantisce un'unica istanza condivisa ed è serializzabile per preservare lo stato tra sessioni. La classe utilizza strutture dati come `esamiList (lista di esami)` e `prenotazioni (mappa di matricole a codici esame)` per organizzare i dati.
Tra i metodi principali, **`aggiungiEsame(Esame esame)`** e **`rimuoviEsame(Esame esame)`** permettono rispettivamente di aggiungere o eliminare un esame, mentre **`prenotaEsame(int matricola, long codiceEsame)`** gestisce le prenotazioni, verificando la validità dell'operazione e aggiornando i dati. I metodi **`salvaDati()`** e **`caricaDati()`** si occupano della persistenza, serializzando su file lo stato di esami e prenotazioni per garantirne la conservazione.
Il metodo **`startServer()`** avvia il server, che rimane in ascolto sulla porta 12345. Ogni connessione è gestita in un thread separato tramite `ClientHandler`, il quale inoltra le richieste a specifiche azioni come `InserisciEsameServerAction` o `EliminaEsameServerAction`. Queste azioni eseguono la logica applicativa richiesta, aggiornando i dati del server e inviando le risposte ai client.

La sincronizzazione nei metodi principali assicura thread-safety, consentendo accessi concorrenti senza compromissione dei dati. La progettazione modulare separa nettamente la gestione degli esami, delle prenotazioni e della persistenza, garantendo un'architettura scalabile e facilmente manutenibile. La `classe UniversityServer` è il cuore del sistema, fornendo un'interfaccia robusta e centralizzata per tutte le operazioni legate agli esami universitari.

ClientHandler.java



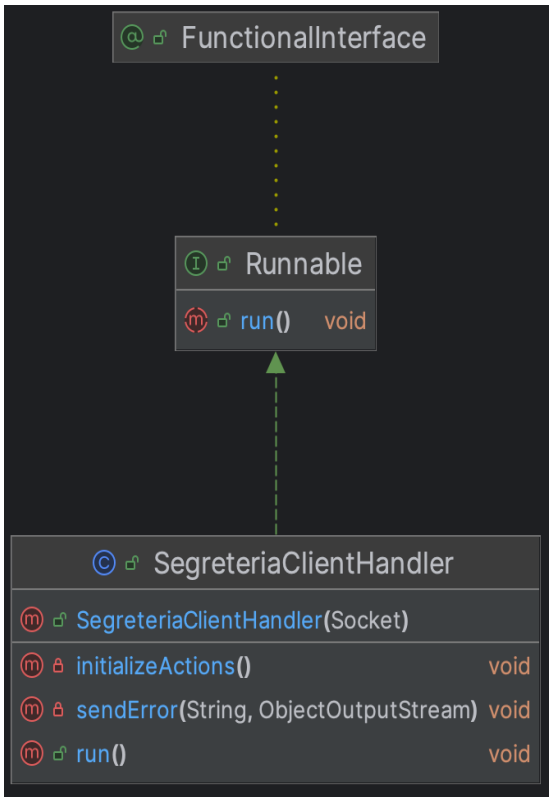
La classe **`ClientHandler`**, che implementa l'interfaccia **`Runnable`**, gestisce le richieste client in modo parallelo. Ogni istanza è associata a un socket, consentendo di elaborare ogni connessione in un thread separato.
Il costruttore **`ClientHandler(Socket socket)`** inizializza il socket, mentre il metodo **`initializeActions()`** configura una mappa che associa le richieste client (es. LOGIN, INSERISCI_ESAME) a specifiche classi di azione, come `LoginServerAction` o `InserisciEsameServerAction`.
Nel metodo principale **`run()`**, la classe riceve un oggetto `Packet` dal client tramite il socket e successivamente identifica l'azione richiesta usando la mappa configurata e infine esegue l'azione o invia un errore tramite **`sendError(String message, ObjectOutputStream output)`**.
Questa classe supporta modularità ed estensibilità tramite l'uso di mapping di azioni, garantendo parallelismo con il supporto di thread separati e una gestione centralizzata degli errori. È un elemento chiave per processare in modo scalabile le richieste verso il server.

