UNIVERSITÀ
DEGLI STUDI
FIRENZE

UniCoach

Un'applicazione per gestire ed organizzare la propria carriera universitaria

Alessandro Bianco Riccardo Becciolini

Giugno 2023

Indice

1 Analisi

1.1 Statement

UniCoach è un'applicazione che permette allo studente di tenere traccia dei propri progressi e della propria carriera universitaria, garantendo una maggiore organizzazione del piano di studi, degli esami conseguiti, e di quelli ancora da sostenere. L'obiettivo è quello di sollecitare l'utente ad avere una maggiore qualità di studio, anche grazie alla possibilità di ricevere notifiche e aggiornamenti sul proprio percorso. Abbiamo deciso di sviluppare questa applicazione al fine di mantenere una corretta programmazione del nostro studio e una migliore visualizzazione della nostra situazione universitaria. UniCoach presenta tre feature principali ed un sistema di login:

- **Gestione e visualizzazione dei voti:** Tramite UniCoach è possibile avere una visualizzazione completa del proprio libretto universitario;
  l'applicazione fornisce allo studente una panoramica dei voti ricevuti per ciascun esame svolto, e la possibilità di conoscere in tempo reale la propria media, i CFU ottenuti e di poter visualizzare tutte le informazioni richieste tramite appositi grafici. L'aggiornamento dei voti viene eseguito in modo automatico appena il docente del relativo corso carica il voto associato allo studente.

- **Gestione degli orari:** L'applicazione propone allo studente, una volta effettuato il login al primo accesso, i vari corsi da seguire tra quelli presenti nella propria facoltà. In seguito UniCoach fornirà all'utente il calendario settimanale completo delle lezioni scelte. Dal punto di vista del professore, quest'ultimo potrà gestire gli orari delle proprie lezioni, visualizzare i voti degli studenti iscritti, e potrà inoltre fornire ai propri studenti notifiche ed aggiornamenti sullo svolgimento del corso tramite mail. In particolare sarà possibile appuntare, al termine di ciascuna lezione, gli argomenti trattati ed eventuali suggerimenti per gli studenti. Infine il professore può decidere le date del proprio esame le quali verranno recapitate a tutti coloro iscritti al corso.

- **Tracker study-time:** Ogni giorno lo studente è invitato a compilare un form specificando le ore di studio spese durante la giornata. Questo permette ad UniCoach di tenere traccia dei progressi dello studente e quindi di monitorare la sua situazione col passare del tempo. Dal punto di vista del professore, quest'ultimo potrà visualizzare, per ogni studente, le ore dedicate alla propria materia, ed eventualmente confrontarle con le ore dedicate allo studio di altri corsi.

- **Sistema di autenticazione:** UniCoach è dotata di un sistema di autenticazione dell'utente:
  - **Studente:** per tenere traccia dei progressi universitari e degli orari dei corsi scelti.
  - **Professore:** per assegnare voti, gestire lo svolgimento delle proprie lezioni e visualizzare l'andamento complessivo degli studenti.

1.2 Use Case Diagram

Di seguito sono presentati i diagrammi dei casi d'uso dell'applicazione realizzati con StarUML. I diagrammi sono suddivisi in tre sezioni, corrispondenti alle tipologie di utenti (Professore e Studente), e includono anche i casi d'uso comuni rappresentati nel diagramma generale dell'Utente. Gli stereotipi *include* ed *extend* sono utilizzati per indicare rispettivamente un'azione atomica che fa parte di un contesto più ampio e un comportamento aggiuntivo in determinate condizioni specifiche.

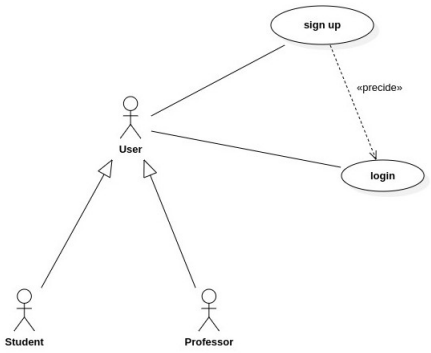

Figura 1: diagramma Use Case generico per gli utenti

Figura 2: diagramma Use Case per lo studente

Figura 3: diagramma Use Case per il professore

1.3 Use Case Template

Di seguito vengono riportati i template degli Use Case per documentare alcuni casi d'uso. È stato deciso di documentare soltanto le feature principali per ciascuna tipologia di utente.

Use Case	Registrazione
Scope	Azione riservata agli utenti
Level	User Goal
Actor	User
Basic course	1 L'utente entra nella pagina di registrazione (Figura 6) 2 Inserisce le proprie credenziali e si identifica come Professore oppure Studente 3 Il sistema verifica la correttezza dei dati inseriti 4 Il nuovo utente è registrato nel database 5 L'utente registrato accede alla sua HomePage
Alternative course	3_a L'utente inserisce le credenziali errate $3a_1$ Viene segnalato l'errore tramite un messaggio $3a_2$ Registrazione fallita e nuovo tentativo

Tabella 1: Use Case registrazione

Use Case	Login
Scope	Azione riservata agli utenti
Level	User Goal
Actor	User
Basic course	1 L'utente entra nella pagina di login (Figura 6) 2 Inserisce le proprie credenziali 3 Preme il pulsante "accedi" 4 Il sistema verifica che le credenziali corrispondano in maniera corretta 5 L'utente viene reindirizzato alla sua homepage
Alternative course	$4a$ Le credenziali non corrispondono $4a_1$ Viene segnalato l'errore tramite un messaggio $4a_2$ Accesso fallito, nuovo tentativo o se l'utente non è ancora registrato viene reindirizzato alla pagina di registrazione

Tabella 2: Use Case login

Use Case	Visualizzazione voti lato studente
Scope	Azione riservata allo studente
Level	Function
Actor	Studente
Basic course	1 Lo studente dalla homepage (Figura 9) sceglie di visualizzare i suoi voti 2 Il sistema reindirizza alla pagina dei voti 3 Lo studente vede la sua media, i CFU convalidati e i voti che ha preso ad ogni esame svolto

Tabella 3: Use Case visualizzazione voti lato studente

Use Case	Visualizzazione voti lato professore
Scope	Azione riservata al professore
Level	Function
Actor	Professore
Basic course	1 Il professore dalla homepage (Figura 10) può scegliere di visualizzare i voti di uno studente 2 Il sistema indirizza il professore alla schermata dedicata alla visualizzazione dei voti (Figura 12) 3 Il professore sceglie uno studente 4 Il sistema mostra al professore la media dello studente scelto e quanto ha preso a ogni esame
Alternative course	1a Il professore dalla homepage può scegliere di visualizzare i voti del suo corso 1a_1 Il sistema reindirizza il professore alla schermata dedicata alla visualizzazione dei voti dei corsi 1a_2 Il sistema mostra al professore i voti che sono stati presi al suo corso per ogni studente 1a_3 Il sistema mostra al professore il confronto della media del proprio corso con la media di tutti gli altri corsi

Tabella 4: Use Case visualizzazione voti lato professore

Use Case	Visualizzazione e inserimento study time studente
Scope	Azione riservata allo studente
Level	Function
Actor	Studente
Basic course	1 Lo studente dalla homepage giunge sulla pagina dedicata allo study time (Figura 13) 2 È possibile visualizzare le informazioni sullo study time del giorno precedente 3 È possibile visualizzare le informazioni sullo study time completo per ogni corso e tipo di studio
Alternative course	$2a$ Lo studente clicca sul tasto "Inserisci study time" $2a_1$ Il sistema reindirizza lo studente alla schermata dedicata all'inserimento delle ore dedicate allo studio del giorno corrente

Tabella 5: Use Case visualizzazione e inserimento study time studente

Use Case	Visualizzazione study time professore
Scope	Azione riservata al professore
Level	Function
Actor	Professore
Basic course	1 Il professore dalla homepage giunge sulla pagina dedicata allo study time completo del suo corso (Figura 14) 2 Il professore visualizza le informazioni relative alla suddivisione dello studio dei suoi studenti nel suo corso 3 Il professore visualizza le informazioni relative alla suddivisione dello studio in tutti gli altri corsi dell'università

Tabella 6: Use Case visualizzazione study time professore

2 Progettazione

2.1 Page navigation diagram

Si analizza lo schema di navigazione tra le diverse sezioni dell'applicazione tramite il diagramma di navigazione delle pagine (page navigation diagram). In particolare, vengono illustrate le azioni principali eseguite dagli utenti, concentrandosi sugli aspetti rilevanti e tralasciando i dettagli sulle azioni secondarie. Sono presentati due diagrammi: uno specifico per le azioni dello studente ed uno dedicato alla versione del professore.

