**“Ingegneria del Software”**

**2019-2020**

**Docente: Prof. Angelo Furfaro**

**Esami online**

|  |  |
| --- | --- |
| **Data** | <gg/mm/aaaa> |
| **Documento** | Documento Finale – D3 |

|  |  |  |
| --- | --- | --- |
| **Team Members** | | |
| **Nome e Cognome** | **Matricola** | **E-mail address** |
| **Alessandro Mileto** | **201262** | **Alessandromileto22@gmail.com** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Sommario

**Non è stata trovata alcuna voce d'indice.**

## 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** |
| **Gestire lo svolgimento di un esame** |  |  | **Pattern State e multithreading** |
| **Gestire la sincronizzazione e l’allineamento delle informazioni** |  |  | **Pattern Observer** |
| **Costruire comandi e rappresentazioni degli stessi o delle informazioni in maniera sistematica** |  |  | **Pattern Builder** |
| **Gestire gli esami** |  |  | **Componente ad hoc** |

A. Stato dell’Arte

A fronte del sopraggiungere della pandemia di SARS-Cov2 (comunemente detto Covid-19) le università di tutto il mondo hanno cominciato ad impiegare soluzioni per far sostenere gli esami universitari online agli studenti. Le piattaforme che si sono affermate sono molteplici, alcune proprietarie, altre gestite da terzi.

Uno dei progetti proprietari più in voga è Respondus. La piattaforma in questione si avvale dell’uso di un browser opportunamente progettato per limitare i tentativi di “cheating” degli utenti; in particolare, forzando l’esaminando a mantenere il focus sulla finestra che espone il foglio di lavoro e la traccia di esame, il browser rileva eventuali sequenze di tasti che possono portare in secondo piano la schermata. Rilevate tali anomalie, una segnalazione al supervisore dell’esame viene prontamente trasmessa (tramite connessione dedicata). Sono bloccate le app di chat, altri browser, le macchine virtuali e le app per il remote desktop (che devono essere chiuse manualmente o possono essere chiuse automaticamente dal browser). Non si possono, inoltre, realizzare catture dello schermo e screenshot e le segnalazioni all’utente esaminando sono tali da essere sempre imposte in primo piano in maniera tale che costui abbia la completa visione di ciò che accade durante la prova. Il browser pocanzi menzionato è utilizzato spesso in concomitanza di Respondus Monitor per monitorare in maniera visiva la condizione dello studente che sta affrontando l’esame. Il monitor si avvale dell’uso dell’intelligenza artificiale per rilevare comportamenti sospetti (es. sguardo dello studente in direzioni differenti dal foglio) e viene lanciato automaticamente a partire da qualsiasi browser (nel caso in cui non si utilizzi il browser proprietario) qualora l’esame a cui si sta accedendo richieda l’uso della piattaforma in questione. Una delle feature più interessanti è la capacità di integrare Respondus (software per il monitoraggio) con CAD e strumenti di grafica, nonché strumenti didattici forniti da terzi (es. Pearson Mylab) per la simulazione, il disegno e la creazione di prototipi in maniera agevole. Si nota, infine, che la piattaforma non richiede registrazione alcuna; ergo, un docente che voglia tenere un esame apre quest’ultimo interfacciandosi con il sito e gli studenti vi accedono previo impiego di un codice fornito precedentemente.

Focalizzando l’attenzione sul concetto di login, è opportuno includere nelle presenti riflessioni sullo stato dell’arte anche la discussione della piattaforma Exam.net . Essa si avvale di un meccanismo di adesione (previa immissione di codice) ad un esame precedentemente aperto da un docente che prevede l’immissione in un form di una serie di informazioni che permettono di compiere una prima identificazione dell’utente nel sistema. Tale procedura non ha, in genere, valenza legale ai fini dell’esame ma serve a comprendere quanti e quali studenti stiano sostenendo la prova e quale sia lo stato della prova del singolo studente. Nel caso in cui dovessero verificarsi dei problemi, il sistema permette di comprendere quale utente ha accusato difficoltà (perché, ad esempio, ha portato in background la finestra dell’esame). Tramite una lista ed un meccanismo di chat privata (usata anche per chiedere informazioni in merito alla traccia di esame o per segnalare problemi e, quindi, garante dell’interattività del processo di svolgimento dell’esame) il docente può verificare cosa è realmente successo e valutare se sia opportuno o meno far proseguire la prova dello studente interessato. In tutto questo, la soluzione Exam.net si avvale spesso dell’uso di un browser simile a LockDown Browser di Respondus e di sorveglianza video effettuata mediante piattaforme come Microsoft Teams o Google Meet; attraverso la videosorveglianza, spessissimo, si consuma il riconoscimento vero e proprio dello studente mediante presa visione da parte del docente del ID universitario e del tesserino del primo.

Tornando a Respondus, il sistema è risultato essere estremamente scalabile, come testimoniato dal fatto che durante il periodo del lockdown mondiale (anno 2020) nel contesto pandemico migliaia di studenti in 1500 università hanno usato la piattaforma in maniera intensiva. Naturalmente, la soluzione ha un costo, dipendente dal numero di sessioni di esame che si vogliono realizzare con l’ausilio della piattaforma e, quindi, richiede la stipulazione di un contratto annuale tra le parti coinvolte (università e società che gestisce Respondus).

B. Raffinamento dei Requisiti

*A partire dai servizi minimali richiesti, raffinate la descrizione dei servizi offerti dal vostro applicativo. Descrivete anche I requisiti non funzionali.*

Al fine di spiegare nella maniera più chiara e meno ambigua possibile i requisiti di un sistema per la presa in carico di esami telematici si andranno ad illustrare i domini di competenza degli stakeholder del sistema. Queste ultime figure, in particolare, si identificano negli studenti universitari e negli amministratori del laboratorio, assunti dall’università al fine di gestire le procedure di conduzione e supervisione degli esami. Tra queste ultime figure, relative al dominio universitario, esistono addetti alla gestione del backend (base di dati contenente le informazioni sugli studenti, gli esami e gli appelli) e attori preposti alla orchestrazione delle prove di esame (supervisori e docenti dei corsi che compongono le tracce da somministrare agli esaminandi). In prima approssimazione si può assumere che i due attori coincidano nella figura del **admin** e questa assunzione è alla base del presente elaborato.

Partendo dal dominio dello studente universitario, costui si aspetta che il discostamento tra la gli esami in modalità telematica ed in modalità in presenza sia minimo; in soldoni, sostenere una prova online deve garantire che vengano rispettati alcune condizioni fondamentali che si traducono direttamente in requisiti funzionali. In primis, l’utente esaminando deve avere la garanzia che il riconoscimento in fase di accesso alla piattaforma avvenga nel modo corretto. Tra i requisiti funzionali più importanti, sempre stando alla prospettiva dello studente, deve essere annoverata la capacità dello stesso di visualizzare una lista di appelli aggiornata e priva di incongruenze (ad es. appelli che hanno date di svolgimento già trascorse) nonché di prenotarsi senza difficoltà alcuna per una o più prove di esame.

Visualizzare gli appelli disponibili, per lo studente, deve voler dire che questi può accedere alla lista di tutti gli appelli aperti fino a quel momento di tutte le materie a patto che le prenotazioni non siano scadute e la lista deve aggiornarsi automaticamente in caso di cambiamenti (es. introduzione di un nuovo appello). La rappresentazione di ciascun appello deve mettere in evidenza la data e l’ora dell’appello, la durata e la data di scadenza della prenotazione oltre alla materia ed al codice dell’appello. Lo studente, specificando il codice dell’appello, deve potersi prenotare allo stesso e deve poter visualizzare la prenotazione avvenuta oltre ad essere notificato in caso di errore nell’inserimento dei dati per la prenotazione.

Alla prenotazione deve seguire, in accordo con la data e l’ora effettiva di un appello, lo svolgimento della prova qualora lo studente sia deciso ad affrontarla. Lo svolgimento della prova deve essere semplice e al contempo deve offrire ad un utente il controllo completo sulla stessa (sottomissione e ritiro in qualsiasi momento, passaggio alla domanda successiva).