Segreteria Server

SegreteriaServer.java

La classe **`SegreteriaServer`** funge da intermediario tra i client studente e il server universitario, gestendo le connessioni client in modo concorrente. Utilizzando un **`ServerSocket`**, il server rimane in ascolto sulla porta **54321** (definita dalla costante `SEGRETERIA_SERVER_PORT`) e accetta connessioni in ingresso.
Ogni connessione client viene gestita tramite un thread separato, avviato con un'istanza di **`SegreteriaClientHandler`**, che si occupa della logica applicativa. Il metodo main avvia il server in un ciclo continuo, assicurando una gestione scalabile e parallela delle richieste client. Eventuali eccezioni di I/O vengono gestite per garantire stabilità.
La classe è progettata per separare le responsabilità: il server accetta connessioni, mentre la logica di elaborazione è delegata ai thread. Questo approccio favorisce modularità, estensibilità e alta scalabilità per ambienti multiutente.

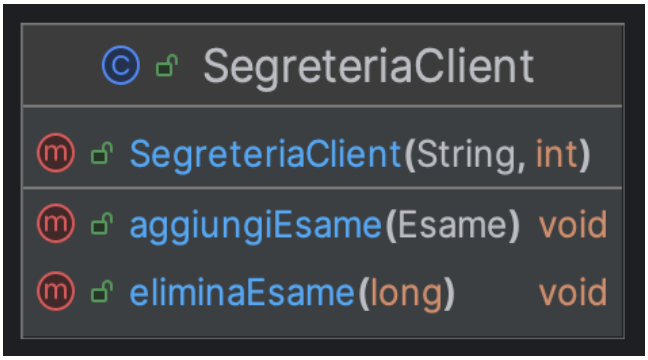
SegreteriaClientHandler.java



La classe **`SegreteriaClientHandler`** si occupa di gestire le richieste provenienti dai client connessi al server della segreteria. Ogni istanza della classe opera in un thread separato, sfruttando l'implementazione dell'interfaccia **`Runnable`**, e consente un'elaborazione concorrente delle richieste.
All'inizializzazione, il costruttore configura il socket per la comunicazione con il client e prepara una mappa di azioni disponibili attraverso il metodo **`initializeActions()`**, associando a ciascuna richiesta (come LOGIN o VISUALIZZA_ESAME) una specifica implementazione della logica corrispondente tramite l'interfaccia `ISegreteriaServerAction`.
Durante l'esecuzione del metodo **`run()`**, la classe legge un oggetto `Packet` inviato dal client contenente i dettagli della richiesta. Utilizzando la mappa configurata, individua l'azione richiesta e ne esegue la logica. Se la richiesta non è riconosciuta, il metodo **`sendError()`** viene utilizzato per inviare al client un pacchetto con un messaggio di errore.
La progettazione modulare e concorrente di **`SegreteriaClientHandler`** permette di gestire in modo efficace e scalabile più connessioni client simultanee. Grazie alla mappa di azioni, è possibile aggiungere nuove funzionalità al sistema senza modificare la struttura della classe, garantendo così manutenibilità ed estensibilità.