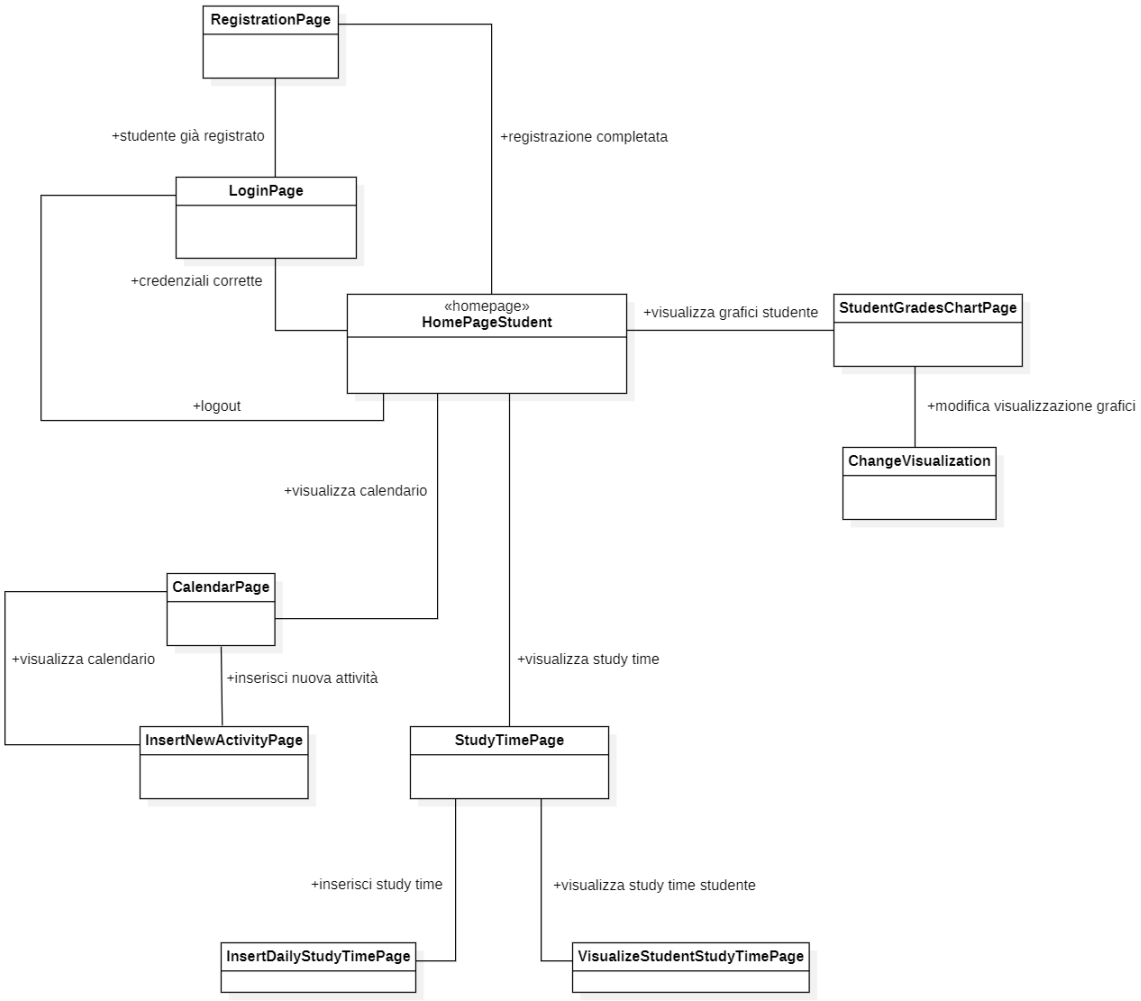

Figura 4: Page navigation studente

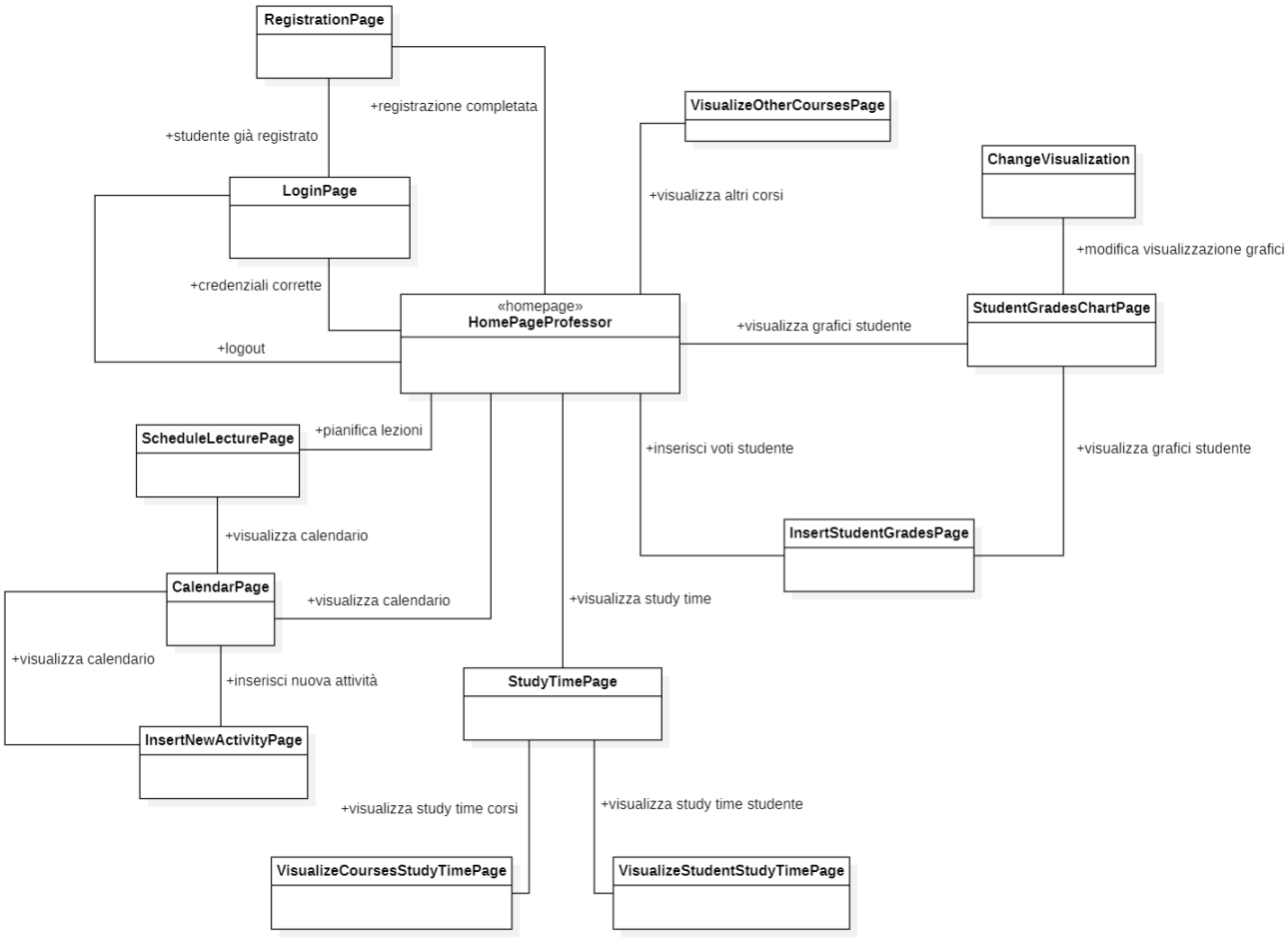

Figura 5: Page navigation professore

2.2 Mockup

Di seguito sono riportati i mockup di una possibile implementazione di UniCoach come app mobile realizzati tramite il sito *Visily.ai*. Vengono evidenziate le varie schermate le quali racchiudono le principali feature, suddivise in base alla tipologia di client.