Per quanto riguarda, quindi, lo svolgimento vero e proprio della prova di esame, l’esaminando deve poter inserire le risposte in accordo con le modalità di somministrazione della prova stabilite preventivamente visualizzando le singole richieste (domande) e le risposte possibili che possono essere fornite (la prova è a risposta multipla) sulla stessa schermata che dovrà differenziare chiaramente e nella maniera il meno ambigua possibile i contenuti di domande e risposte (requisito di usabilità). Al termine della prova di esame (che può concludersi per molteplici motivazioni esistendo diversi scenari) avviene la valutazione della prova. Alla valutazione, segue la comunicazione dell’esito della prova (punteggio totalizzato) tramite un messaggio che contiene anche l’elenco delle risposte corrette a **tutte** le domande proposte in sede di esame. Giunti a questo punto, l’utente deve essere in grado di scegliere se accettare il giudizio ricevuto (nel caso di superamento dell’esame si intende) oppure rifiutarlo. Compiuto questo passo, può scegliere se disconnettersi o procedere a sostenere un altro degli esami a cui è prenotato e che hanno luogo in quella data oppure può visualizzare nuovamente la lista degli esami al fine di prenotarsi per lo svolgimento di altre prove.

Passando al dominio degli stakeholder assunti dall’università e preposti alla gestione del sistema lato laboratorio, un requisito funzionale fondamentale del sistema è la capacità di provvedere all’identificazione di uno studente: il riconoscimento deve avvenire tramite documento di identità valido e deve essere automatico, senza richiedere l’intervento dell’addetto per riconoscere il singolo studente che tenta di eseguire login. Lo studente che si logga deve, poi, includere nelle informazioni di login anche il proprio numero di matricola per verificare che sia immatricolato. Infine, il login si conclude solo previo deposito dei dispositivi elettronici da parte dell’utente.

Il laboratorio deve gestire le prenotazioni avendo cura di negare ad uno studente che intende prenotarsi ad un appello di un esame che ha già superato ed accettato (voto); lo studente, inoltre, non deve poter intraprendere un esame che non ha sostenuto/superato senza prenotazione.

Il sistema deve permettere all’addetto al laboratorio di modificare la lista degli appelli disponibili per inserire un nuovo appello in qualsiasi momento, con tanto di notifica agli utenti collegati del cambiamento avvenuto perché questi possano visualizzare informazioni in linea con lo stato del sistema. In fase di creazione di un appello il docente deve sottomettere una lista di dieci domande, di cui la prova si compone, avendo cura di specificare un numero di risposte possibili per ciascun quesito strettamente maggiore di uno (risposta multipla) ed un tempo massimo per rispondere alla singola domanda. Sottomesse le domande e scelte una data ed un’ora di svolgimento della prova oltre alla durata della stessa, può essere aperto effettivamente l’appello affinché gli studenti eventualmente interessati si possano prenotare.

Parlando dello svolgimento della prova, le domande vengono proposte una per volta all’utente, il quale, normalmente (ma questa cosa deve essere modificabile) dispone di 5 minuti per rispondere al singolo quesito; al termine dei 5 minuti o in caso di sottomissione della risposta da parte dell’esaminando prima del timeout o ancora passaggio alla domanda successiva senza rispondere alla corrente, viene inviato al laboratorio un messaggio (che non necessariamente contiene una risposta) e lo stesso sistema provvede ad aggiornare il punteggio dello studente in funzione dell’evento che innesca il passaggio ad un’eventuale domanda successiva (+3 per una risposta corretta, -1 per una risposta sbagliata e 0 per una risposta non data) che sarà restituito alla fine della prova allo stesso.

Riguardo alle circostanze di conclusione della prova, questa può terminare perché lo studente si è spontaneamente ritirato o perché ha concluso ed ha regolarmente sottomesso o per fine del tempo concesso; nel primo caso deve essere notificato all’utente che il ritiro dalla prova è andato a buon fine. Nel secondo caso e nel terzo caso bisogna provvedere a valutare il punteggio totalizzato dall’utente; se questi ha raggiunto un punteggio maggiore del minimo per superare l’esame (stabilito in sede di creazione dell’appello) bisogna gestire l’eventuale procedura di accettazione del voto a dovere al fine di registrare eventualmente il voto all’esame della materia in questione associandolo allo studente che ha sostenuto la prova. Caso mai non avesse raggiunto il punteggio minimo, è necessario notificare la bocciatura con presentazione della valutazione.

In tutti i casi menzionati pocanzi lo studente deve ricevere un modulo contenente le risposte corrette a tutte le domande di cui l’appello si compone. La procedura pocanzi descritta non deve richiedere l’intervento di un addetto bensì deve essere automatica.

Gli esami e le prenotazioni, insieme con le attività di gestione del laboratorio devono potersi svolgere **in parallelo**.

***A.1 Servizi (con prioritizzazione)***

*Descrivete in* ***dettaglio*** *i servizi offerti dal vostro Sistema, insieme a quelli che ritenete siano le soluzioni concettuali necessarie. In questa fase, non fate riferimento ad alcuna tecnologia specifica. Se volete, intervistate stakeholder e collezionate dati dal web o da altre sorgenti. Dovete acquisire una conoscenza avanzata dei problemi associate ai vostri servizi. Assegnate un ID a ciascun servizio. Prioritizzate inoltre I servizi in base a due scale: importanza alta, media, bassa. Complessità alta, media, bassa.*

Di seguito saranno elencati i servizi che il sistema deve esporre avendo cura di esprimere l’ID di ciascun servizio al lato dello stesso. Una possibile soluzione è indicata di seguito alla descrizione del singolo servizio. I tre livelli di complessità (**COMP**) saranno indicati con le iniziali degli stessi tra parentesi riportate di seguito alle singole soluzioni concettuali. In quanto a priorità (**IMP**), le stesse sono segnalate al lato. I valori sono alto (A), medio(M), basso(B).

