

Corso di Ingegneria del Software Deliverable di progetto	2022-2023
---	-----------

“Ingegneria del Software” 2022-2023

Docente: Prof. Angelo Furfaro

<Esami On-line >

Data	<22-06-2023>
Documento	Documento Finale – D3

Team Members		
Nome e Cognome	Matricola	E-mail address
Giovanni Pascuzzi	220183	pscgnn01m06c352p@st udenti.unical.it

Sommario

Sommario

List of Challenging/Risky Requirements or Tasks.....	3
A.Stato dell'Arte	5
B.Raffinamento dei Requisiti	6
A1.Requisiti con prioritizzazione.....	7
A2.Requisiti non Funzionali	11
A3.Scenari Casi d'Uso	13
A4.Excluded Requirements.....	16
A5.Usecase Diagram	18
C.Architettura Software.....	19
C1.The Static View of System and Component Diagram.....	19
C2.The dynamic view of the software architecture: Sequence Diagram	20
D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)	24
E. Scelte Progettuali (Design Decisions)	25
F. Progettazione di Basso Livello	26
G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs)	27
Appendix. Prototype	29

List of Challenging/Risky Requirements or Tasks

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
Riuscire a realizzare moduli che sfruttassero strutture dati native di Java con gli oggetti generati da Google Remote Procedure Call (Grpc).	04-06-2023	18-06-2023	Inizialmente si è pensato di creare oggetti wrapper per ogni singolo oggetto generato da Grpc, poi, studiando e comprendendo il funzionamento degli stessi, si è proceduto nell'uso di questi oggetti, seppur con qualche difficoltà. E' stato richiesto la modifica della inner classe Appello presente nella classe EsamiOnline (lato server) per poter riadattare il comportamento del metodo hashCode() così da avere un output deterministico nell'uso dei vari metodi di accesso, modifica e verifica della struttura dati HashMap.
Riuscire a gestire i file Json quali misura di serializzazione degli oggetti offerti da Grpc	12-06-2023	16-06-2023	Per la gestione di questa task si è pensato prima di usare la libreria Jackson (idea abbandonata in quanto l'ultima versione di Grpc non risultava compatibile con Jackson) per poi usare la libreria Gson, sviluppata sempre da Google e dunque più che compatibile con Grpc.
Decidere quale linguaggio di programmazione usare	03-06-2023	03-06-2023	Tra i possibili linguaggi individuati per portare a termine il progetto vi erano Python e Java; il primo presentava un ecosistema di classi, generate da Grpc, molto più semplice da intuitivo, mentre Java, al contrario, presentava un ecosistema di classi molto complesso. Nonostante ciò si è scelto di proseguire lo sviluppo tramite Java perché si aveva una maggiore dimestichezza con la sintassi dello stesso nell'ambito del paradigma della

Corso di Ingegneria del Software Deliverable di progetto	2022-2023
---	-----------

			programmazione orientata agli oggetti. L'eventuale scelta di Python, in fase di conclusione del progetto, è stata rivalutata, in quanto il principale vantaggio prima descritto sarebbe risultato estremamente comodo per alcuni problemi incontrati durante lo sviluppo del progetto.
Gestire un timer legato all'esame che si può sostenere	18-06-2023	19-06-2023	Vi era la necessità di impostare un tempo massimo entro il quale poter selezionare una risposta tra quelle proposte durante l'esame. Inizialmente si era pensato di utilizzare un thread e di utilizzare la funzione "sleep()" dell'omonima classe, ma poi, in virtù del fatto che si voleva fornire all'utente la possibilità di sottomettere una risposta prima dei cinque minuti stabiliti si è deciso di far rimanere in attesa il thread su un monitor (nativo di java) così da poterlo risvegliare in modo asincrono tramite una funzione richiamata da un componente java.swing JButton.

A. Stato dell'Arte

Sistemi di gestione degli esami online:

Attualmente, molte università e istituzioni educative offrono sistemi di gestione degli esami online. Questi sistemi consentono agli studenti di svolgere gli esami tramite modalità telematiche, semplificando il processo di prenotazione e svolgimento degli esami. Alcuni esempi noti includono Moodle, Exam.net (utilizzato all'Unical durante il periodo di Emergenza sanitaria legato al Covid-19), Blackboard e Canvas. Questi sistemi spesso offrono funzionalità come la somministrazione di domande a risposta multipla, la valutazione automatica delle risposte e la generazione di punteggi.

Utilizzo di gRPC per lo sviluppo di sistemi distribuiti:

gRPC (Google Remote Procedure Call) è un framework RPC (Remote Procedure Call) moderno e ad alte prestazioni sviluppato da Google. Consente la comunicazione tra servizi distribuiti attraverso la definizione di un'interfaccia di servizio condivisa e l'utilizzo di protocolli di trasporto efficienti come HTTP/2. gRPC supporta numerosi linguaggi di programmazione e offre funzionalità avanzate come streaming bidirezionale e autenticazione. La scelta di utilizzare gRPC per il progetto EsamiOnLine consentirà una comunicazione efficiente tra l'applicazione client e il server, semplificando lo sviluppo di un sistema distribuito.

B. Raffinamento dei Requisiti

Si desidera progettare e realizzare un sistema software distribuito chiamato EsamiOnLine per consentire agli studenti di un'università telematica di visionare le date dei prossimi appelli, prenotarsi per un appello e partecipare agli esami online.

Requisiti funzionali:

1. *Visionare le date dei prossimi appelli previsti.*
2. *Prenotarsi per un appello fornendo il numero di matricola e il codice fiscale.*
3. *Partecipare a un appello, rispondendo a 10 domande a risposta multipla proposte dal docente. Ogni risposta può essere fornita entro 5 minuti dalla sua proposta.*
4. *Ogni risposta corretta vale 3 punti, una risposta errata non vale punti, e una risposta non fornita vale -1 punto.*
5. *Alla fine dell'appello, lo studente riceve un modulo con tutte le risposte esatte e il punteggio ottenuto.*
6. *Dopo l'appello, lo studente può disconnettersi o passare a sostenere un altro esame.*

Requisiti non funzionali:

1. *L'applicazione client deve avere una GUI, preferibilmente web-based.*
2. *La componente server deve avere una GUI di amministrazione.*
3. *Opzionalmente, la componente server può utilizzare persistenza delle informazioni in memoria secondaria tramite un database o file.*
4. *Utilizzare il framework gRPC per implementare il sistema software distribuito.*

A.1 Servizi (con prioritizzazione)

Visionare le date degli appelli previsti

- *Descrizione: Fornisce agli studenti la possibilità di visualizzare le date dei prossimi appelli disponibili per ogni corso.*
- *Soluzione concettuale: Implementazione di una funzionalità per recuperare e presentare le date degli appelli in modo chiaro e accessibile per gli studenti, tramite un componente JTextArea e un JButton.*
- *ID del servizio: S1*
- *Importanza: Alta*
- *Complessità: Bassa*

Prenotarsi per un appello

Descrizione: Consente agli studenti di prenotarsi per un determinato appello fornendo il proprio numero di matricola, codice fiscale, nome appello (identificante per motivi descritti successivamente) e data appello.