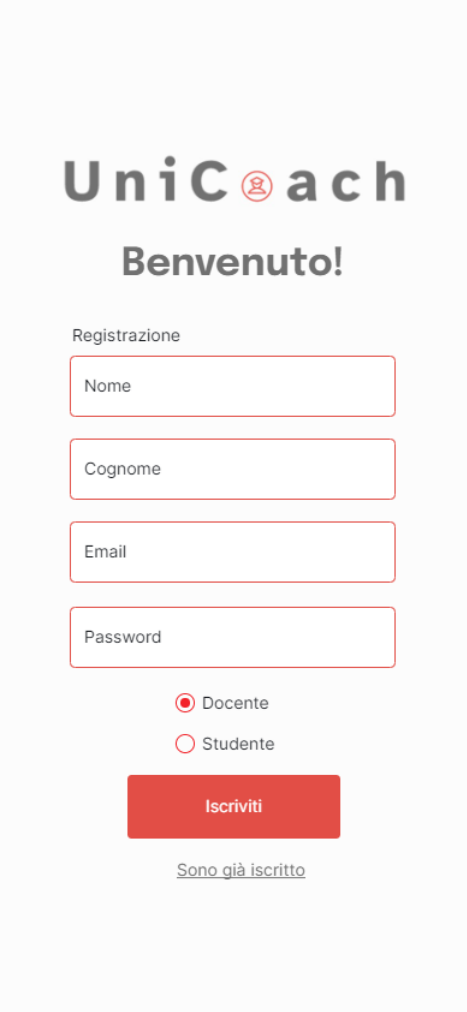

Figura 6: Mockup Registrazione utente

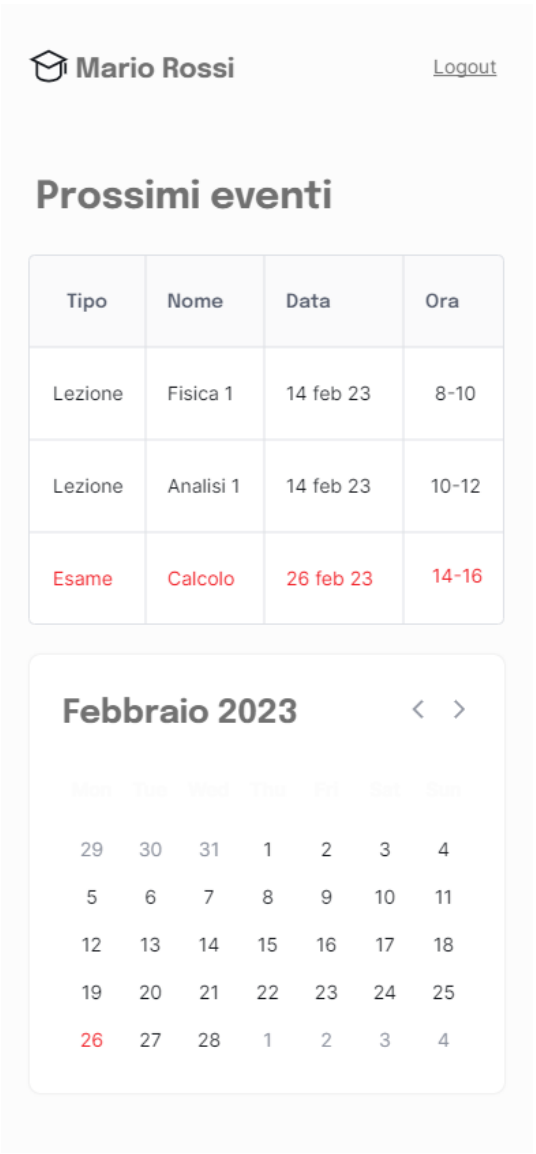

Figura 7: Mockup calendario eventi

Visualizzazione del calendario (identica per tutti gli utenti) dove si mostrano i prossimi impegni ed eventualmente si aggiungono nuove attività

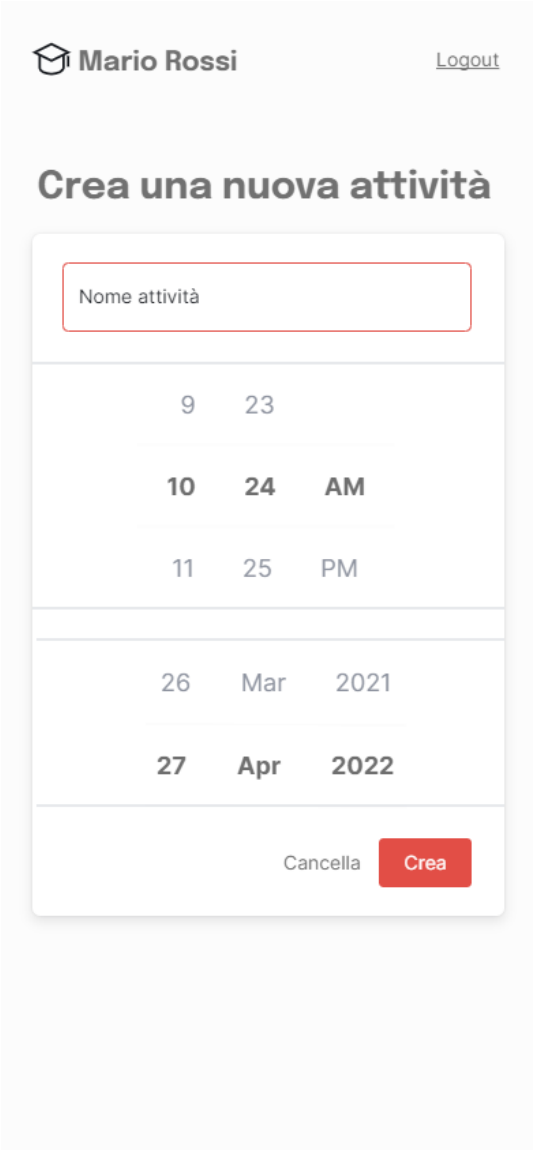

Figura 8: Mockup inserimento nuova attività nel calendario

E' possibile aggiungere una nuova attività nel giorno e ora selezionati

Figura 9: Mockup homepage Studente

In questa pagina lo studente visualizza le informazioni principali sulla sua situazione, il prossimo esame prenotato e il libretto degli esami svolti

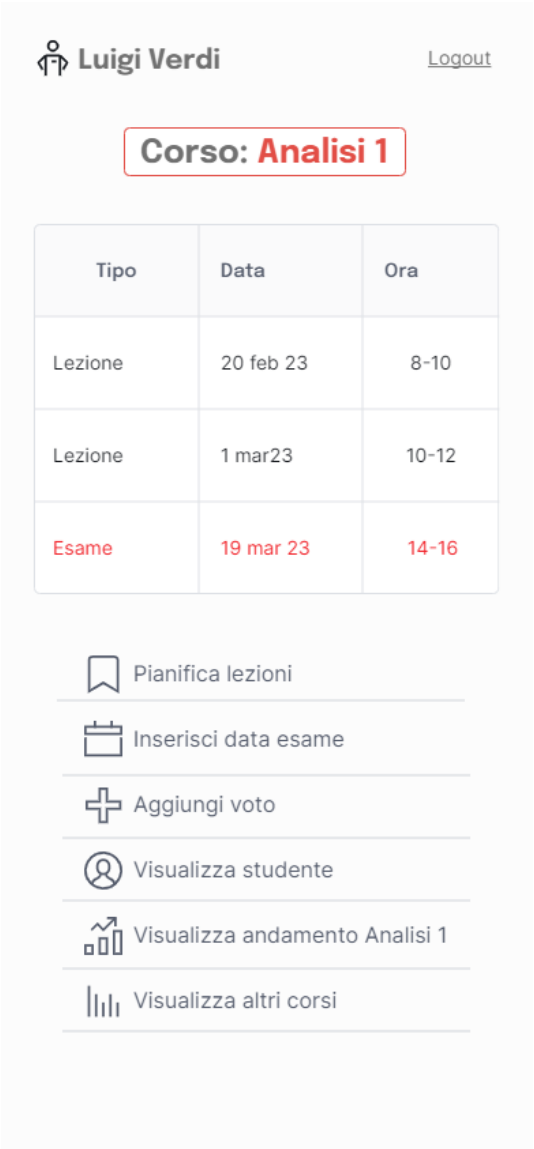

Figura 10: Mockup homepage Docente

In questa pagina il docente visualizza le informazioni riguardanti il suo corso, il prossimo esame prenotato e il libretto degli esami svolti