1. PROCEDURA DI LOGIN AL SISTEMA LABORATORIO LATO UTENTE/STUDENTE CON RICONOSCIMENTO DELL’UTENTE BASATO SU DOCUMENTO DI IDENTITA’ VALIDO ED OBBLIGO DI DEPOSITARE I DISPOSITIVI PERSONALI. Per risolvere un simile problema si passa prima di tutto per la consolle dello studente, che chiede a quest’ultimo i propri dati. Fatta salva la validità dei dati immessi in prima battuta, solo dopo aver validato anche il documento (numero identificativo e date di scadenza) e dopo aver imposto all’utente il deposito dei dispositivi da lui posseduti questo può essere autorizzato ad accedere al laboratorio; il tutto deve essere curato automaticamente dal sistema senza l’intervento degli amministratori dello stesso. Se chi prova a loggarsi non è riconosciuto dal sistema in uno qualunque dei due passi di validazione, costui viene tassativamente respinto. Sia l’accettazione che il respingimento devono essere notificati all’utente **(IMP A, COMP M)**
2. CONCEDERE LA POSSIBILITA’ DI APPORTARE MODIFICHE(INSERIMENTI) IN QUALSIASI MOMENTO ALLE LISTE DEGLI APPELLI DISPONIBILI AGLI ADDETTI DELL’UNIVERSITA’. Dal lato del laboratorio, la piattaforma deve esporre una user interface che fornisce all’addetto l’abilità di manipolare gli appelli disponibili per aggiungerne di nuovi. Una soluzione concettuale valida è l’impiego di un componente che oltre a rendere possibili queste operazioni aggiorni gli utenti connessi perché questi visualizzino sempre gli appelli aggiornati. **(COMP B, IMP A)**.
3. VISUALIZZAZIONE DA PARTE DEGLI STUDENTI DELLA LISTA DEGLI APPELLI A CUI CI SI PUO’ PRENOTARE AGGIORNATA AUTOMATICAMENTE. Tale servizio è caratterizzato dal fatto che gli appelli prenotabili hanno una scadenza(una data oltre la quale nessuno può più prenotarsi) e dal fatto che nuovi appelli potrebbero essere inseriti in ogni momento. La soluzione ad un simile problema richiede una sincronizzazione tra laboratorio e singoli utenti, che devono poter vedere gli appelli disponibili in base alla data nella quale si loggano al sistema. Il requisito per accedere a tale servizio è l’avvenuto login nel sistema da parte dello studente. **(IMP A, COMP M)**.
4. PIATTAFORMA PER LO SVOLGIMENTO DEGLI APPELLI. Lo studente deve poter svolgere, in accordo con le modalità di esame e con le date dello stesso gli appelli **previo login al sistema del laboratorio(assodato se è giunto fin qui), immissione in sede di esame dei propri dati (numero di matricola e codice fiscale), accettazione di questi ultimi dal sistema e avvenuta prenotazione all’appello(il tutto da verificarsi tramite una opportuna procedura di sistema)** il giorno dell’appello ed esattamente o comunque poco prima dell’ora dello stesso. Inoltre, deve potersi ritirare dalla prova in qualsiasi momento compreso tra l’inizio e lo scadere del tempo della stessa. Per quanto riguarda lo svolgimento vero e proprio della prova di esame, l’esaminando deve poter inserire le risposte in accordo con le modalità di somministrazione della prova stabilite preventivamente visualizzando le singole richieste (domande) nella schermata che si trova davanti insieme con le possibili risposte. Al termine della prova di esame (o nel momento in cui costui, indipendentemente dai quesiti svolti, **esprime chiaramente la volontà consegnare**), deve essere possibile per l’utente sottomettere l’elaborato; all’avvenuta sottomissione l’utente deve ricevere una notifica di conferma contenenti l’**esito** della prova (punteggio totalizzato) ed elenco delle risposte corrette a **tutte** le domande proposte in sede di esame. Giunti a questo punto, l’utente deve essere in grado di scegliere se accettare il giudizio ricevuto (nel caso di superamento dell’esame si intende) oppure rifiutarlo per poi prendere parte ad un altro esame o eseguire il logout. Una soluzione concettuale elegante deve gestire l’evoluzione del processo di svolgimento dell’esame sulla scorta di eventi innescati dall’utente o dai timeout della prova, da gestire in maniera opportuna per evitare inconsistenze deleterie per colui il quale intraprende un appello; per risolvere un simile problema bisogna adottare misure di sincronizzazione stringenti e rigorose. **(IMP A , COMP A)**
5. PRENOTAZIONE AGLI APPELLI DA PARTE DEGLI STUDENTI. Il singolo studente deve sempre poter tenere traccia delle proprie prenotazioni, aggiungerne di nuove (previo login e specifica in sede di prenotazione di numero di matricola e codice fiscale). Questa modifica deve essere propagata al laboratorio e richiede, dunque, una sincronizzazione tra piattaforma lato studente e laboratorio stesso. **(COMP B, IMP A)**
6. VERIFICA DEI DATI IMMESSI DA UN UTENTE PRIMA DI POTER AFFRONTARE UN ESAME. Il sistema, presa in carico la richiesta dell’esaminando deve appurare che i dati siano corretti prima di permettere a questo di prendere parte alla prova. Lo studente (loggato) fornisce la matricola ed il codice fiscale ed il sistema deve controllare i dati per verificare che effettivamente siano allineati con quanto noto al laboratorio. Nel caso in cui la verifica vada in porto, va appurata la prenotazione dell’utente all’appello che questi desidera sostenere, specificato preventivamente mediante il codice identificativo dello stesso; in caso di prenotazione mancante o di mancata accettazione dei dati, il sistema laboratorio provvede a segnalare l’errore all’utente, il quale può ritentare in seguito. Se il tutto va a buon fine viene recuperato l’esame nel sistema laboratorio e lo studente può intraprenderlo compatibilmente con orario e data di svolgimento dello stesso. (**IMP A, COMP B)**

Lo sviluppo del sistema procederà in maniera tale da permettere l’introduzione di ulteriori servizi come, ad esempio, l’aggiunta/rimozione di appelli, studenti e prenotazioni al sistema lato admin e l’integrazione con un supporto per la persistenza (come ad esempio un DBMS).

* ***A.2 Requisiti non Funzionali***
* *EElencare i requisiti non funzionali più’ importanti per il vostro Sistema*

1. CHIAREZZA NELLE SEGNALAZIONI E NOTIFICHE VARIE. È importante garantire l’usabilità del sistema passando per i messaggi che l’utente riceve dal laboratorio; un sistema poco chiaro nel segnalare/notificare eventi mal si concilia con un contesto in cui la chiarezza espositiva pregiudica il corretto adempimento di un task importante come la gestione di una carriera o comunque di uno o più esami universitari. Dal punto di vista dello studente, data la distanza fisica dal docente e/o corpus di sorveglianza della prova di consueto presente in sede è importante disporre di riferimenti chiari e privi di ambiguità quanto più possibile nonché di segnalazioni che permettano di identificare un eventuale problema e risolverlo. Dal punto di vista dell’addetto, è fondamentale che questi sia messo in condizione di amministrare un sistema complesso quale quello universitario.
2. SEMPLICITA’ DELL’INTERFACCIA DI PRENOTAZIONE DEGLI ESAMI. L’interfaccia deve essere tendenzialmente scarna ma funzionale; naturalmente, le date di stop alle prenotazioni e svolgimento dell’esame devono essere inequivocabilmente esposte poiché lo studente non deve “fare fatica” a comprendere le dinamiche dietro la gestione di un appello al fine di programmare con efficacia gli appelli che è intenzionato a sostenere per ciascuna sessione di esame.
3. CHIAREZZA IN SEDE DI ESAME. L’utente deve poter distinguere chiaramente cosa ha a che fare con le richieste (domande) e cosa ha a che fare con le risposte, quindi l’interfaccia per sostenere la prova deve essere estremamente intuitiva. Inoltre, i comandi per sottomettere la prova e ritirarsi devono essere esplicitamente esposti ed opportunamente segnalati all’utente. Anche la modalità di immissione della risposta deve essere chiara, come del resto la vista post esame in cui sono esposte le risposte a tutti i quesiti.
4. TEMPORIZZAZIONE. Siccome alcune interazioni con il sistema sono caratterizzate dall’intervento di agenti umani è necessario adottare delle misure per non permettere ad alcune tipologie di operazioni (ad esempio quelle bloccanti di IO) di bloccare l’esecuzione dell’applicativo.
5. 5) EFFICIENZA. Le prestazioni del sistema devono quanto meno garantire un utilizzo fluido dello stesso, senza impuntamenti eccessivi.

* ***A.3 Scenari d’uso dettagliati***
* *DDescrivere gli scenari più comuni, più interessanti, o più complicati d’uso dei vostri servizi.*

Si descriveranno gli scenari di interesse intercalandoli nel contesto dei casi d’suo di cui sono istanze e presentando gli scenari alternativi associati in concomitanza.