Soluzione concettuale: Creazione di un'interfaccia utente intuitiva per la prenotazione degli appelli, validazione dei dati inseriti dagli studenti (lato server) e registrazione delle prenotazioni effettuate.

ID del servizio: S2

Importanza: Alta

Complessità: Media

Partecipare ad un appello

Descrizione: Permette agli studenti di partecipare agli appelli effettuando le domande proposte e fornendo le risposte entro i limiti di tempo stabiliti.

Soluzione concettuale: Implementazione di un sistema interattivo in cui gli studenti ricevono le domande, selezionano le risposte e le sottomettono entro il tempo previsto. Il sistema deve calcolare (lato server) automaticamente i punteggi in base alle risposte fornite.

ID del servizio: S3

Importanza: Alta

Complessità: Alta

Generazione del modulo con le risposte esatte e il punteggio ottenuto

Descrizione: Dopo la partecipazione all'appello, viene generato un modulo per lo studente che contiene le risposte corrette e il punteggio ottenuto in base alle risposte fornite.

Soluzione concettuale: Creazione di un modulo generico, che elenca le domande, le risposte corrette e il punteggio ottenuto. Il modulo deve essere facilmente accessibile e stampabile, per tale ragione è stato scelto il componente JTextArea per rappresentare il modulo (così da favorire manipolazioni dello stesso).

ID del servizio: S4

Importanza: Media

Complessità: Media

Interfaccia di amministrazione

Descrizione: Fornisce un'interfaccia dedicata agli amministratori per gestire gli appelli.

Soluzione concettuale: Sviluppo di una GUI di amministrazione che permetta agli amministratori di creare appelli.

ID del servizio: S5

Importanza: Alta

Complessità: Alta (complessità legata alla gestione di accessi concorrenti)

Gestione della persistenza dei dati

Descrizione: Consente di memorizzare in modo persistente le informazioni relative agli appelli, alle prenotazioni, alle domande.

Soluzione concettuale: Implementazione di un sistema di persistenza dei dati tramite l'utilizzo di una serie di file json, che permetta di archiviare e recuperare le informazioni necessarie per il funzionamento del sistema.

ID del servizio: S6

Importanza: Alta

Complessità: Alta

Testing con JUnit

Descrizione: Utilizzo del framework JUnit per testare in modo automatizzato i moduli significativi del software, garantendo il corretto funzionamento delle funzionalità implementate.

Soluzione concettuale: Creazione di suite di test JUnit per i vari moduli del sistema, verificando che i risultati siano conformi alle aspettative. Si utilizzeranno asserzioni e annotazioni di JUnit per facilitare l'esecuzione dei test.

ID del servizio: S7

Importanza: Media

Complessità: Media

Priorità dell'importanza:

1. S1 (*Visionare le date degli appelli previsti*)
2. S2 (*Prenotarsi per un appello*)
3. S3 (*Partecipare ad un appello*)
4. S5 (*Interfaccia di amministrazione*)
5. S4 (*Generazione del modulo con le risposte esatte e il punteggio ottenuto*)
6. S6 (*Gestione della persistenza dei dati*)
7. S7 (*Testing con JUnit*)

Priorità dell'importanza:

1. S1 (*Visionare le date degli appelli previsti*)
 2. S2 (*Prenotarsi per un appello*)
 3. S3 (*Partecipare ad un appello*)
 4. S5 (*Interfaccia di amministrazione*)
 5. S4 (*Generazione del modulo con le risposte esatte e il punteggio ottenuto*)
 6. S6 (*Gestione della persistenza dei dati*)
 7. S7 (*Testing con JUnit*)
-

A.2 Requisiti non Funzionali

I requisiti non funzionali più importanti per il sistema EsamiOnLine sono i seguenti:

Affidabilità: Assicurare che il sistema sia sempre disponibile e che le funzionalità principali siano operative senza interruzioni, errori critici o problemi di inconsistenza. Deve essere progettato per gestire un alto carico di utenti simultanei e prevenire eventuali guasti o malfunzionamenti che potrebbero compromettere l'esperienza degli studenti durante gli appelli.

Prestazioni: Garantire tempi di risposta rapidi ed efficienza nell'elaborazione delle richieste (anche grazie ad apposite strutture dati) degli studenti durante la visualizzazione delle date degli appelli, la prenotazione e la partecipazione agli appelli. Il sistema deve essere progettato per gestire il carico di lavoro e le operazioni richieste in modo ottimale, riducendo al minimo i tempi di attesa degli utenti.

Usabilità: Fornire un'interfaccia utente intuitiva, facile da usare e ben strutturata per consentire agli studenti di navigare e interagire con il sistema in modo semplice e intuitivo. L'interfaccia dovrebbe essere intuitiva, con indicazioni chiare e guida degli utenti attraverso i passaggi necessari per eseguire le azioni richieste.

Scalabilità: Il sistema deve essere in grado di scalare in modo efficiente per gestire un numero crescente di studenti, appelli e domande senza compromettere le prestazioni o la qualità del servizio. Dovrebbe essere possibile aumentare le risorse di elaborazione e la capacità di archiviazione in modo flessibile per adattarsi alla crescita del sistema.

Manutenibilità: Il sistema deve essere progettato in modo modulare e ben strutturato, con codice pulito e documentazione adeguata. Ciò agevolerà la manutenzione del software nel tempo, facilitando l'aggiunta di nuove funzionalità, la correzione di bug e l'aggiornamento del sistema.

Compatibilità: Il sistema deve essere compatibile con diverse piattaforme, browser e dispositivi, garantendo un'esperienza utente uniforme e di alta qualità su diversi ambienti di utilizzo. Ciò è facilmente ottenibile grazie all'uso intrinseco di http2.0 di Google remote procedure call.