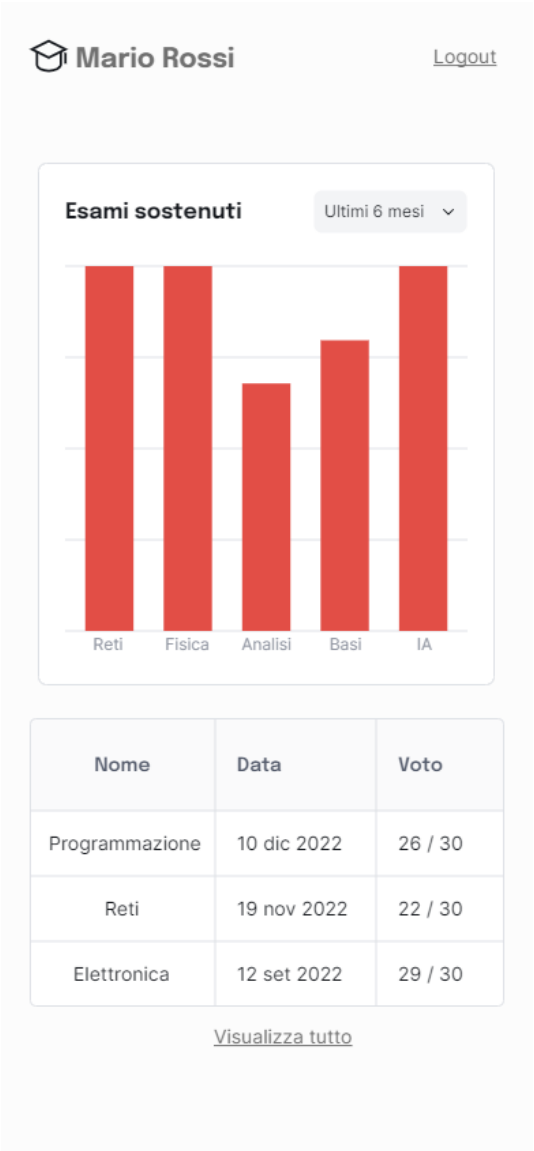

Figura 11: Mockup visualizzazione grafici libretto studente

Da questa schermata è possibile visualizzare (lato studente) il grafico corrispondente ai voti degli esami svolti

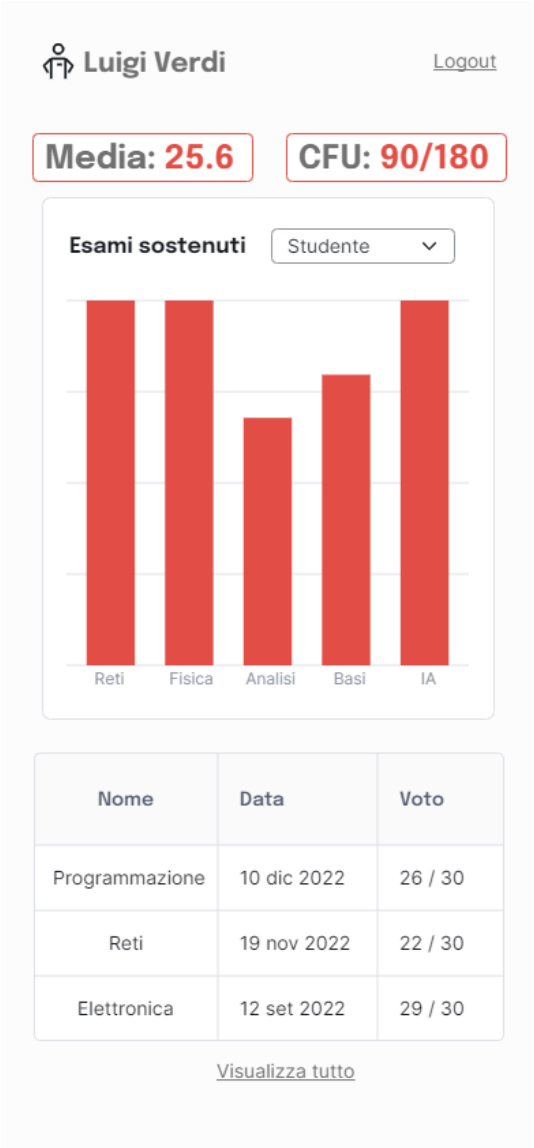

Figura 12: Mockup visualizzazione grafici dello studente iscritto al corso del professore

Da questa schermata è possibile visualizzare (lato docente) il grafico corrispondente ai voti degli esami svolti dello studente selezionato per verificarne l'andamento complessivo

Figura 13: Mockup visualizzazione grafici libretto studente

Visualizzazione dello study time dello studente, con visualizzazione della suddivisione dello studio nel giorno precedente e panoramica sul totale dello studio per ciascun esame suddiviso in tipologie

Figura 14: Mockup visualizzazione grafici dello studente iscritto al corso del professore

Visualizzazione (lato docente) dello study time medio degli studenti iscritti al corso del professore e panoramica dello study time riferito agli altri corsi dell'università

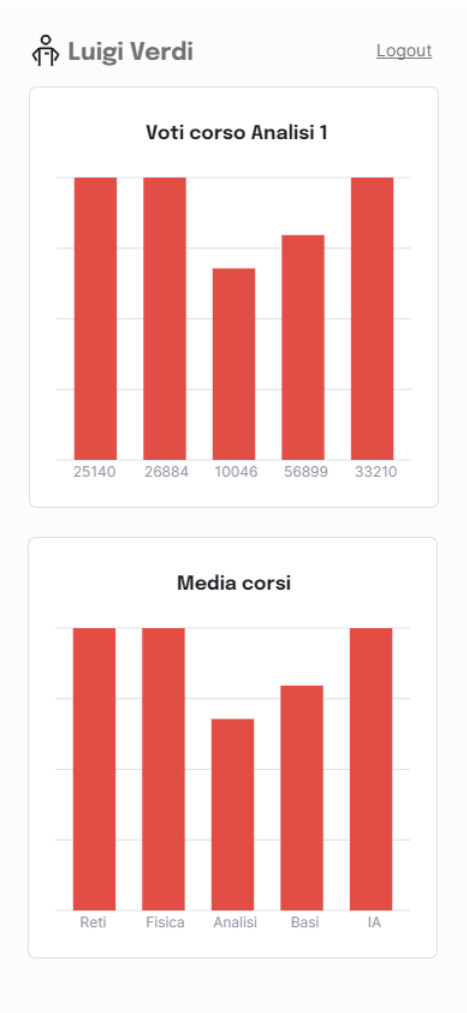

Figura 15: Mockup grafici andamento corsi

Visualizzazione (lato docente) della media dei voti di tutti gli esami svolti dagli studenti iscritti al corso del docente e la media degli esami per ciascun altro corso dell'università

2.3 Class Diagram

Di seguito è rappresentato il class diagram contenente la struttura delle classi utilizzate e i metodi contenuti in esse, per semplicità non sono stati inserititi i metodi getter e setter. Le classi sono racchiuse nei rispettivi package di appartenenza in modo tale da poterne incapsulare le funzionalità specifiche. In seguito nella sezione implementazione ogni package verrà discusso in maniera più approfondita.