**LOGIN DELLO STUDENTE NEL SISTEMA LABORATORIO**

|  |  |
| --- | --- |
| Titolo | Login dell’utente |
| Evento trigger | L’utente ha richiesto di eseguire una procedura di login al sistema. |
| Post condizione (garanzia) | L’utente può accedere agli ulteriori servizi del sistema (scenario di successo) oppure viene respinto perché non registrato o non identificabile mediante il documento sottomesso (scenari alternativi). |
| Attori | Lo studente/utente che intende eseguire il login (attore unico e primario). |
| Obiettivo | Eseguire il login al sistema per intraprendere le azioni di gestione della propria carriera universitaria. |
| Tipo | Primario (sea-level), sollecitato direttamente dall’utente. |
| Descrizione/corpo  (scenario principale) | 1. Lo studente esprime la propria volontà di effettuare il login. 2. Lo studente immette nel sistema i seguenti dati:   **matricola, ID e data di scadenza di un documento di identità**   1. *Si verifica se i dati immessi corrispondono ad uno studente noto al laboratorio universitario e se il documento è valido\**. 2. Lo studente viene identificato correttamente ed è tenuto a depositare i propri oggetti tecnologici (es. smartwatch, tablet, cellulare). 3. Lo studente è abilitato al servizio richiesto. |
| Scenario alternativo  (estensione) | 3alt)Il tentativo di login dell’utente viene rigettato poiché costui non è uno studente noto al laboratorio e la procedura di login si conclude qui con esito fallimentare. |
| Scenario alternativo  (estensione) | 3alt)Il tentativo di login dello studente **noto** viene rigettato poiché il documento di identità immesso nel sistema non è valido e la procedura di login si conclude qui con esito fallimentare. |

**CHECK DEI DATI RELATIVI AL LOGIN (PRIMA FASE) DELL’UTENTE**

|  |  |
| --- | --- |
| Titolo | Check dati studente login \* |
| Evento trigger | L’utente ha immesso i propri dati al fine di loggarsi nel sistema. |
| Post condizione (garanzia) | I dati sono validati (nel senso che appartengono ad un utente noto) oppure l’utente non viene riconosciuto ed il login negato. |
| Attori | Lo studente/utente che intende eseguire il login (attore unico e primario). |
| Obiettivo | Controllare i dati immessi da un utente prima di concedere il login. |
| Tipo | Secondario (fish-level), non sollecitato direttamente dall’utente. |
| Descrizione/corpo  (scenario principale) | 1. Si verifica la validità del documento immesso dall’utente. 2. L’ID del documento è usato per cercare l’utente nel sistema. 3. La presenza dell’utente nel sistema è appurata. 4. All’utente viene fornita l’autorizzazione per procedere alla fase successiva del login (deposito dispositivi). |
| Scenario alternativo  (estensione) | 2alt) I dati utente sono rigettati perché non presenti nel sistema. Lo scenario termina qui con esito fallimentare ossia negazione dell’autorizzazione a procedere al login. |
| Scenario alternativo  (estensione) | 1alt) Il documento di identità non è valido. Lo scenario termina qui con esito fallimentare ossia negazione dell’autorizzazione a procedere al login. |

**PRENOTAZIONE DI UN ESAME**

|  |  |
| --- | --- |
| Titolo | Prenotazione di un esame da parte di uno studente |
| Precondizione | Lo studente è loggato nel sistema e l’esame a cui si vuole prenotare è disponibile nella bacheca degli appelli (non sono scaduti i termini di prenotazione) |
| Evento trigger | Lo studente ha richiesto di prenotarsi ad un esame |
| Post condizione (garanzia) | Lo studente è prenotato per l’appello d’esame. |
| Attori | Lo studente/utente che intende eseguire la prenotazione(attore unico e primario). |
| Obiettivo | Prenotare lo studente all’esame richiesto dallo stesso. |
| Tipo | Primario (sea-level), sollecitato direttamente dall’utente. |
| Descrizione/corpo  (scenario principale) | 1. Lo studente sceglie un esame tra quelli prenotabili nella propria bacheca. 2. Lo studente richiede la prenotazione tramite l’immissione del proprio codice fiscale e del proprio numero di matricola. 3. Il sistema notifica all’utente l’avvenuta prenotazione e da questo momento in poi la prenotazione diviene visibile allo studente. |
| Scenario alternativo  (estensione) | 2alt) I dati immessi dall’utente non sono validi, pertanto la prenotazione è rigettata. |

**SVOLGIMENTO DI UN ESAME**

|  |  |
| --- | --- |
| Titolo | Svolgimento di un esame. |
| Evento trigger | Lo studente richiede di sottoporsi ad un appello d’esame. |
| Precondizione | Lo studente deve essere loggato nel sistema e prenotato all’esame; inoltre, l’appello deve essere aperto (la data deve essere quella stabilita a priori per lo svolgimento dell’esame). |
| Post condizione (garanzia) | Lo studente si è sottoposto ad un appello: potrebbe aver superato l’esame, essere stato bocciato, aver avuto problemi o essersi ritirato. |
| Attori | Lo studente/utente che intende sottoporsi all’appello (attore unico e primario). |
| Obiettivo | Intraprendere un appello d’esame. |
| Tipo | Primario (sea-level), sollecitato direttamente dall’utente. |
| Descrizione/corpo  (scenario principale) | 1. Lo studente richiede di iniziare l’appello (*extension* notYetTheTime). 2. L’utente inizia a svolgere la prova. 3. Lo svolgimento termina quando lo studente segnala di voler sottomettere il proprio elaborato. 4. Lo studente riceve una valutazione in merito all’esame svolto e la prova termina. |
| Estensione dello scenario principale  [notYetTheTime] | 1.1) Se non è ancora l’ora prestabilita per l’esame lo studente attende in una waiting room virtuale fin quando il sistema non autorizza l’inizio dell’appello. Quando ciò avviene si passa al passo 2) dello scenario principale. |
| Scenario alternativo  (estensione) | 3alt)Lo svolgimento viene interrotto perché è scaduto il tempo totale della prova. Si passa poi al punto 4) dello scenario principale. |
| Scenario alternativo  (estensione) | 3alt)Lo svolgimento viene interrotto dallo studente che si ritira dalla prova.  4alt)Viene notificato il ritiro con successo all’utente e termina qui l’esame. |

* ***A.4 Excluded Requirements***
* *DDescrivere i servizi eventualmente i esclusi, e spiegare il perchè*

1. **Blocco dell’esame quando la schermata principale perde il focus (viene messa in background).**

L’implementazione di tale servizio è stata esclusa in quanto richiede interazioni potenzialmente complesse con il sistema operativo ma è comunque una delle più importanti funzionalità dei software per gli esami telematici.

1. **Check dei documenti (possibile punto di estensione futura)**

Per verificare la validità di eventuali documenti si deve ricorrere a servizi ad hoc. In questa versione del sistema, per semplicità l’unico check blando è quello sulla data di scadenza (del solo documento dello studente), ma ciò non toglie che in futuro questa funzionalità possa essere estesa a dovere (per esempio con l’introduzione di un’interazione diretta con un addetto preposto a verificare personalmente ciascun documento).

1. **Modifica della data di un appello (possibile punto di estensione futura)**

Per quanto possa essere improbabile che un appello subisca modifiche di data, l’evento è possibile. In questa sede questa eventualità non viene gestita per semplicità in quanto le interazioni tra gli oggetti del sistema sarebbero state potenzialmente complesse.

1. **Sistema di chat tra lo studente ed il docente**

Il sistema di chat tra utente e docente è una “costante” nell’ambito dei sistemi preesistenti per gli esami online, ma implementarne uno in questa sede avrebbe implicato la necessità di progettarlo a dovere e la pratica sarebbe stata comunque abbastanza ardua.

1. **Utilizzo di un DBMS (possibile punto di estensione futura)**

Il sistema non si interfaccia con una base di dati, quindi non è garantita la persistenza. La scelta è principalmente dovuta al fatto che il processo di progettazione è stato orientato alla realizzazione delle funzionalità più importanti, tralasciando dettagli come questo. Naturalmente, il modulo preposto a gestire i dati (che per il momento contiene una sola interfaccia) è facilmente estendibile per implementare tutte le funzioni di cui il sistema ha bisogno.

1. **Annullamento appello (possibile punto di estensione futura)**

Non è possibile annullare gli appelli disponibili; questa funzionalità non è stata inclusa nel progetto per via del fatto che non è molto frequente nella realtà che un appello venga annullato. Ciò nonostante, il sistema è predisposto per una sua introduzione successiva grazie alle misure di sincronizzazione introdotte a livello dei componenti preposti a gestire gli esami e le sessioni.

1. **Inserimento ed eliminazione di uno studente(possibile punto di estensione futura)**

Non è possibile, nella versione attuale, introdurre ed eliminare studenti ma le interfacce prevedono la possibilità di estendere il sistema con introduzione successiva delle funzionalità di cui sopra.

1. **Sorveglianza dello studente durante l’esame**

Come da progetto, si prevede che lo studente possa svolgere l’esame indipendentemente dall’intervento (anche solo per sorveglianza) di un addetto. Introdurre questa funzionalità avrebbe implicato l’aggiunta di una serie di componenti potenzialmente enorme al progetto.