Riusabilità: il sistema deve essere strutturato in modo da garantire la riusabilità di determinati moduli. Per perseguire ciò è stato usato un design pattern Adapter per avvolgere i metodi offerti dallo STUB di Grpc, così da poter riutilizzare l'intero sistema Client su più sistemi di remote call diversi.

A.3 Scenari d'uso dettagliati

<i>Caso d'uso</i>	<i>Visionare le date degli appelli</i>
<i>Tipo</i>	<i>Primario</i>
<i>Precondizione</i>	<i>Lo studente accede all'applicazione identificandosi tramite matricola e codice fiscale</i>
<i>Svolgimento Normale</i>	<i>Lo studente visualizza le date dei prossimi appelli disponibili.</i>
<i>Postcondizione</i>	<i>Lo studente ha accesso alle informazioni sulle date degli appelli e può procedere con la prenotazione.</i>
<i>Descrizione</i>	<i>In seguito alla registrazione mediante matricola e codice fiscale l'utente potrà interagire con una nuova finestra che gli permetterà di scaricare la lista degli appelli, riportando il nome dell'appello (identificante) e la data dello stesso visualizzata secondo il fuso orario del sistema operativo. La lista degli appelli sarà rappresentata su una nuova finestra editabile.</i>

<i>Caso d'uso</i>	<i>Prenotarsi per un appello</i>
<i>Tipo</i>	<i>Primario</i>
<i>Precondizione</i>	<i>Lo studente ha visualizzato le date degli appelli disponibili.</i>
<i>Svolgimento Normale</i>	<i>Lo studente si prenota indicando la data ed il nome dell'appello.</i>
<i>Postcondizione</i>	<i>Lo studente riceve la notifica di avvenuta prenotazione per poi poter partecipare all'appello nella data prevista.</i>
<i>Descrizione</i>	<i>In seguito all'inserimento del nome appello e data appello negli appositi campi (implementati tramite componenti JTextField) l'utente sarà abilitato all'interazione con il pulsante "Prenota Appello", il quale, una volta cliccato, farà comparire un componente JTextArea in cui sarà visualizzato l'esito della prenotazione. In caso di esito positivo l'utente potrà, in futuro, partecipare all'esame. La prenotazione può avvenire solo a determinate condizioni, come ad esempio l'anticipo di almeno 30 minuti dal momento di inizio dell'esame.</i>

<i>Caso d'uso</i>	<i>Partecipare ad un appello</i>
<i>Tipo</i>	<i>Primario</i>

<i>Precondizione</i>	<i>Lo studente è prenotato per un appello e la data e l'ora dell'appello sono arrivate.</i>
<i>Svolgimento Normale</i>	<i>Lo studente accede al laboratorio didattico e si identifica tramite un documento di identità valido. Successivamente, risponde alle domande dell'appello presentate in successione.</i>
<i>Postcondizione</i>	<i>Lo studente ha completato l'appello, riceve un modulo con tutte le risposte esatte e il punteggio ottenuto. Può disconnettersi o passare a sostenere un altro esame.</i>
<i>Descrizione</i>	<i>L'utente, solo se è in anticipo di 30 minuti rispetto alla data prevista, può sostenere l'appello, tramite una nuova finestra apposita che mostrerà ogni 5 minuti una nuova domanda e 3 risposte, più una quarta opzione per ignorare la domanda. L'utente potrà passare da una domanda alla successiva prima dello scadere dei 5 minuti tramite un apposito JButton; l'anticipata sottomissione di una domanda non permetterà all'utente di poter tornare sui suoi passi e modificare la risposta. In fine l'utente riceverà un modulo indicante le domande corrette ad ogni risposta ed il punteggio. L'utente non potrà risostenere l'esame a meno di una nuova</i>

	<i>programmazione dello stesso in data successiva.</i>
--	--

A.4 Excluded Requirements

Durante la stesura del codice sono emersi svariati meccanismi e servizi che avrebbero, se opportunamente implementati, potuto migliorare l'esperienza sia lato client che lato server. I seguenti requisiti non sono stati implementati principalmente per ragioni temporali, in quanto vista la buona modularità del software avrebbero potuto essere implementati con poca difficoltà. I servizi che non sono stati, nella versione attuale del software, aggiunti sono i seguenti:

- 1. possibilità di aggiungere domande e risposte ad un appello tramite interfaccia grafica lato server;*
 - 2. possibilità di rimuovere e/o modificare appelli già presenti lato server;*
 - 3. possibilità di identificarsi univocamente lato client;*
 - 4. possibilità di mantenere uno storico degli appelli sostenuti con relativi risultati;*
 - 5. possibilità di gestire in modo più professionale vari casi limite;*
 - 6. possibilità di gestire l'identità degli oggetti Appello in modo più pulito.*
 - 7. possibilità di visualizzare un timer indicante i minuti mancanti per ogni domanda lato client, nel momento di sostenere un esame;*
 - 8. possibilità di inserire sistemi di sicurezza per tutelare i vari utenti.*
-

A.5 Assunzioni

Si assuma che l'utente non provi a registrarsi mediante matricola e codice fiscale di un altro utente;

Si assuma che chi utilizza il server sappia come compilare dei file json per poter inserire domande e risposte per ogni appello;

Si assuma che ogni appello sia identificato dal solo nome del corso e non anche dalla data;