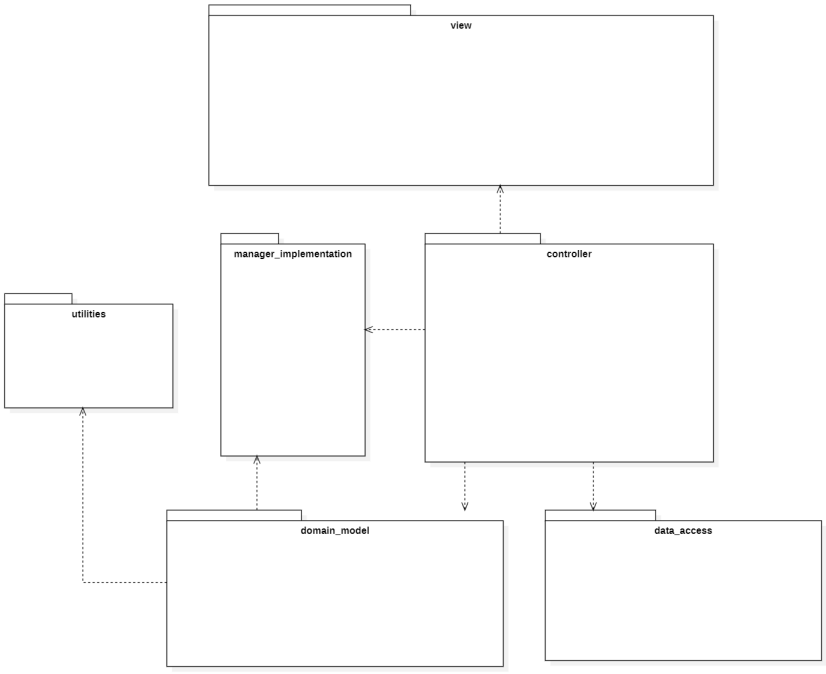

Figura 16: Class Diagram: focus sui package

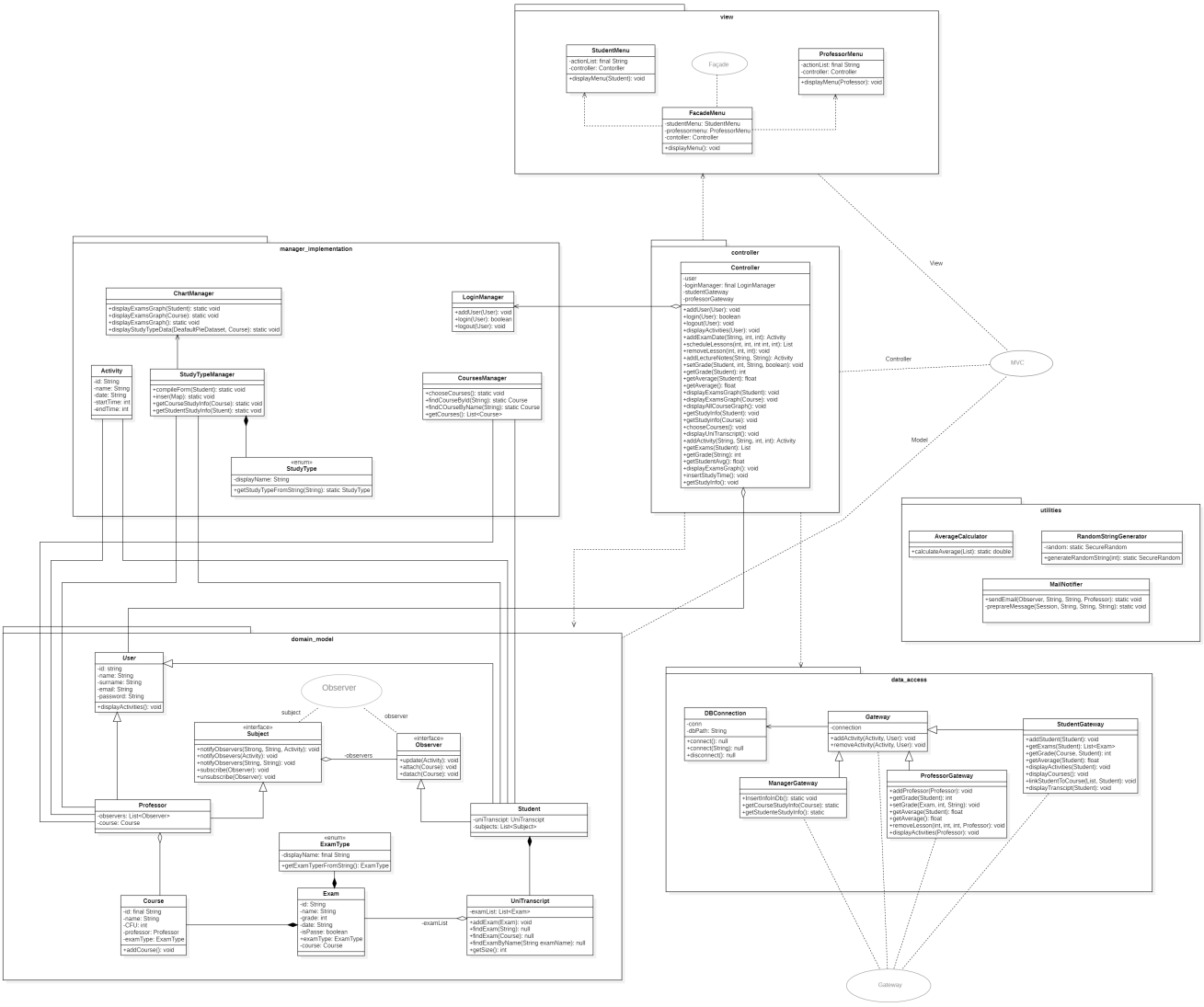

Figura 17: Class Diagram completo

2.4 Database

È stato scelto di implementare un database in modo tale da consentire il salvataggio dei dati dell'applicazione. Di seguito è riportato lo schema Entity Relationship utilizzato durante la progettazione del database:

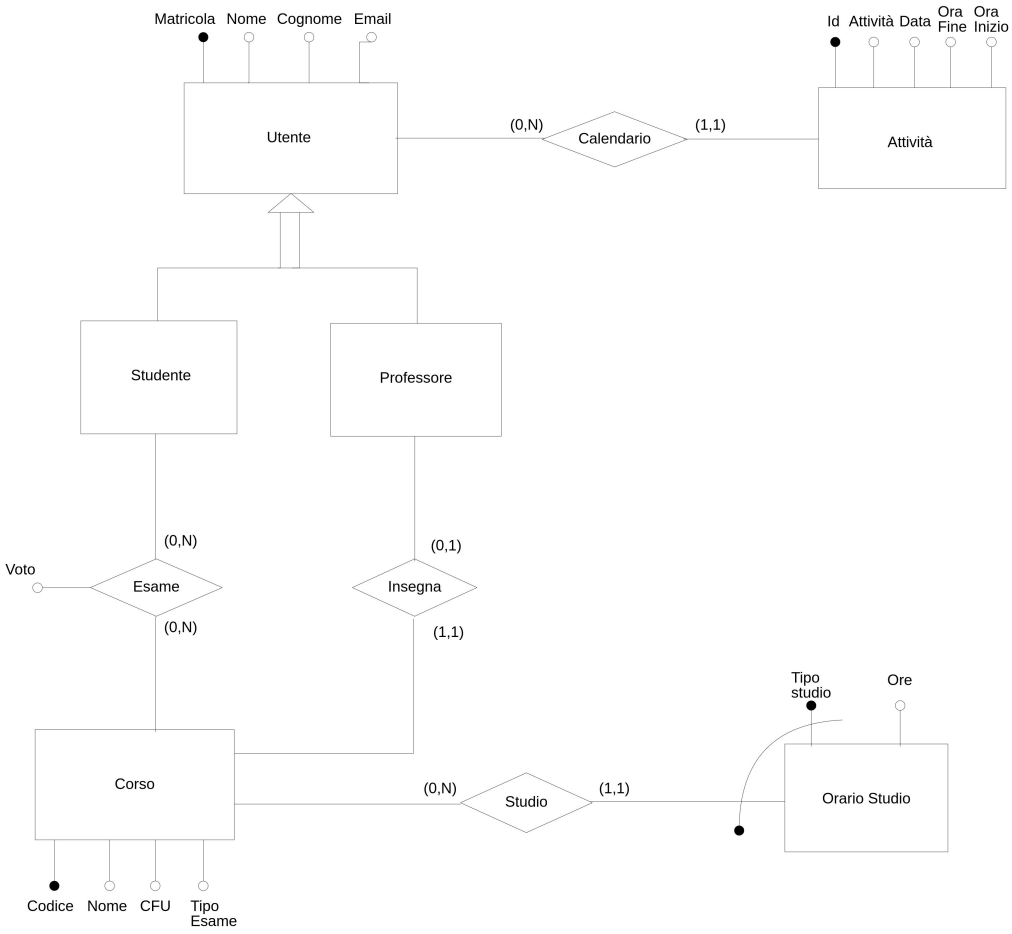

Figura 18: Schema E-R

Da questo schema si può osservare che:

- L'entità Utente è una generalizzazione totale delle rispetto alle entità figlie Studente e Professore, inoltre l'entità Utente ha una relazione con uno-a-molti con l'entità Attività. Questo, durante la progettazione logica del database, viene tradotto in due tabelle calendario, una relativa alla tabella Studente ed una relativa alla tabella Professore.
- La relazione tra Studente e Corso tiene traccia del voto che lo studente ha preso all'esame relativo ad un determinato corso.
- L'entità Orario Studio possiede un identificatore esterno poichè nel database si tiene traccia di quale corso si sta riferendo.

3 Implementazione

3.1 Architettura e librerie esterne

UniCoach è pensata per essere un'applicazione mobile, e necessita quindi di un'interfaccia che relazioni il client con tutte le funzioni proposte; questo viene realizzato tramite il pattern MVC. Per quanto riguarda la permanenza dei dati si utilizza un database accessibile tramite le classi gateway che hanno il compito di interrogare ed interfacciarsi con la base dati. Una delle feature principali di UniCoach è il fatto di poter ricevere notifiche su eventi programmati, ad esempio lezioni e nuove date di esami disponibili; per fare questo si utilizza il framework JavaMail, il quale consente di inviare mail e notificare quindi gli utenti (in particolare gli studenti) dei nuovi eventi disponibili. Infine, per la realizzazione dei grafici visualizzabili sia dal docente che dallo studente, è stato utilizzato il framework JFreeChart.