1. **Sistema di gestione delle propedeuticità**

Non tutti gli esami possono e devono essere prenotabili da tutti gli studenti. Anzi, ogni corso di studi gestisce in maniera opportuna le propedeuticità ed è per questo che tale dettaglio è stato omesso in questo progetto.

***A.5 Assunzioni***

*<Briefly document, in this section, the most relevant requirement assumptions/decisions you had to made during your project>*

1. L’applicativo è stato progettato sulla scorta del paradigma client-server. Il laboratorio è un server che fornisce servizi sotto forma di oggetti in rete mentre il client è la consolle o endpoint dell’utente, mediante il quale questi inoltra delle richieste.
2. I dati relativi agli utenti e agli esami sono resi disponibili a priori nel sistema (ma la console dell’amministratore più “ricca” è punto di estensione futura per la manipolazione degli appelli disponibili) .
3. Un documento è valido se non è scaduto (niente check sul ID del documento, per lo meno nella versione naïve ma si può cambiare questo comportamento personalizzandolo).
4. L’esame si svolge ad una data ora, ma c’è un minuto di tolleranza per chi si collega in ritardo. Oltre il minuto non si può svolgere l’esame recuperato.
5. Non ci si può prenotare ad un appello di un esame già superato.
6. Un ID relativo ad un documento di uno studente NON identifica lo stesso univocamente nel sistema; piuttosto, il laboratorio prevede che la matricola funga da chiave per discriminare uno studente dall’altro (simile ad una chiave primaria in uno schema di base di dati). Diversa è la situazione per un amministratore : l’ID del documento lo identifica nel sistema.

***A.6 Use Case Diagrams***

Diagram

Description automatically generated

C. Architettura Software   
*<IF RELEVANT, Report here both the static and the dynamic view of your system design, in terms of a Component Diagram, and their related Sequence Diagrams >*

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

Diagram, schematic

Description automatically generated

***Diagram

Description automatically generated***Diagram

Description automatically generated***C.2 The dynamic view of the software architecture: Sequence Diagram***

***Diagram

Description automatically generated***

***Diagram

Description automatically generated***

***Diagram

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

*Definite le sorgenti di dati a voi necessarie per realizzare I servizi di cui sopra. Modellate tali dati tramite un ER o similari. Specificate se e quali di tali dati sono gia’ forniti da applicativi esistenti.*

Il sistema non si interfaccia con alcun DBMS ma è predisposto per farlo.

E. Scelte Progettuali (Design Decisions)   
<Document here the **5** most important design decisions you had to take. You can use both a textual or a diagrammatic specification.>

1. **Consolle priva di context-awareness**

La consolle (studente o admin che sia) non conosce i dettagli degli oggetti sessione, ma funge da mero tramite per operare dei cambiamenti di stato su questi ultimi in seno agli eventi innescati da parte degli utenti che si interfacciano con questa stessa (e non, quindi, direttamente con una sessione). Questa scelta, se da un lato favorisce il disaccoppiamento totale tra un oggetto complesso come una Session e la Consolle stessa, dall’altro pregiudica in parte la fruizione dei contenuti nel caso in cui la gerarchia di classi Consolle dovesse essere estesa per supportare una GUI. La gerarchia è estendibile, ma ciò richiederebbe un processamento potenzialmente elaborato degli oggetti String e Iterable<String> che vengono restituiti dall’oggetto Session sulla base delle sollecitazioni dell’utente per poi rappresentare gli stessi su una GUI (ad es. con bottoni, campi testuali ecc.).

1. **Gestione centralizzata**

La gestione dei servizi è realizzata facendo appello ad un tipico modello architetturale client-server, sulla scorta del quale l’applicativo è strutturato. Il server (entità laboratorio) gestisce il login degli utenti ed il dispiegamento di oggetti sessione preposti a gestire le interazioni di questi ultimi con il sistema laboratorio. Le sessioni si interfacciano con il sistema laboratorio passando per la mediazione di un oggetto helper (ExamsHelper) in grado di erogare i servizi connessi agli esami stessi (prenotazioni, visualizzazioni ecc.). Di questo oggetto esiste una sola istanza globale (Singleton) implementata in maniera tale da non avere problemi con processi complessi come la serializzazione di Java, funzionale all’uso di RMI.

1. **Gestione dell’esame da parte dell’oggetto sessione**

L’oggetto Session (nello specifico StudentSession) è preposto a gestire l’esame facendo da tramite tra l’oggetto ExamHandler e la consolle dello studente. La scelta di non introdurre un’ulteriore classe per regolare lo svolgimento di una prova è principalmente orientata alla diminuzione dell’accoppiamento tra i moduli laboratorio ed il modulo client (questo “conosce” la sola interfaccia Session) oltre che al contenimento della proliferazione di oggetti nel sistema software.

1. **Design dell’interfaccia Session come “macchina a stati“**

Una Session è un oggetto dalle dinamiche complesse e dettate dallo stato di quest’ultima. Il concetto di “stato” è in questo caso strettamente legato al concetto di sollecitazione esterna a cui risponde, pertanto una soluzione concettuale ideale per disegnare un oggetto Session è basata sul concetto di macchina a stati. A seguito di una transizione, il nuovo stato detta la risposta del modulo Session all’input dell’utente sulla base dei comandi che esso espone. Ad una sollecitazione corrisponde un’altra transizione e così via. All’oggetto consolle vengono fornite informazioni sullo stato corrente (messaggi) all’ingresso in quest’ultimo ed informazioni riguardanti ogni transizione. Il funzionamento della consolle è articolato di conseguenza e ciò è un vantaggio poiché questa **non** conosce gli stati; se ne possono aggiungere di nuovi o si possono modificare quelli già esistenti senza che la stessa accusi gli effetti del cambiamento (disaccoppiamento dei moduli).

1. **Laboratorio con due endpoint**

La scelta di utilizzare un doppio endpoint è funzionale alla discriminazione dei processi di dispiegamento di una sessione studente e di una sessione admin. Ogni utenza conosce la propria interfaccia e tramite questa può accedere ai servizi che le sono riservati in base al proprio ruolo nel sistema.

F. Progettazione di Basso Livello

**Diagram

Description automatically generated**

Diagram, engineering drawing

Description automatically generatedDiagram

Description automatically generated

* GG. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs) *<Report in this section how the architectural and low level design you produced satisfies the FRs and the NFRs>*

Per quanto riguarda il login dello studente (lato client dell’applicativo), questo, interagendo con la consolle specifica i propri dati che poi vengono trasmessi al laboratorio mediante la chiamata al metodo dell’interfaccia StudentEndpoint (implementata dalla classe Laboratory) *studentLogin(int,LocalDate,int)*. L’oggetto Laboratory, in veste di StudentEndpoint , è un oggetto remoto, dunque la chiamata procede per mezzo del framework middleware RMI. Per eseguire il check del documento di identità un oggetto forgiato dalla classe Laboratory si interfaccia con un componente CredentialsHelper, di cui è fornita un’implementazione “naïve” che è preposta alla semplice verifica della data di scadenza del documento; un documento è identificato da una sequenza di cifre, ma disponendo ogni persona di più documenti (es. patente, carta di identità, tessera sanitaria) non si può sapere a priori quale dei documenti verrà usato dall’utente per essere riconosciuto né si richiede un documento specifico. Pertanto, seppur ulteriori verifiche siano implementabili con un approccio incrementale allo sviluppo del componente che implementa l’interfaccia CredentialsHelper, in questa versione la chiamata a *IDCheck(int, LocalDate)* esegue un mero controllo di scadenza. L’implementazione inclusa dell’interfaccia CredentialsHelper si avvale dell’uso di un tipo enumerato, il che consente di rendere l’istanza (unica) dell’implementazione serializzabile compatibilmente con le procedure RMI. Una simile scelta progettuale, oltre a superare tutti i problemi legati alla serializzabilità, facilita l’implementazione del servizio in accordo con il design pattern Singleton, funzionale alla risoluzione di problemi di ingegneria del software in cui una sola istanza di una data classe deve esistere al più in un sistema.