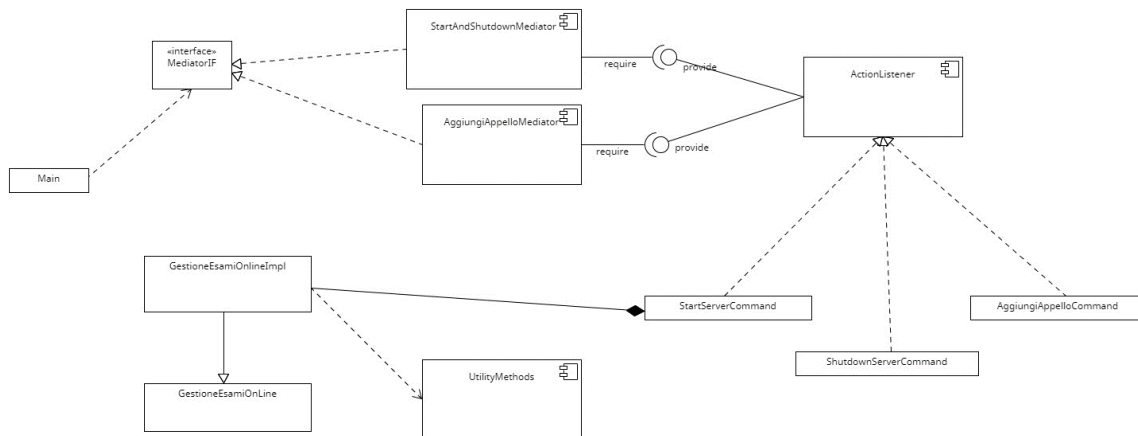
A.6 Use Case Diagrams



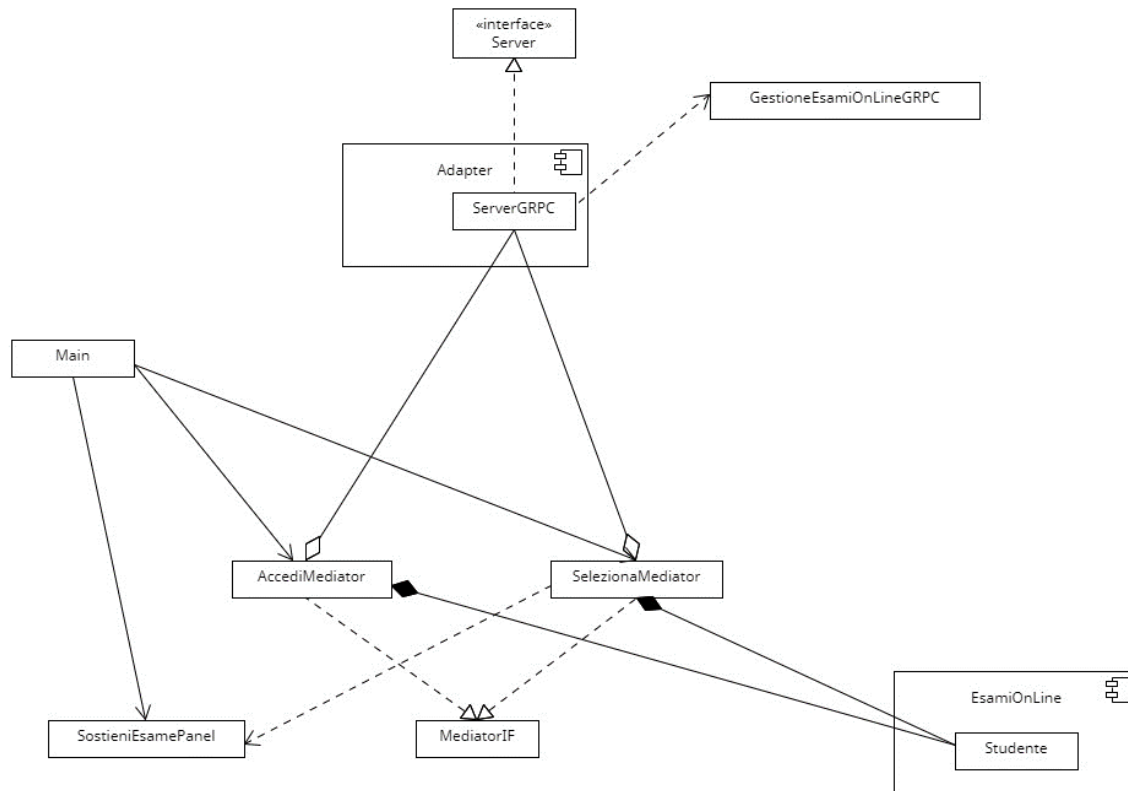
C. Architettura Software

C.1 The static view of the system: Component Diagram

Component diagram lato Server:



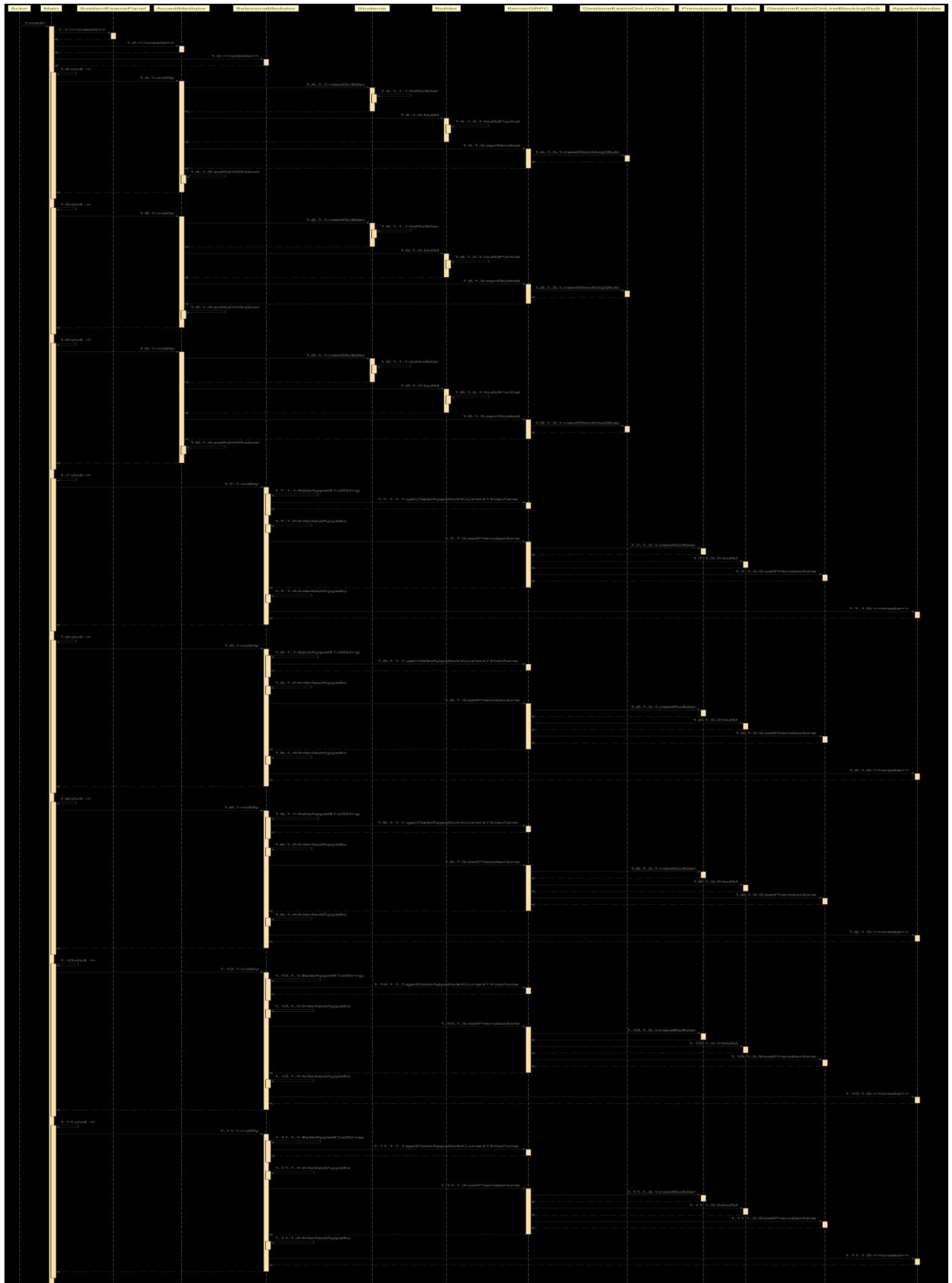
Component diagram lato Client:



E' possibile notare dal diagramma che un'ottimizzazione sarebbe potute essere quella di accoppiare **AccediMediator** e **SelezionaMediator** in un'unica classe, così da favorire un minor accoppiamento con l'esterno.

C.2 The dynamic view of the software architecture: Sequence Diagram

Sequence diagram lato Client:



Corso di Ingegneria del Software Deliverable di progetto	2022-2023
---	------------------

Sequence diagram lato Server:



D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)

L'applicazione offre un meccanismo di persistenza dei dati tramite una serie di file json. L'applicazione server fornisce un'interfaccia grafica per aggiungere Appelli e dunque serializzarli in degli appositi file json. È possibile salvare su disco anche gli oggetti Domande e Risposte, ma non è fornita una procedura grafica per fare ciò.

E. Scelte Progettuali (Design Decisions)

Il progetto è stato realizzato utilizzando i seguenti design pattern:

1. *Adapter;*
2. *Mediator;*
3. *Command;*
4. *Observer;*
5. *Proxy (seppur fornito nativamente da Grpc).*

Il pattern Mediator è stato quello maggiormente usato per poter permettere una facile gestione dell'interfaccia grafica e per poter introdurre politiche di gestione degli eventi.

F. Progettazione di Basso Livello

Proxy:

Il pattern proxy è stato fornito nativamente da Grpc, in particolare, lato client, è possibile utilizzare l'istanza di un oggetto di tipo BlockingStub. E' possibile interagire con questo oggetto come se esso stesso fosse un surrogato del server di cui offre i servizi invocabili tramite dot notation.

Mediator:

Sia lato client che server si è fatto largo uso di questo design pattern così da mediare e disaccoppiare le interazioni tra i vari componenti grafici e i vari oggetti command. L'uso degli oggetti presenti nel package Mediator ci ha permesso di introdurre politiche di gestione che altrimenti non avremmo potuto perseguire tramite il solo uso delle interfacce grafiche e dei loro ActionListener utilizzati, magari, tramite lambda expressions.

Command/Observer:

Inizialmente si è pensato di introdurre degli oggetti che seguissero la filosofia dei design pattern Command, così da poter prima sviluppare la logica del programma su cui poi mappare una interfaccia grafica. In fase di sviluppo, non avendo bisogno di un command handler per introdurre politiche già gestite dai vari mediator, si sono attribuite caratteristiche quasi da Observer a tali oggetti, così da renderli compatibili con i componenti JButton; infatti è possibile notare come tali oggetti implementino la classe ActionListener.

Adapter:

Lato client è stata introdotta una classe Wrapper che include i metodi forniti dal BlockingStub così da fornire un'interfaccia alternativa ai servizi offerti dal server. Così facendo sarà possibile intercambiare l'intero sistema client favorendone notevolmente la riusabilità.

G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs)

Il progetto EsamiOnLine soddisfa i requisiti funzionali (FRs) e non funzionali (NFRs) come segue:

Requisiti funzionali (FRs):

Visionare le date degli appelli: Il sistema permette agli studenti di visualizzare le date dei prossimi appelli previsti. Questo requisito è soddisfatto fornendo una funzionalità nel client che recupera e visualizza su una JTextArea le informazioni sulle date degli appelli dal server.

Prenotarsi per un appello: Gli studenti possono prenotarsi per un appello specifico fornendo il loro numero di matricola e codice fiscale. Il sistema memorizza queste informazioni e in seguito alla compilazione di un form contenente il nome dell'appello e la data procede a gestire la prenotazione lato server. Questo requisito è soddisfatto dal server, che gestisce la prenotazione degli appelli e fornisce le interfacce necessarie per il client per effettuare la prenotazione.

Partecipare ad un appello: Gli studenti prenotati possono partecipare all'appello nel laboratorio didattico. Durante l'appello, vengono presentate in successione 10 domande, e gli studenti devono selezionare le risposte corrette. Il sistema tiene traccia del tempo trascorso e calcola il punteggio finale. Questo requisito è soddisfatto sia dal client che dal server, il primo che fornisce l'interfaccia per la gestione delle domande ed il secondo che fornisce un sistema di calcolo dei punteggi.

Requisiti non funzionali (NFRs):

Identificazione mediante documento di identità valido: Il sistema richiede agli studenti di identificarsi nel laboratorio didattico mediante la coppia di matricola e codice fiscale. Ciò garantisce l'autenticità e l'integrità dell'esame. Questo requisito è soddisfatto mediante l'implementazione di un meccanismo di identificazione nel lato client.