3.2 Classi

Di seguito si analizza l'implementazione delle varie classi nei package, evidenziandone il ruolo ed il funzionamento

3.2.1 Package: view

Il package *view* contiene le classi dedicate all'interfaccia tra il client e l'applicazione. Sono presenti tre classi:

- StudentMenu
- ProfessorMenu
- FacadeMenu

Le prime due classi racchiudono i menu dedicati agli utenti registrati come Studenti e come Docenti, con le rispettive azioni disponibili, specificate per la tipologia di utente selezionata. La classe *FacadeMenu* implementa il design pattern *facade* in modo tale da presentarsi come unica interfaccia all'utente, la quale andrà a specializzare successivamente il menu specifico in base alla richiesta fornita. Entrambi i menu permettono di navigare nelle varie azioni disponibili tramite prompt da terminale, simulando una possibile interazione su un dispositivo mobile (come mostrato nei mockup).

3.2.2 Package: controller

Questo package contiene la classe *Controller*, la quale ha il compito di gestire le azioni selezionate dal client attraverso le richieste inviate dal corrispettivo menu. Il *controller* si occupa di invocare le funzioni appropriate per ciascuna azione scelta e di garantire quindi il corretto funzionamento dell'applicazione. Questa separazione delle responsabilità e dei compiti tra le classi rende il codice più manutenibile e semplifica l'implementazione di eventuali nuove funzionalità in futuro.

3.2.3 Package: domain_model

Il package *domain_model* rappresenta il *back-end* del software. Le classi contenute in questo package sono:

- User: è una classe base che ha due sottoclassi, *Student* e *Professor*. Rappresenta la generalizzazione di un client all'interno del software e contiene i dati anagrafici pertinenti. Gli attributi *email* e *password* vengono utilizzati per il processo di login.
- Student: rappresenta uno studente che possiede un libretto universitario. Contiene quindi tutte le informazioni appropriate per la modellazione dello studente.

- Professor: analogamente alla classe Student, la classe Professor mantiene traccia delle informazioni relative all'entità del professore.
- Course: rappresenta un corso offerto dalla facoltà, al quale è associato un professore. Uno studente può scegliere di iscriversi a un determinato corso. Tutti gli studenti iscritti a un corso riceveranno aggiornamenti via email relativi al corso stesso.
- Exam: questa classe incapsula le informazioni relative agli esami svolti dagli studenti. L'esame viene inserito nel database al momento della creazione del corso e dell'iscrizione dello studente. Successivamente, il voto viene aggiornato dal professore una volta che lo studente ha sostenuto l'esame.
- UniTranscript: contiene la lista di tutti gli esami di uno studente, sia quelli svolti che quelli ancora da svolgere ma programmati nel suo piano di studi.

3.2.4 Package: data_access

Il package *data_access* contiene le classi responsabili della comunicazione con il database al fine di ottenere le informazioni necessarie per l'esecuzione delle varie operazioni richieste dagli utenti. All'interno di questo package, sono presenti diverse classi specializzate in base al tipo di richiesta effettuata, che può provenire da utenti come Studente o Docente, o da classi di tipo *Manager*. Di seguito sono elencate le classi che compongono il package:

- DBConnection
- Gateway
- StudentGateway
- ProfessorGateway
- ManagerGateway

La classe *DBConnection* fornisce i metodi statici per la connessione e la disconnessione al database, che vengono richiamati dalle altre classi del package. La classe astratta *Gateway* consente di generalizzare la successiva specializzazione delle classi gateway dedicate a ciascun tipo di richiedente. Questa separazione ci permette di gestire in modo più efficace la comunicazione tra l'applicazione e il database, distinguendo il tipo di richiesta effettuata.

3.2.5 Package: manager_implementation

Il package *manager_implementation* è stato progettato per contenere diversi *Manager* che implementano funzioni specifiche di UniCoach. I manager implementati sono i seguenti:

- LoginManager: questa classe gestisce il processo di accesso e registrazione dell'utente all'applicazione. Viene utilizzata dal *Controller* per l'autenticazione del client.
- CoursesManager: si occupa dell'organizzazione dei corsi disponibili ed è responsabile di consentire agli studenti di selezionare i corsi da seguire durante l'anno accademico.
- StudyTimeManager: è dedicato al monitoraggio del tempo di studio degli studenti. Ha il compito di notificare l'utente per l'inserimento delle informazioni sullo studio e fornisce metodi per la visualizzazione dei grafici correlati. Inoltre, possiede un attributo *StudyType* definito come un'enum omonima, che specifica la categoria di studio inserita.
- ChartManager: si occupa della generazione dei vari grafici richiesti nell'applicazione.

  Nel package è presente anche una classe *Activity*, che incapsula le informazioni relative alle diverse attività aggiunte dagli utenti.

3.2.6 Package: utility

Il package *utility* viene creato per raggruppare le funzioni che non fanno parte direttamente della logica principale del programma, ma sono necessarie per soddisfare diversi task generici dell'applicazione.

- RandomStringGenerator: utilizzata per generare stringhe casuali utili per simulare gli identificatori degli elementi dell'applicazione.
- AverageCalculator: classe che calcola la media dei voti forniti in input.
- MailNotifier: classe utilizzata per gestire il servizio di notifica via email implementato nell'applicazione. Questa classe contiene il metodo per inviare un'email al destinatario selezionato con il messaggio specificato. Per questa funzionalità viene utilizzata l'API JavaMail.

3.3 Dettagli implementativi: design patterns utilizzati

Di seguito si descrivono i design pattern utilizzati nello sviluppo del programma.

3.3.1 MVC

Il design pattern architetturale Model-View-Controller (MVC) viene utilizzato per separare le responsabilità dell'applicazione in tre componenti: Modello, Vista e Controller. Nel contesto dell'applicazione in questione, il package *domain_model* rappresenta il Modello, che gestisce i dati e la logica di business. La Vista, anche se è stata implementata solo con richieste da terminale, rappresenta l'interfaccia utente. Il Controller funge da intermediario tra il Modello e la Vista, gestendo gli input dell'utente, coordinando le operazioni sul Modello e aggiornando la Vista di conseguenza.

3.3.2 Observer

Nell'applicazione è stato implementato il design pattern Observer, utilizzato per la comunicazione tra classi. Sono create due classi astratte, *Observer* e *Subject*, con sottoclassi concrete chiamate *Student* e *Professore*. La classe *Professore* mantiene una lista di osservatori che viene aggiornata ogni volta che uno studente si iscrive ad un suo corso. Allo stesso modo, la classe *Studente* mantiene una lista di soggetti ai quali è iscritto; questo consente ad uno studente di iscriversi a più corsi e ricevere aggiornamenti da più professori. Quando un professore deve notificare gli studenti, utilizza il metodo *notify* che, tra le altre azioni, invia un'email agli studenti interessati. In questa implementazione dell'Observer pattern, viene utilizzato il meccanismo di "push", in cui le informazioni sullo stato del soggetto vengono passate come parametro nella funzione di aggiornamento.

3.3.3 Façade

Il design pattern Façade è un pattern strutturale che fornisce un'interfaccia semplificata per un sistema complesso di classi, al fine di nascondere la complessità del programma. Nel progetto in questione, il pattern Façade è stato utilizzato per nascondere le implementazioni dei menu dietro a una singola interfaccia chiamata *FacadeMenu*. In questo modo, i diversi tipi di utenti possono accedere allo stesso menu, ma possono utilizzare solo i metodi a cui hanno accesso.

3.3.4 Gateway

Viene implementato il pattern Gateway, che è un pattern architetturale che fornisce un'interfaccia unificata per l'accesso a sistemi esterni o risorse remote, aumentando l'astrazione e separando le responsabilità. Nel contesto dell'applicazione, il pattern Gateway è stato utilizzato per l'accesso al database. È stata introdotta una classe astratta chiamata *Gateway*, con tre sottoclassi: *ProfessorGateway*, *StudentGateway* e *ManagerGateway*. A seconda dell'oggetto che deve interagire con il database,

viene utilizzata la classe gateway corrispondente, i cui metodi utilizzano l'API JDBC per accedere al database ed eseguire le query necessarie.