Tornando alla procedura di login vera e propria, fatta salva la “validità” del documento, si procede a verificare che il numero di matricola inserito in fase di login sia appartenente ad uno studente noto al laboratorio; si chiama il DBManager per effettuare un controllo sulla base di dati preposta a gestire simili dati. Questa procedura è funzionale alla verifica dell’utente e alla trasmissione del numero di matricola all’oggetto StudentSession eventualmente restituito da parte del Laboratory se il login va a buon fine. Perché la procedura abbia esito positivo, però, è necessario l’ulteriore step di deposito dei dispositivi, gestito mediante l’utilizzo di una chiamata ad un metodo bloccante dal lato della consolle (si richiede un input all’utente). In tal modo si “simula” il deposito e si abilita la creazione e restituzione dell’oggetto remoto che implementa l’interfaccia Session particolarizzandola per adattarla alle esigenze di uno studente.

Un oggetto di tipo StudentSession mantiene un riferimento all’unica istanza della classe ConcreteExamsHelper, implementata sempre a partire dal design pattern Singleton in accordo con il contratto sancito dall’interfaccia ExamsHelper. Un altro tipo enumerato risolve dunque il problema di mettere a disposizione di tutte le eventuali istanze attive contemporaneamente di Session (siano esse Student o Admin) l’unica istanza della classe preposta alla gestione degli esami (prenotazioni, recupero degli appelli o di un esame, aggiunta di appelli ed eventuale rimozione se successivamente resa disponibile). Questo componente si interfaccia a sua volta con un componente preposto a gestire i dati del laboratorio il cui funzionamento è conforme all’interfaccia DBManager, in questa sede non implementata (se non per effettuare i test del modulo laboratorio). Certo è che, in presenza di oggetti che si interfacciano con DBMS, ancora una volta bisogna gestire opportunamente il numero delle istanze della classe forgiabili ed il recupero delle stesse, quindi anche qui il pattern Singleton può tornare utile. Comunque sia, spostando ancora una volta l’attenzione sul helper, questo collabora al soddisfacimento dei requisiti funzionali mettendo a disposizione metodi che implementano le azioni di controllo e verifica delle prenotazioni, di recupero degli esami inseriti, di inserimento di nuove prenotazioni ed esami in accordo con le regole del laboratorio (ad esempio, non ci si può prenotare ad appelli di esami sostenuti, non vengono proposti agli utenti esami non prenotabili o già tenutisi ecc.). Questo componente funge da traduttore delle richieste degli utenti (es. lista degli appelli) in query (invocazioni dei metodi di query) al DB sottostante; i risultati delle interrogazioni sono poi opportunamente filtrati e restituiti sotto forma di messaggi agli oggetti Session. Tali messaggi si compongono di aggregati di stringhe (java.lang.String) iterabili (implementano java.util.Iterable<String>) che a sua volta l’oggetto Session converte in messaggi da presentare all’utente attraverso l’interfaccia grafica Consolle. In tal modo sono soddisfatti i requisiti funzionali come la visualizzazione degli appelli, delle prenotazioni e dei comandi che un utente può (sempre grazie tramite interfaccia grafica Consolle) sottomettere alla Session. Ogni aggregato di stringhe ha una struttura standard; essa prevede che una stringa esplicativa nei confronti del contesto sia in prima linea, seguita poi da tutte le altre stringhe che descrivono comandi ed opzioni varie. In tal modo si ottiene una vista chiara dell’esame e degli stati in cui la sessione versa volta per volta, rendendo l’interazione utente più immediata pre fare fronte ai requisiti di usabilità.

L’allineamento tra i dati del laboratorio e tutti gli oggetti Session è realizzato mediante l’impiego di una versione non perfettamente conforme alla struttura studiata nel presente corso del design pattern Observer. La classe astratta Subject deve lasciare il posto ad una interfaccia per un vincolo sancito dal linguaggio utilizzato per produrre il presente elaborato (una enum di Java non può estendere classi bensì solo implementare interfacce). Comunque sia, l’oggetto che concretizza ExamsHelper (che è un’istanza di tipo enumerato, è opportuno rimarcarlo), appena una modifica alla lista degli appelli viene apportata da parte di un admin (a patto che questa funzionalità sia resa disponibile estendendo l’implementazione corrente) aggiorna i propri Observer (gli oggetti Session sia Student che Admin) e questi, in accordo con la logica “pull” prelevano il contenuto della lista degli appelli disponibili al fine di proporlo agli utenti che richiedono di visionare gli appelli stessi; entro tale modalità si consuma l’aggiornamento degli appelli disponibili in maniera automatica e on-demand. Tale misura, se da un lato implica un certo accoppiamento tra Session e Helper, dall’altro può aumentare l’efficienza in quanto, se la lista non subisce cambiamenti per un periodo di tempo abbastanza lungo, nessuna query viene inoltrata al sistema laboratorio per visualizzare gli appelli disponibili.

Per completare la veduta di insieme alle soluzioni preposte al soddisfacimento dei requisiti è opportuno prendere visione dell’oggetto StudentSession, realizzato secondo una logica conforme a quella alla base del design pattern State. Come già detto, il comportamento di un oggetto sessione (sia esso Admin o Student) è sempre caratterizzato dallo stato in cui l’oggetto si trova. La mancanza di context awareness della generica Consolle che si rapporta con la Session e l’implementazione semplice di comportamenti complessi sono possibili grazie al fatto che la sessione muta il proprio modo di rispondere alle chiamate al metodo *submitCommand(String),*le quali hanno l’effetto di innescare sull’oggetto oggetto (remoto) una procedura di validazione dell’input (stringa) e traduzione dello stesso in numeri o caratteri (secondo una certa logica di validazione e parsing). Se il comando non viene riconosciuto (immissione errata da parte dell’utente) la sessione evolve verso uno stato di errore notificando l’utente, altrimenti esegue la richiesta pervenuta dallo stesso. Tutto questo è possibile grazie al fatto che le due azioni che, in sequenza, vengono compiute ad ogni transizione di stato, ossia *callback(String,Session)* ed *entryAction(Session)* esposte dall’interfaccia SessionState<T extends Session>, sono implementate dai vari stati incapsulati da parte degli oggetti utilizzati da StudentSession ed AdminSession. Allo stato corrente è delegata, dunque, la gestione delle callback e la costruzione delle rappresentazioni testuali dello stato stesso, esplicative del contesto (requisito di usabilità). Questo ultimo task vede il processo di costruzione degli oggetti contenenti il testo (Iterable<String>) demandata ad un componente nei dettami del design pattern Builder. Il CommandBuilder, forgiato a partire da una inner class statica esposta dalla classe astratta AbstractSession, usando un tipo enumerato che espone i comandi con relative rappresentazioni testuali, vede lo stato corrente in veste di Director che lo “guida” alla costruzione degli oggetti iterabili in seno al metodo *append(Object)* dell’oggetto, che aggiunge una stringa all’aggregato finale restituito da *build()*. Questa soluzione si adatta all’uso proposto di State in quanto ogni stato costruisce rappresentazioni testuali caratteristiche dello stato stesso e di nessun altro stato e permette di soddisfare i requisiti funzionali di specifica delle operazioni che possono essere eseguite o di eventuali output relativi ad operazioni precedenti (come ad es. l’output del processo di esame contenente le risposte a tutte le domande). Quanto alla modifica dello stato, questa è un riflesso della callback e, cioè, dipende da quello che lo stato corrente mette a disposizione a livello di comandi. Per implementare un meccanismo di callback efficiente il mapping tra codice numerico associato ad un comando ed azione sullo stato corrente relativa si utilizza in ciascuna delle classi concrete derivanti da AbstractSession una struttura dati Map<K,V> dal framework Collection di Java. Grazie alla mappa, implementata tramite una HashMap, variabile statica inizializzata con un blocco static ad hoc, è possibile associare un intero (codice numerico del comando) ad un’istanza di un tipo enumerato che modella i comandi con relativa rappresentazione testuale con semplicità. La complessità del lookup su (hash)mappa è O(1), dunque, dato un codice di comando valido (validato mediante il metodo statico *validateNumericInput(String)* o mediante procedure ad hoc del singolo stato) realizzare una callback è semplice grazie a blocchi switch che usano lo stesso tipo enumerato che modella i comandi. Ad un comando, come già detto, corrisponde una transizione, orchestrata tramite il metodo della classe AbstractSession *transition(SessionState)*; esso si avvale di un meccanismo di sincronizzazione per la gestione della mutua esclusione nell’esecuzione della sezione critica di transizione, il monitor nativo di Java. In determinate interazioni con l’utente, la variabile di stato di una Session (specie di una StudentSession) può, infatti, essere letta o modificata da più Thread che eseguono in parallelo. Per comprendere questo passo e spiegare come concorra a soddisfare i requisiti funzionali del sistema si passerà ora in rassegna la procedura di esame. Ogni (appello di) esame è modellato per mezzo di un’astrazione, la classe Exam. Sostanzialmente un Exam è un aggregato di Question, interfaccia degli oggetti domanda. Ciascuna Question è a risposta multipla, incapsula un tempo massimo per rispondere, usato dall’oggetto preposto alla gestione del singolo appello per scandire temporalmente i vari momenti dello svolgimento di una prova. L’oggetto Exam è, infatti, a sua volta incapsulato da un ExamHandler, la cui interfaccia espone diversi metodi funzionali allo svolgimento dell’appello. Tornando alla StudentSession, questa esegue una procedura di recupero dell’esame che un utente intende affrontare interfacciandosi con l’oggetto (o comunque l’unica istanza) di tipo ExamsHelper; l’helper, con il numero di matricola dell’utente e all’ID dell’esame che questi intende affrontare (trasmessi tramite chiamata al metodo *fetchExam(int,int)* dalla Session), compie delle valutazioni circa la correttezza e la validità dei dati trasmessi ed eventualmente forgia un oggetto ExamHandler a partire dall’esame recuperato per poi restituirlo alla Session. La procedura di recupero prevede che si verifichi che l’appello di esame esiste, che lo studente è prenotato e che è il giorno giusto e l’ora giusta per tentare l’esame. Solo allora, se tutto va per il meglio, un riferimento ad un handler è restituito, altrimenti si esegue la return di null. La sessione, verificato che l’handler non è null, avvia i preliminari della procedura di esame.