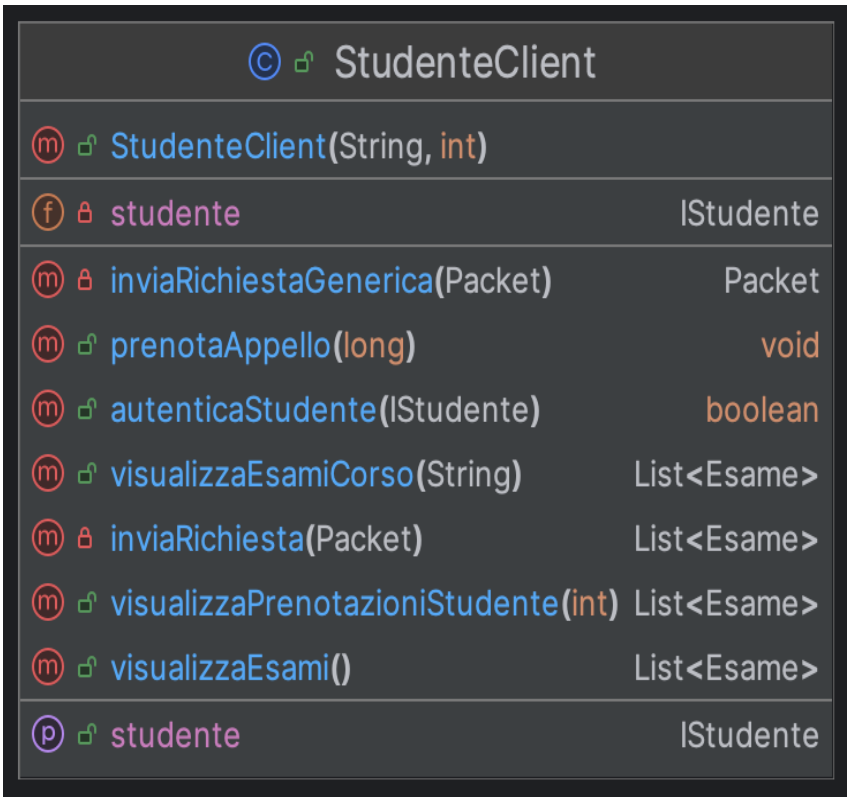
Client

SegreteriaClient.java



La classe **`SegreteriaClient`** fornisce la logica per comunicare con il server universitario, gestendo operazioni come l'aggiunta e l'eliminazione di appelli. È progettata per stabilire connessioni `socket`, inviare richieste al server e ricevere risposte in modo strutturato.
Durante l'inizializzazione, il costruttore accetta l'indirizzo IP del server e la porta su cui quest'ultimo è in ascolto, memorizzandoli in variabili `final`. Questo consente di stabilire connessioni verso il server specificato per ogni richiesta.
Il metodo **`aggiungiEsame(Esame esame)`** costruisce un pacchetto `Packet` con la richiesta INSERISCI_ESAME e i dettagli dell'esame. Apre una connessione socket, invia il pacchetto al server tramite `ObjectOutputStream`, e legge la risposta tramite `ObjectInputStream`. A seconda del contenuto del pacchetto di risposta, notifica l'utente del successo o di eventuali errori nell'inserimento.
Il metodo **`eliminaEsame(long codiceEsame)`** segue una logica simile. Costruisce un pacchetto `Packet` con la richiesta ELIMINA_ESAME e il codice dell'esame da eliminare, invia il pacchetto al server e gestisce la risposta. Anche qui, i messaggi di successo o errore vengono comunicati all'utente in base alla risposta ricevuta.
La gestione degli errori è curata attraverso il trattamento di eccezioni come `ConnectException` o `IOException`, che vengono intercettate per informare l'utente di problemi di rete o di input/output.
Questa classe è fondamentale per abilitare il client della segreteria a comunicare con il server universitario, garantendo una gestione chiara e modulare delle operazioni richieste.

StudenteClient.java



La classe **`StudenteClient`** gestisce la comunicazione tra il client studente e il server della segreteria. Attraverso questa classe, uno studente può autenticarsi, visualizzare esami, prenotare appelli e recuperare le proprie prenotazioni. La classe stabilisce connessioni socket con il server configurato, invia richieste sotto forma di oggetti `Packet`, e interpreta le risposte ricevute.
Il costruttore accetta l'indirizzo IP e la porta del server, configurando i parametri necessari per la connessione. L'oggetto `IStudente` rappresenta lo studente autenticato e viene memorizzato una volta completata l'autenticazione tramite il metodo autenticaStudente. Ogni operazione come prenotaAppello, visualizzaEsami o visualizzaPrenotazioniStudente crea un pacchetto di richiesta (`Packet`) che viene inviato al server tramite i metodi privati `inviaRichiestaGenerica` o `inviaRichiesta`. Questi metodi stabiliscono la connessione socket, inviano il pacchetto e leggono la risposta, garantendo la gestione di errori di connessione o di risposta non valida.
L'autenticazione verifica l'identità dello studente inviando un pacchetto di tipo LOGIN, mentre le prenotazioni e le richieste di visualizzazione utilizzano identificatori come PRENOTA_ESAME o VISUALIZZA_ESAME. Le risposte, che possono contenere liste di esami o conferme, vengono elaborate e restituite al chiamante.
La progettazione modulare e l'uso di metodi generici per inviare richieste al server rendono la classe facilmente estensibile. La gestione centralizzata degli errori e la separazione delle operazioni logiche garantiscono robustezza e mantenibilità, fornendo un'interfaccia essenziale per le interazioni del client studente con il sistema.

Istruzioni su compilazione ed esecuzione

Fare riferimeto al file README.md presente nella repository del progeto all’indirizzo https://github.com/alexdist/Gestionale_Universita/blob/main/README.md