4 Test

Questa sezione presenta i test dell'applicazione, realizzati tramite il framework JUnit integrato in IntelliJ. Utilizzando la pratica di Test-Driven Development sono stati testati tutti i metodi per garantire e verificare che le operazioni richieste fossero eseguite correttamente. I test svolti comprendono test funzionali e test di unità, per verificare il corretto funzionamento delle principali funzionalità dell'applicazione e delle singole componenti di codice. I test sono stati organizzati in base alle classi specifiche, garantendo una migliore organizzazione. La struttura della cartella dei test è illustrata nell'immagine seguente:

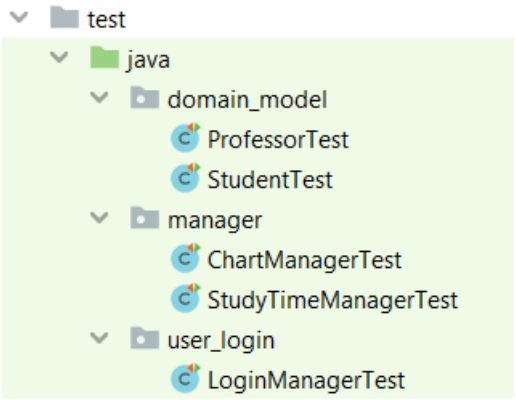

Figura 19: Struttura della cartella dei test

Vengono di seguito riportati esempi significativi, selezionati in base alla funzionalità più rappresentativa di ciascun package di test. Sottolineiamo infine l'importanza dei metodi *setUp()* e *tearDown()* presenti in ogni classe di test. Questi metodi consentono di inizializzare la connessione al database e rimuovere i campi di test inseriti, garantendo così la corretta gestione delle operazioni di avvio e chiusura della connessione alla base di dati. Di seguito vediamo un esempio di implementazione dei due metodi di configurazione:

```
@Before
public void setUp(){
    conn = DBConnection.connect( testPath: "../database/unicoachdb.db");
    loginManager = new LoginManager( testDbPath: "../database/unicoachdb.db");
}
```

Figura 20: *setUp()* LoginManager test

```
@After
public void tearDown() throws SQLException {

    //Cancella l'utente inserito
    String deleteUserSql = "DELETE FROM Utente WHERE Nome = ?";
    PreparedStatement deleteUserStatement = conn.prepareStatement(deleteUserSql);
    deleteUserStatement.setString( parameterIndex: 1,  x: "testNome");

    deleteUserStatement.executeUpdate();
    deleteUserStatement.close();

    if (conn != null) {
        conn = DBConnection.disconnect();
    }
}
```

Figura 21: *tearDown()* LoginManager test

4.1 Test registrazione utente

Il seguente test serve a verificare il corretto inserimento dell'utente all'interno del database al fine di registrarne i dati e le informazioni per il successivo login nell'applicazione.

```
public void testAddUser() throws SQLException {
    Professor professor = new Professor( id: "12121",  name: "testNome",  surname: "testCognome");

    // Prepara la password simulata
    String simulatedInput = "password123\n";
    InputStream inputStream = new ByteArrayInputStream(simulatedInput.getBytes());
    System.setIn(inputStream);

    loginManager.addUser(professor);

    //Verifica che siano aggiunti i due utenti
    String sql = "SELECT * FROM Utente WHERE Email = ? and Password = ?";
    PreparedStatement statement = conn.prepareStatement(sql);
    statement.setString( parameterIndex: 1, professor.getEmail());
    statement.setString( parameterIndex: 2, professor.getPassword());

    ResultSet result = statement.executeQuery();

    assertTrue(result.next());
    assertEquals( expected: "testNome", result.getString( columnLabel: "Nome"));
    assertEquals( expected: "testCognome", result.getString( columnLabel: "Cognome"));
    assertEquals(professor.getEmail(), result.getString( columnLabel: "Email"));
    assertEquals(professor.getPassword(), result.getString( columnLabel: "Password"));

    statement.close();
}
```

Figura 22: Test registrazione utente

4.2 Test inserimento voti

Il seguente test verifica il corretto inserimento del voto da parte del docente allo studente, tramite il metodo *setGrade()* ed il conseguente funzionamento del metodo che ritorna il voto inserito tramite query al database detto *getGrade()*.

```
@Test
public void testSetGetGrade() throws SQLException, MessagingException {
    Professor professorTest = new Professor( id: "12345",  name: "TestNome1",  surname: "TestCognome1",  email: "professor.unicoach@gmail.com"
    Student studentTest = new Student( id: "12345",  name: "TestNome2",  surname: "TestCognome2",  email: "unicoach2023@gmail.com");
    Course courseTest = new Course( name: "TestCorso",  CFU: 6, professorTest, ExamType.WRITTEN_AND_ORAL_TEST);

    Controller professorController = new Controller(professorTest);
    Controller studentController = new Controller(studentTest);

    LoginManager loginManager = new LoginManager( testDbPath: "../database/unicoachdb.db");

    // Simuliamo l'inserimento della password dell'utente
    String input = "knpjdyxlkuzjetfx\n";
    InputStream in = new ByteArrayInputStream(input.getBytes());
    System.setIn(in);
    ByteArrayOutputStream outContent = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outContent));

    loginManager.addUser(professorTest);

    // Colleghiamo il corso allo studente
    List<Course> courseList = new ArrayList<>();
    courseList.add(courseTest);
    studentController.getStudentGateway().linkStudentToCourse(courseList, studentTest);

    professorController.setGrade(studentTest,  grade: 22,  data: "dataTest",  sendEmail: true);
    int grade = professorController.getGrade(studentTest);

    conn = DBConnection.connect( testPath: "../database/unicoachdb.db");
      // Verifica che l'esame venga inserito correttamente
      String sql = "SELECT Voto FROM Esame WHERE Corso = ?";

      PreparedStatement statement = conn.prepareStatement(sql);
      statement.setString( parameterIndex: 1, courseTest.getId());

      ResultSet result = statement.executeQuery();

      assertTrue(result.next());
      assertEquals(grade, result.getInt( columnLabel: "Voto"));

      statement.close();

}
```

Figura 23: Test inserimento del voto allo studente

4.3 Test calcolo media e visualizzazione libretto

In questo test, dedicato alla classe Student, si verifica il corretto funzionamento del metodo *getStudentAvg()* il quale calcola la media pesata dei voti dello studente. Successivamente si esegue anche il display del libretto.

```
@Test
public void testAverageAndDisplayTranscript() throws SQLException, MessagingException {
    Student student = new Student( id: "12345",  name: "TestNome",  surname: "TestCognome");
    Professor professor = new Professor( id: "12345",  name: "TestNome",  surname: "TestCognome");
    Professor professor2 = new Professor( id: "12346",  name: "TestNome",  surname: "TestCognome");
    Course courseTest1 = new Course( name: "TestCorso1",  CFU: 6, professor, ExamType.WRITTEN_AND_ORAL_TEST);
    Course courseTest2 = new Course( name: "TestCorso2",  CFU: 6, professor2, ExamType.WRITTEN_AND_ORAL_TEST);

    Controller professorController = new Controller(professor);
    Controller professorController2 = new Controller(professor2);
    Controller studentController = new Controller(student);

    // Simuliamo l'input utente con tutti i courseTest.getId()
    String input = courseTest1.getId()  + "\n" + courseTest2.getId() + "\n0";
    InputStream in = new ByteArrayInputStream(input.getBytes());
    System.setIn(in);
    ByteArrayOutputStream outContent = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outContent));

    studentController.chooseCourses();

    professorController.setGrade(student,  grade: 25,  data: "dataTest",  sendEmail: false);
    professorController2.setGrade(student,  grade: 24,  data: "dataTest",  sendEmail: false);

    float average = ( 25 * 6 + 24 * 6 ) / 12f;
    assertEquals(average, studentController.getStudentAvg(),  delta: 0.0001f);

    studentController.displayUniTranscript();
}
```

Figura 24: Test calcolo media dello studente

4.4 Test creazione e visualizzazione dei grafici

Per quanto riguarda i test sulla visualizzazione dei grafici, la valutazione della correttezza avviene esaminando direttamente la struttura dei grafici stessi. Di seguito sono presentati i grafici che illustrano gli andamenti degli studenti, dei corsi e dello study-time.