Se l’utente ha richiesto di sottoporsi alla prova di esame in anticipo questo viene posto in una waiting room mediante una transizione allo stato WAITING-ROOM. Dopo l’attesa, arrivata l’ora dell’esame, la procedura è gestita mediante lo stato EXAM\_MODE (e qui entra in gioco il controllo serrato di concorrenza). L’ingresso della StudentSession nello stato per la prima volta implica l’inizio della prova di esame; questa richiede (requisito funzionale) che siano presenti dei timer a regolare la risposta alle singole domande e per gestire l’esame, pertanto è qui che vengono implementate le soluzioni per far fronte a queste richieste. Innanzitutto, quando la Session si trova nello stato WAITING\_ROOM o EXAM\_FETCHING, appena arriva il momento di cominciare l’esame, una procedura di preparazione viene invocata tramite la chiamata al metodo *prepareExam()* della classe StudentSession. Esso predispone l’oggetto StudentSession a gestire il timing dell’esame; in poche parole, viene istanziato un Thread che attende la terminazione del timer istanziato dal ExamHandler in seguito all’invocazione del metodo *startExam()*; l’handler, infatti, avvia un processo che attende tanti ms quanti sono quelli che designano la durata dell’esame per poi terminare. A terminazione avvenuta, lato StudentSession, il Thread istanziato in *prepareExam()* modifica lo stato dell’oggetto Session (solo se l’esame non è già terminato) e lo setta ad EXAM\_RESULT\_VIEW. Analogo è il comportamento dei timer sulle domande; l’handler di esame istanzia e avvia Thread timer e alla loro eventuale terminazione (timeout della domanda) la sessione, in “ascolto” mediante un thread istanziato ad ogni iterazione-domanda , esegue una transizione verso lo stesso stato EXAM MODE. Così facendo, vengono gestite le evenienze di fine delle domande (il metodo *isOver()* di ExamHandler restituisce false) e si fa fronte ai vari timeout. La soluzione più elegante al problema dei timer, modellati con l’ausilio dell’astrazione Thread di Java, è quella che fa uso del Executor Framework. Tramite un oggetto ExecutorService l’examHandler ha modo di gestire la sottomissione ed esecuzione dei task timer, oltre che la loro cancellazione e la sincronizzazione tra i vari processi. Per ogni timer, un thread lato sessione ne attende la terminazione, il che è facilmente implementabile con l’ausilio degli oggetti Future<K> appartenenti al framework e forniti dall’ExecutorService ogni qual volta un task viene sottomesso. L’handler e la sessione comunicano (e si sincronizzano) proprio grazie a tali oggetti Future<K>, richiesti dalla sessione ad ogni iterazione del processo di esame ed all’inizio della prova stessa. Fatta questa premessa alquanto prolissa , è, infine, opportuno parlare di transizioni. Le transizioni sono un processo delicato, specie quando più Thread possono leggere e scrivere lo stato di un oggetto. A tal proposito, per fare fronte al problema delle race conditions all’interno della gerarchia di classi che derivano da AbstractSession (questa ultima inclusa) si è fatto uso del monitor nativo di Java per gestire l’accesso in mutua esclusione a sezioni critiche di codice. In particolare, tutto il codice che legge e/o scrive lo stato della Session deve essere considerato sezione critica; in base a tale assunzione, dunque, i blocchi synchronized sono funzionali alla realizzazione di una gestione del controllo di concorrenza necessario a soddisfare i requisiti funzionali senza che si possano presentare errori dovuti al threading(es. transizioni “fantasma”) che generano inconsistenze . Questa soluzione, però, non aiuta le performance dell’applicativo; è, infatti, noto che le misure di accesso in mutua esclusione a sezioni di codice potrebbero ridurre l’efficienza dello stesso, quindi questo requisito non funzionale in questa sede è un po’ messo da parte di fronte alla necessità di scrivere un gestore di esame robusto e che eviti le inconsistenze deleterie. In seno a queste misure, il programma soddisfa i requisiti funzionali di temporizzazione, specie in sede di esame, dove due thread timer per volta sono attivi insieme con un processo dell’utente.

AAppendix. Prototype   
*<Provide a brief report on your prototype, and especially: information on what you have implemented, how the implementation covers the FR and NFR, how the prototypes demonstrates your project correctness with respect to the FR and NFR. You may add some screenshots to describe what required above. Be ready to show your prototype during the oral examination>* 

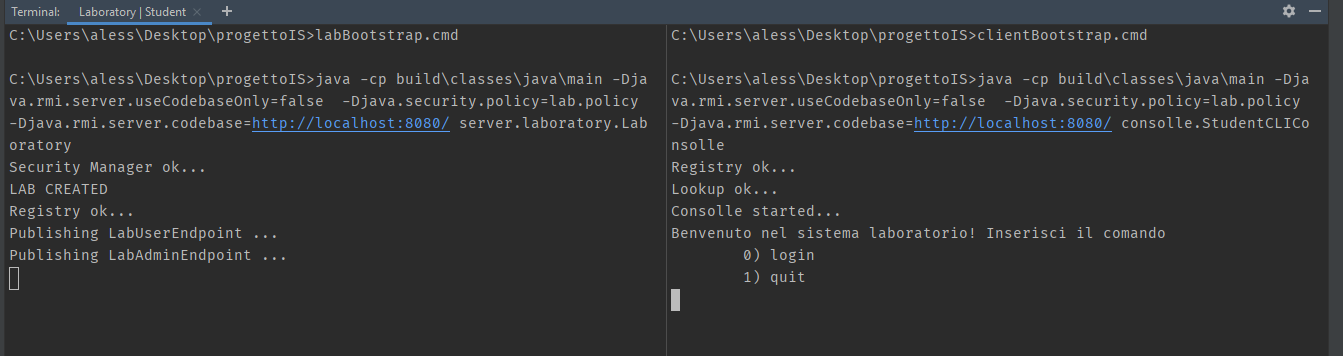


Figura 1. Schermata di login studente (destra) e finestra di avvio del laboratorio.