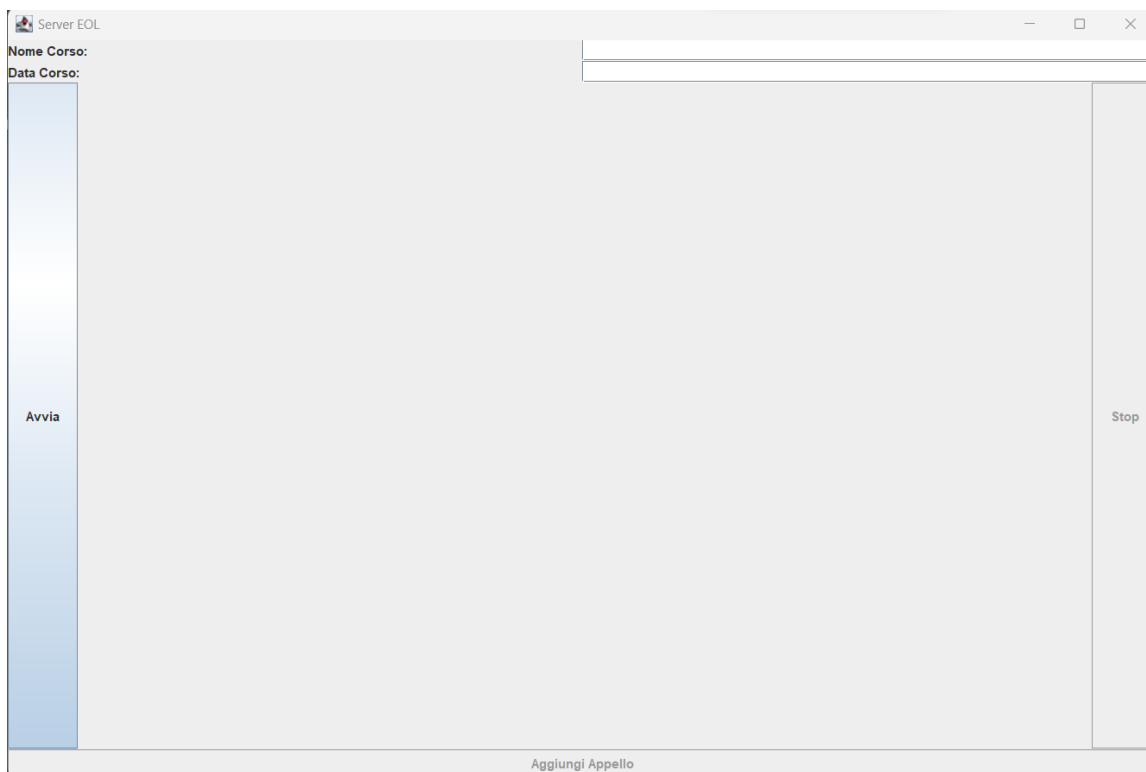
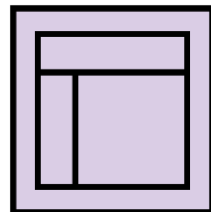
Persistenza delle informazioni: Il sistema può utilizzare un meccanismo di persistenza per memorizzare le informazioni sugli studenti, le prenotazioni degli appelli e i punteggi ottenuti. Ciò consente di mantenere i dati in modo affidabile anche dopo il riavvio del sistema. Questo requisito è soddisfatto tramite l'utilizzo di un sistema di archiviazione persistente nel server.

GUI (Graphical User Interface): L'applicazione client è dotata di una GUI minima, che consente agli studenti di interagire con il sistema in modo intuitivo. Questo requisito è soddisfatto fornendo un'interfaccia utente grafica nel client che consente agli studenti di visualizzare le date degli appelli, prenotarsi e partecipare all'appello in modo interattivo.

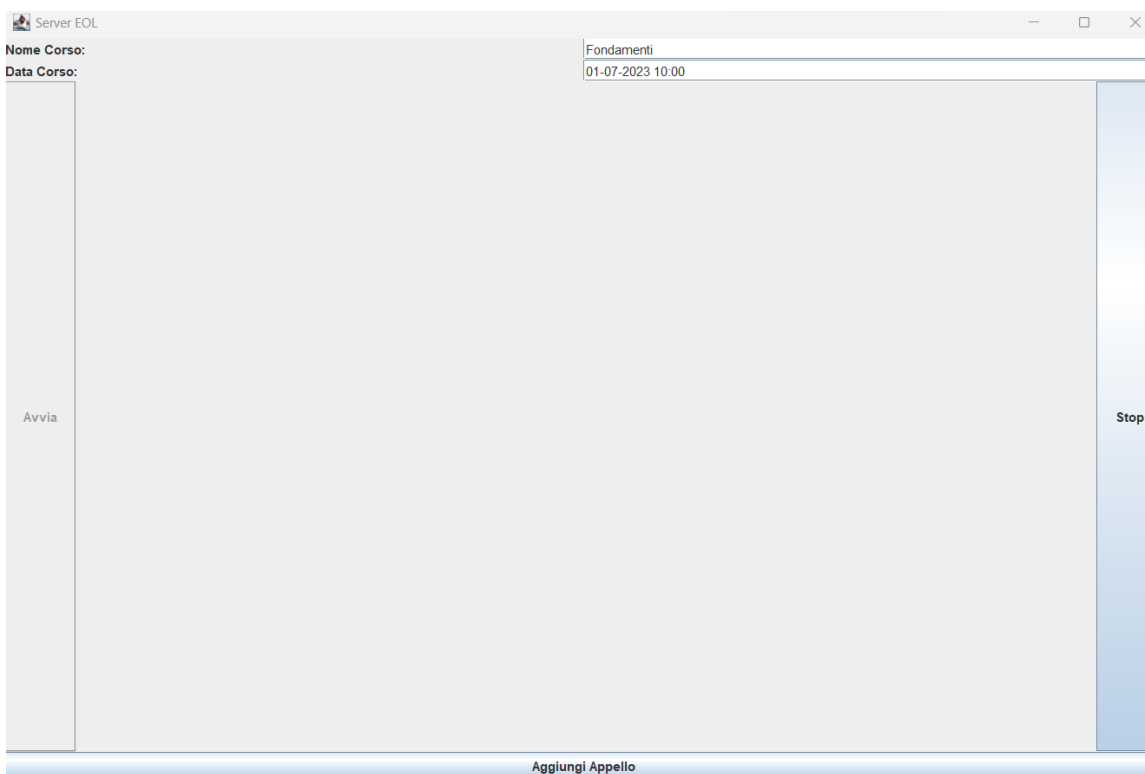
In sintesi, il progetto EsamiOnLine soddisfa i requisiti funzionali fornendo le funzionalità richieste per la visualizzazione delle date degli appelli, la prenotazione e la partecipazione all'appello. Allo stesso tempo, soddisfa i requisiti non funzionali garantendo l'autenticazione degli studenti, gestendo la persistenza delle informazioni e fornendo un'interfaccia utente grafica intuitiva.