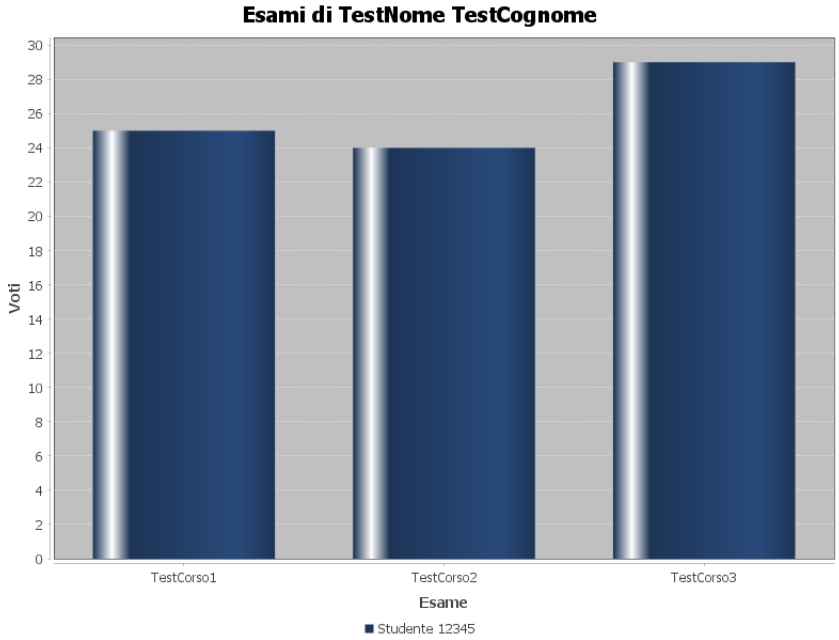

Figura 25: Grafico voti esami dello studente

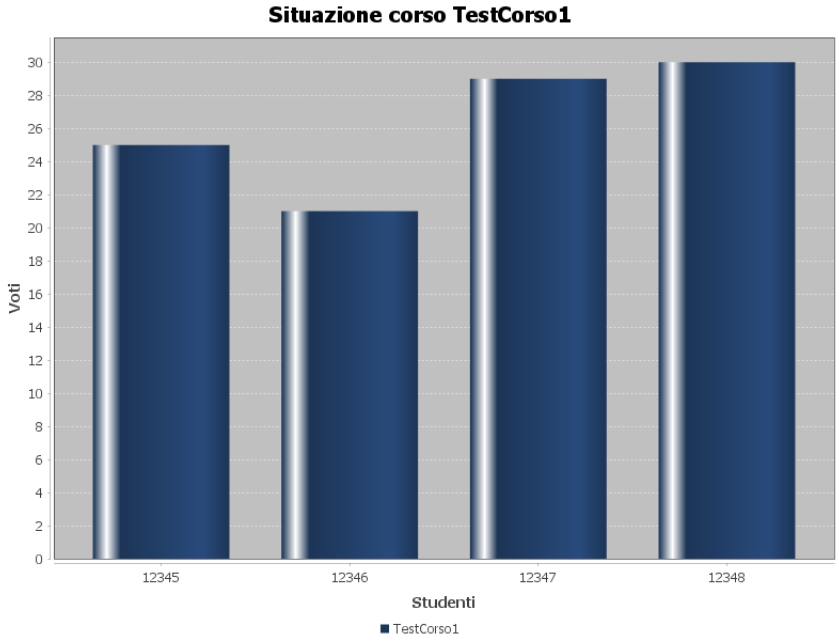

Figura 26: Grafico media degli esami del corso del professore

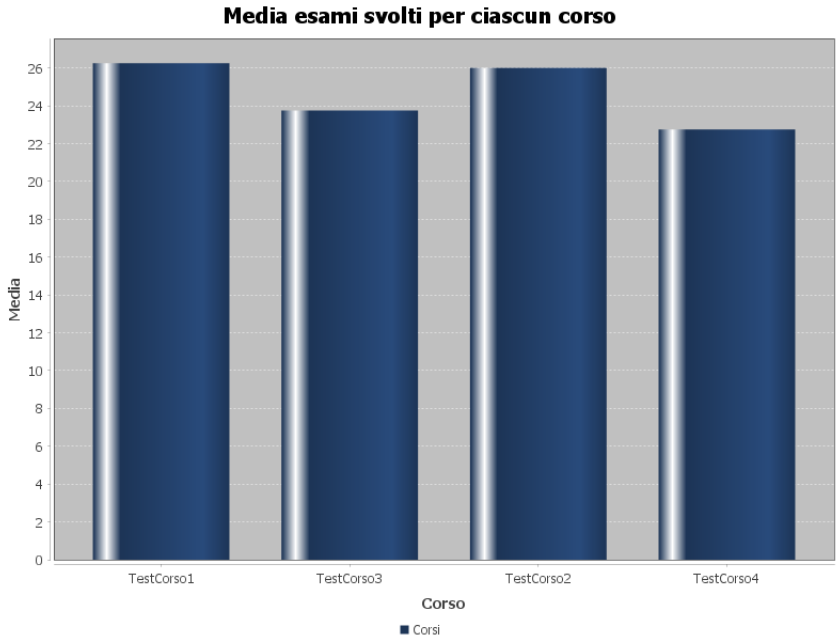

Figura 27: Grafico andamento degli altri corsi

Di seguito sono presentati due grafici relativi alle informazioni sullo study time. Il primo grafico rappresenta il tempo di studio degli studenti in relazione alla media dei voti del corso, accessibile al professore. Il secondo grafico mostra il tempo di studio suddiviso per categoria in relazione al voto dell'esame per ciascun corso, accessibile sia agli studenti che al docente.

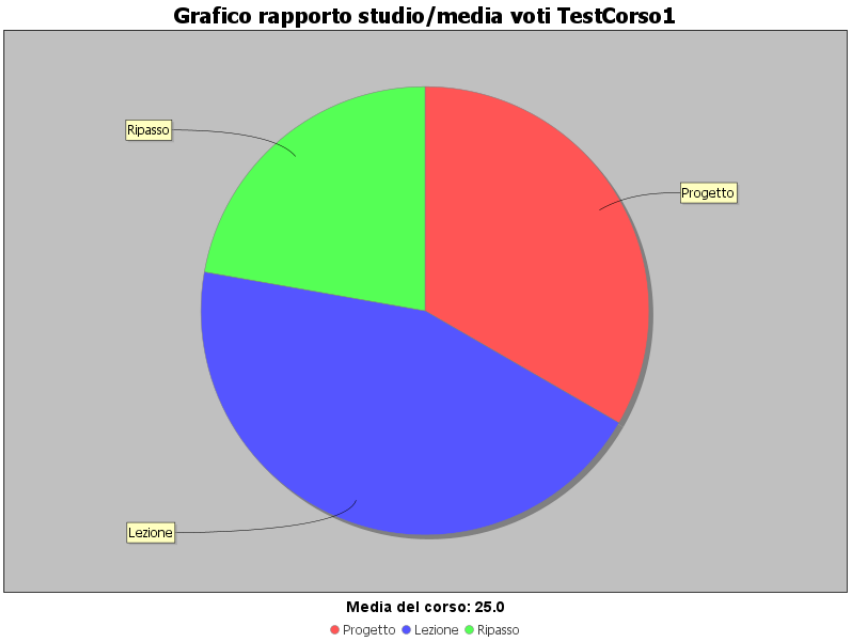

Figura 28: Grafico study time degli studenti in relazione alla media dei voti del corso

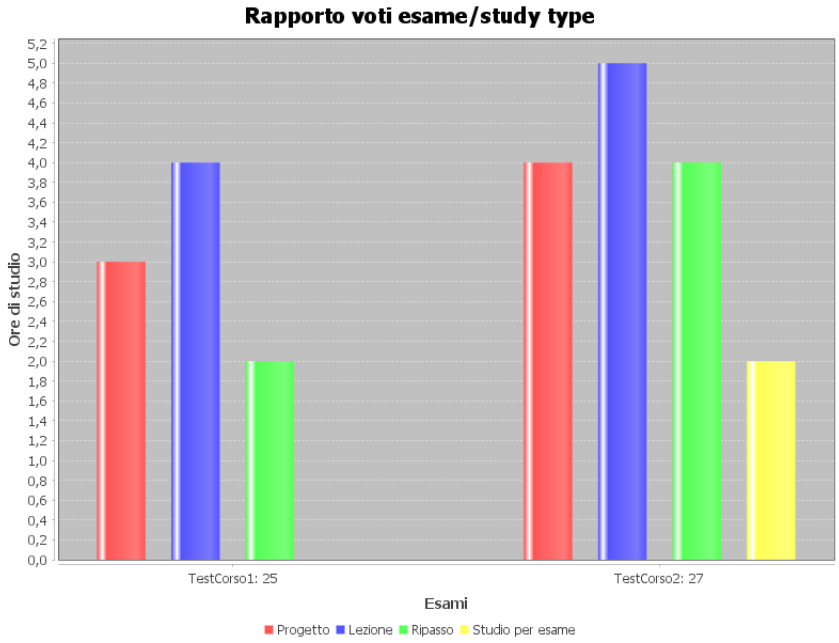

Figura 29: Grafico study time suddiviso per categoria in relazione al voto dell'esame per ciascun corso