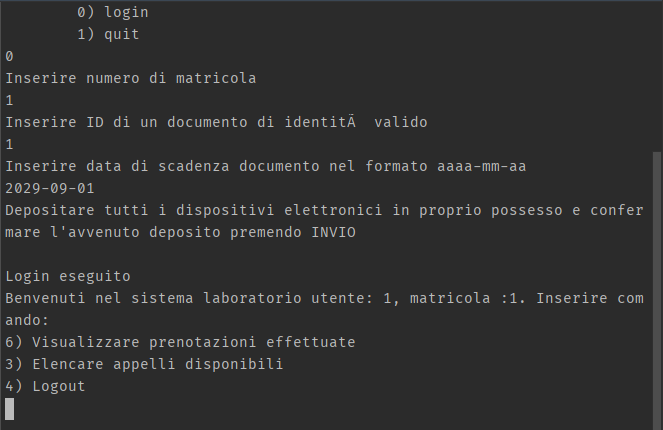


Figura 2. Login dello studente con matricola 1 e ID 1.

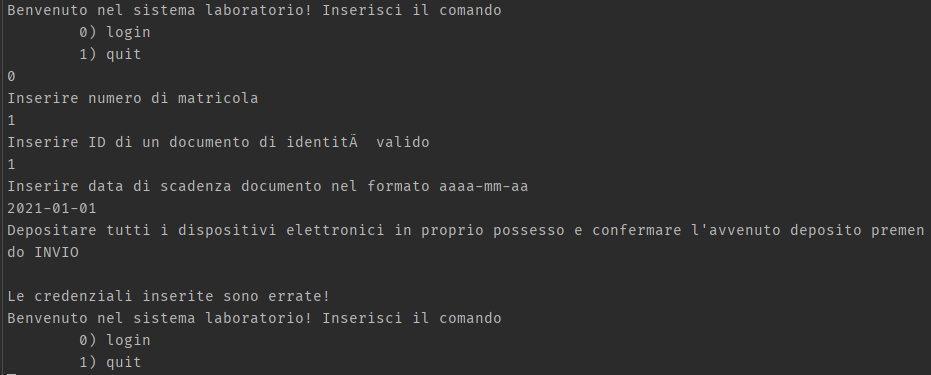


Figura 3. Errore nel login.

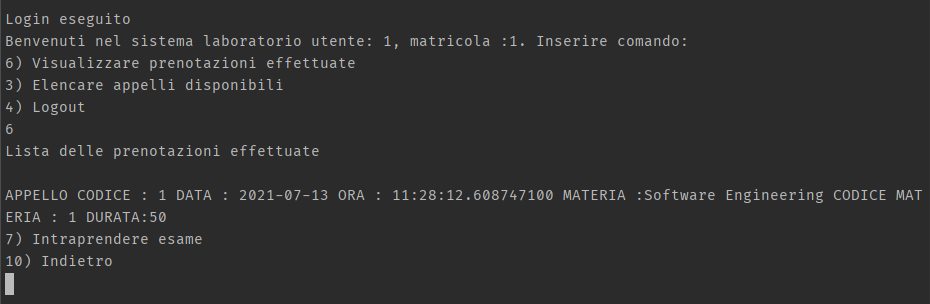


Figura 4. Lista delle prenotazioni effettuate.

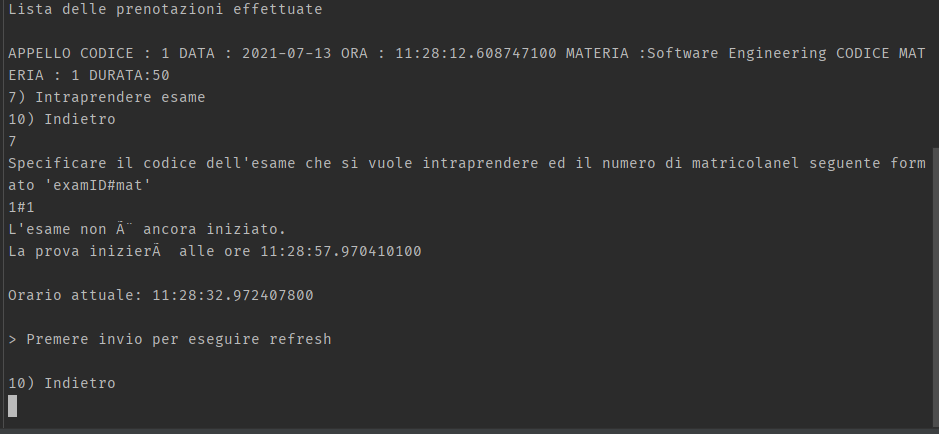


Figura 5. Veduta della waiting room preesame.



Figura 6. Vista d’esame.

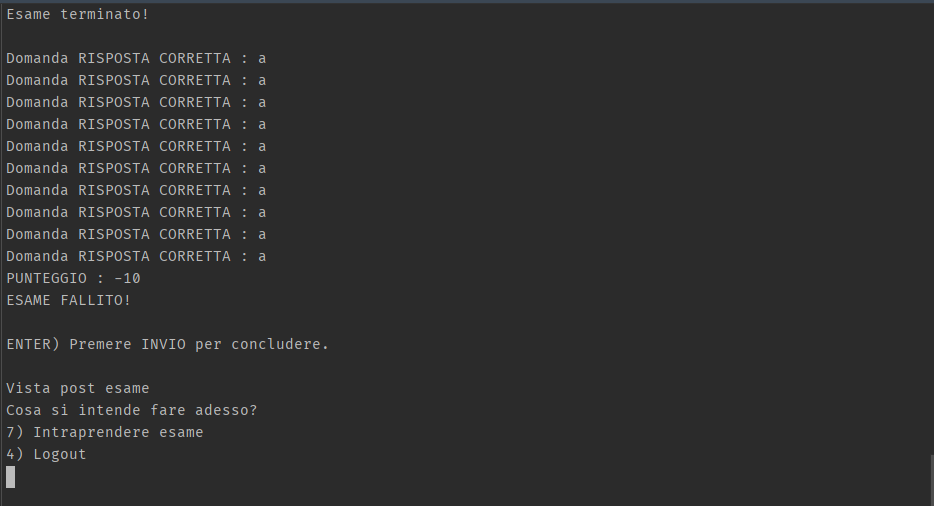


Figura 7. Vista post esame.

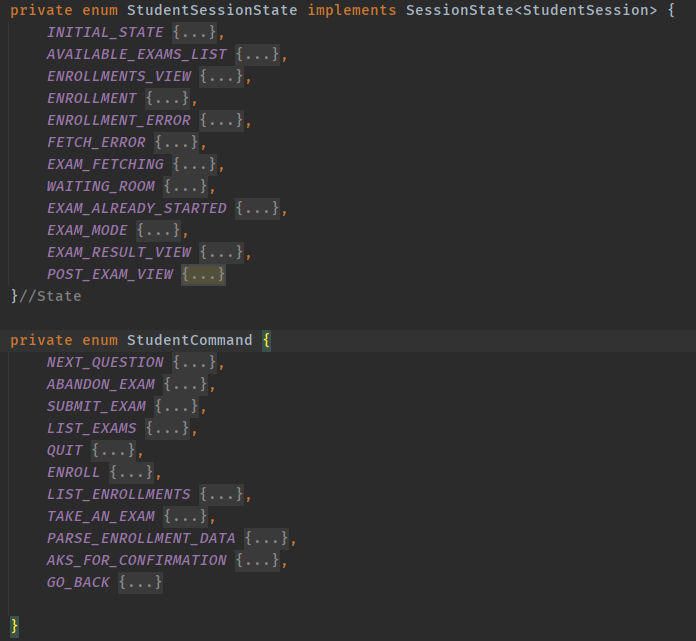


Figura 8. Stati e comandi della StudentSession.