Appendix. Prototype

La GUI lato server si presenta come raffigurato nella seguente immagine



Una volta avviato il server e compilati i campi per l'aggiunta di un nuovo appello i vari bottoni cambieranno stato, il nuovo stato della GUI è osservabile nella seguente immagine:



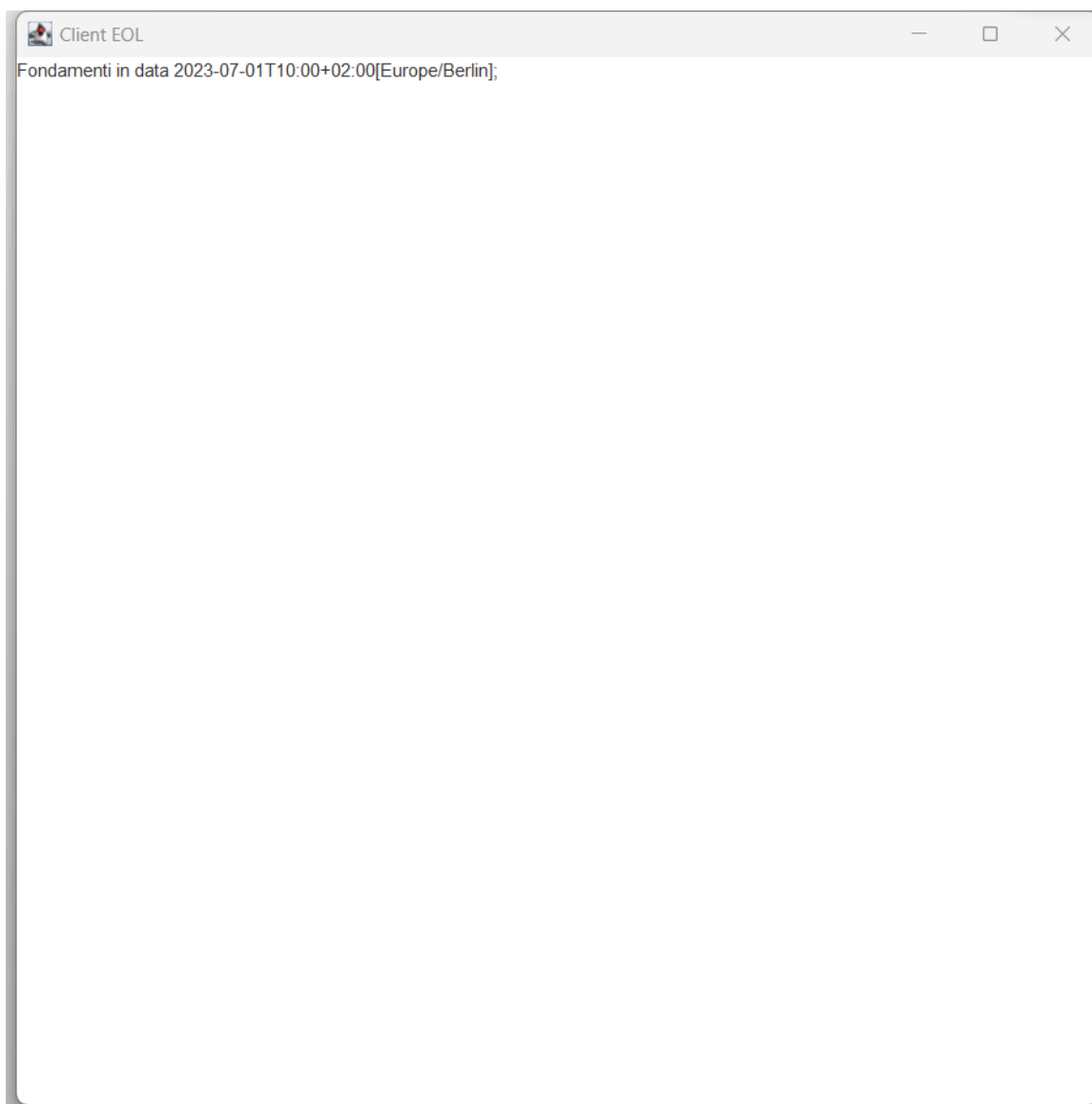
Per quanto riguarda l'interfaccia lato client, all'avvio dell'applicazione ci verrà mostrato un form in cui inserire le nostre credenziali:

A screenshot of a login form. It has a title bar with a small icon and standard window controls. Below the title bar, there are two input fields. The first is labeled "Matricola:" and the second is labeled "Codice Fiscale:". Below these fields is a blue button with the text "Accedi".

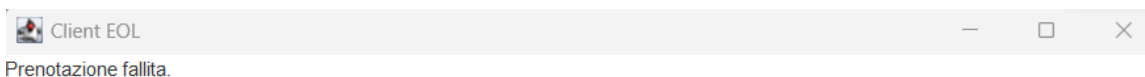
Una volta inserite le credenziali ed aver premuto il pulsante "Accedi", ci sarà presentata la seguente schermata:

The screenshot shows a Java Swing window titled "Client EOL". The window is divided into two main sections. The left section is a light gray area containing two labels: "Nome appello:" and "Data appello:". The right section is a white area divided into two horizontal text input fields. At the bottom of the window, there is a horizontal bar with three buttons: "Visualizza Appelli" (highlighted in blue), "Salva prenotazione", and "Sostieni Appello".

Cliccando sul pulsante “Visualizza Appelli” sarà possibile visualizzare la lista degli appelli in una JTextArea:



Tramite il pulsante “Salva Prenotazione” sarà possibile effettuare una prenotazione a nome dell’utente a cui corrispondono le credenziali inserite inizialmente. In risposta ci verrà mostrato, in una JTextArea, l’esito della prenotazione:



In questo caso la prenotazione è fallita perché l'utente è già prenotato; dunque, in prossimità dell'ora dell'appello sarà possibile sostenere l'esame, tramite la seguente GUI:

Client EOL

Quanto fa 2+2?

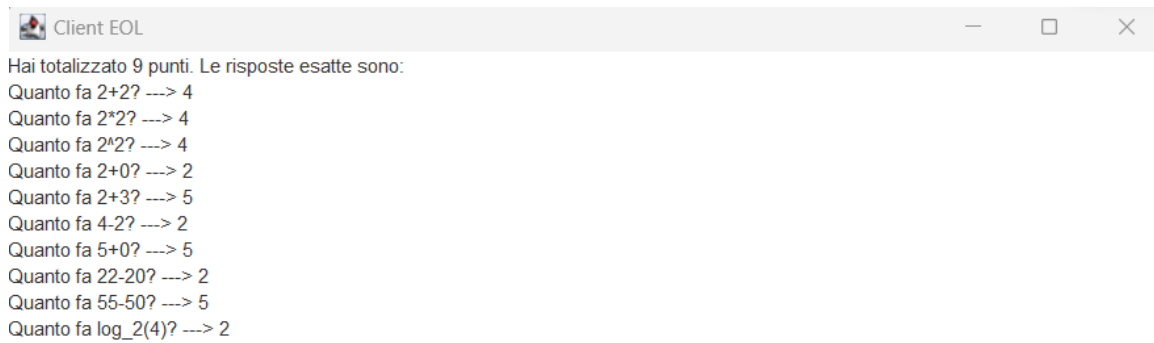
☐ 4 ☐ 5

Domanda numero:1

☐ 2 ☐ Ignora domanda

Conferma risposta

In fine, mediante una JTextArea sarà possibile visionare il punteggio e le domande con le rispettive risposte corrette:



Client EOL

Hai totalizzato 9 punti. Le risposte esatte sono:

- Quanto fa $2+2$? ----> 4
- Quanto fa $2*2$? ----> 4
- Quanto fa 2^2 ? ----> 4
- Quanto fa $2+0$? ----> 2
- Quanto fa $2+3$? ----> 5
- Quanto fa $4-2$? ----> 2
- Quanto fa $5+0$? ----> 5
- Quanto fa $22-20$? ----> 2
- Quanto fa $55-50$? ----> 5
- Quanto fa $\log_2(4)$? ----> 2

Di seguito sono riportati due test effettuati lato client, tramite il framework Junit, per verificare se l'apertura del socket avviene con successo e se il controllo di una prenotazione già esistente avviene, anch'essa, con successo:

```
@Test
public void testPrenotazione()
{
    ServerGRPC server=new ServerGRPC();
    server.apriSocket( host: "127.0.0.1", port: 8989);
    Appello appello= Appello.newBuilder().setNomeCorso("Fondamenti").build();
    Studente studente= Studente.newBuilder().setMatricola(220183).setCodiceFiscale("pscgnn01m06c352p").build();
    boolean esito=server.setPrenotazione(appello,studente);
    Assertions.assertEquals(esito, actual: false);
}
```

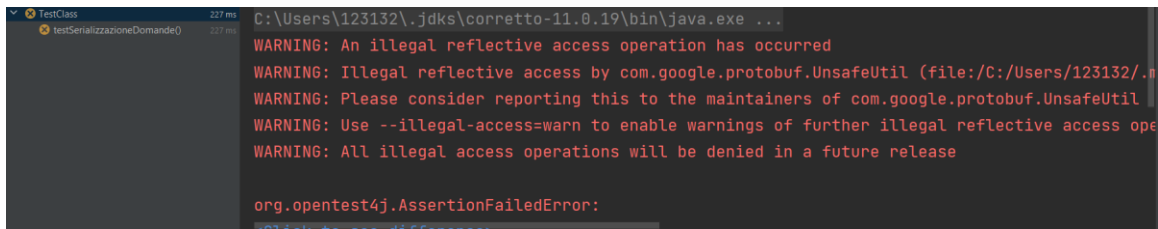
```
@Test
public void testAperturaSocket()
{
    ServerGRPC server=new ServerGRPC();
    server.apriSocket( host: "127.0.0.1", port: 8989);
    Assertions.assertNotEquals(server, actual: null);
}
```

Run: TestClass
Tests passed: 2 of 2 tests - 1 sec 271 ms
testPrenotazione() 1 sec 271 ms
testAperturaSocket() 2 ms

C:\Users\123132\.jdk\corretto-11.0.19\bin\java.exe ...
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.protobuf.UnsafeUtil (file:/C:/Users/123132/.m...
WARNING: Please consider reporting this to the maintainers of com.google.protobuf.UnsafeUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access op...
WARNING: All illegal access operations will be denied in a future release

Come si può notare i test hanno avuto esito positivo. Invece lato server, si è usato un test per inferire la struttura interna degli oggetti generati da GRPC, in particolare si è cercato di capire se il `toString()` dell'oggetto corrispondesse alla versione serializzata dello stesso. Tale test ha avuto esito negativo:

```
@Test
public void testSerializzazioneDomande()
{
    String tmp="{\n" +
        "  \"domande\": [\n" +
        "    \"testoDomanda\": \"Quanto fa 2+2?\",\n" +
        "    \"risposta1\": \"4\",\n" +
        "    \"risposta2\": \"5\",\n" +
        "    \"risposta3\": \"2\"},\n" +
        "  ]\n" +
        "  \"testoDomanda\": \"Quanto fa 2*2?\",\n" +
        "  \"risposta1\": \"4\",\n" +
        "  \"risposta2\": \"5\",\n" +
        "  \"risposta3\": \"2\"},\n" +
        "  \"testoDomanda\": \"Quanto fa 2^2?\",\n" +
        "  \"risposta1\": \"4\",\n" +
        "  \"risposta2\": \"5\",\n" +
        "  \"risposta3\": \"2\"},\n" +
        "  \"testoDomanda\": \"Quanto fa 55-50?\",\n" +
        "  \"risposta1\": \"4\",\n" +
        "  \"risposta2\": \"5\",\n" +
        "  \"risposta3\": \"2\"},\n" +
        "  \"testoDomanda\": \"Quanto fa log_2(4)?\",\n" +
        "  \"risposta1\": \"4\",\n" +
        "  \"risposta2\": \"5\",\n" +
        "  \"risposta3\": \"2\"}\n" +
        "};";
    EsamiOnline.Domande domanda=fromJsonFileDomande( filePath: "fileJsonDomande/"+"Fondamenti".json");
    assertEquals(tmp,domanda.toString());
}
```



The screenshot shows a Java IDE with a test class named 'testSerializzazioneDomande()' running. The output console displays several warnings from the protobuf library and an assertion failure. The warnings are: 'WARNING: An illegal reflective access operation has occurred', 'WARNING: Illegal reflective access by com.google.protobuf.UnsafeUtil (file:/C:/Users/123132/.m2/repository/com/google/protobuf/3.11.4/protobuf-3.11.4.jar) of method java.lang.Object.hashCode()', 'WARNING: Please consider reporting this to the maintainers of com.google.protobuf.UnsafeUtil', 'WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations', and 'WARNING: All illegal access operations will be denied in a future release'. The assertion failure is: 'org.opentest4j.AssertionFailedError: Expected: 41, but was: 1455688888'.

Nota: Per far funzionare l'applicativo lato server è necessario modificare l'hashcode della classe Appello all'interno della classe EsamiOnLine con ciò che segue:

```
@java.lang.Override
public int hashCode() {
    int hash = 41;
    hash = (53 * hash) + getNomeCorso().hashCode();
    memoizedHashCode = hash;
    return hash;
}
```